How to
Reverse Engineer Web Applications

First things first

Prerequisites

- **node.js** (web tools are made with node first)
- **npm** (comes with node)

Nice-to-haves

- **Visual Studio code** (or suitable code editor)
- **git** (because it's better than cvs)

First things first

Lab repository

https://github.com/jsoverson/workshop-reverse-engineering

-or-

Zipfile: http://bit.ly/reveng-webapps

How to Reverse Engineer Web Applications

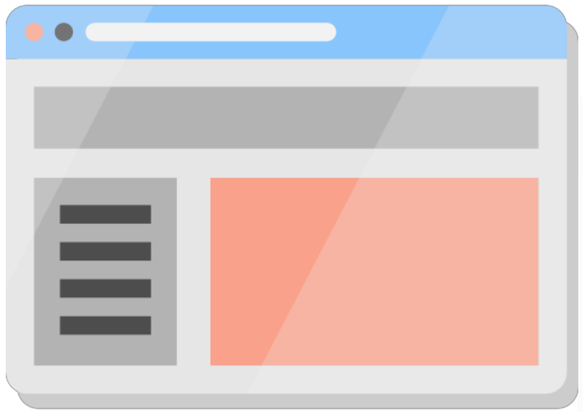
https://github.com/jsoverson/workshop-reverse-engineering

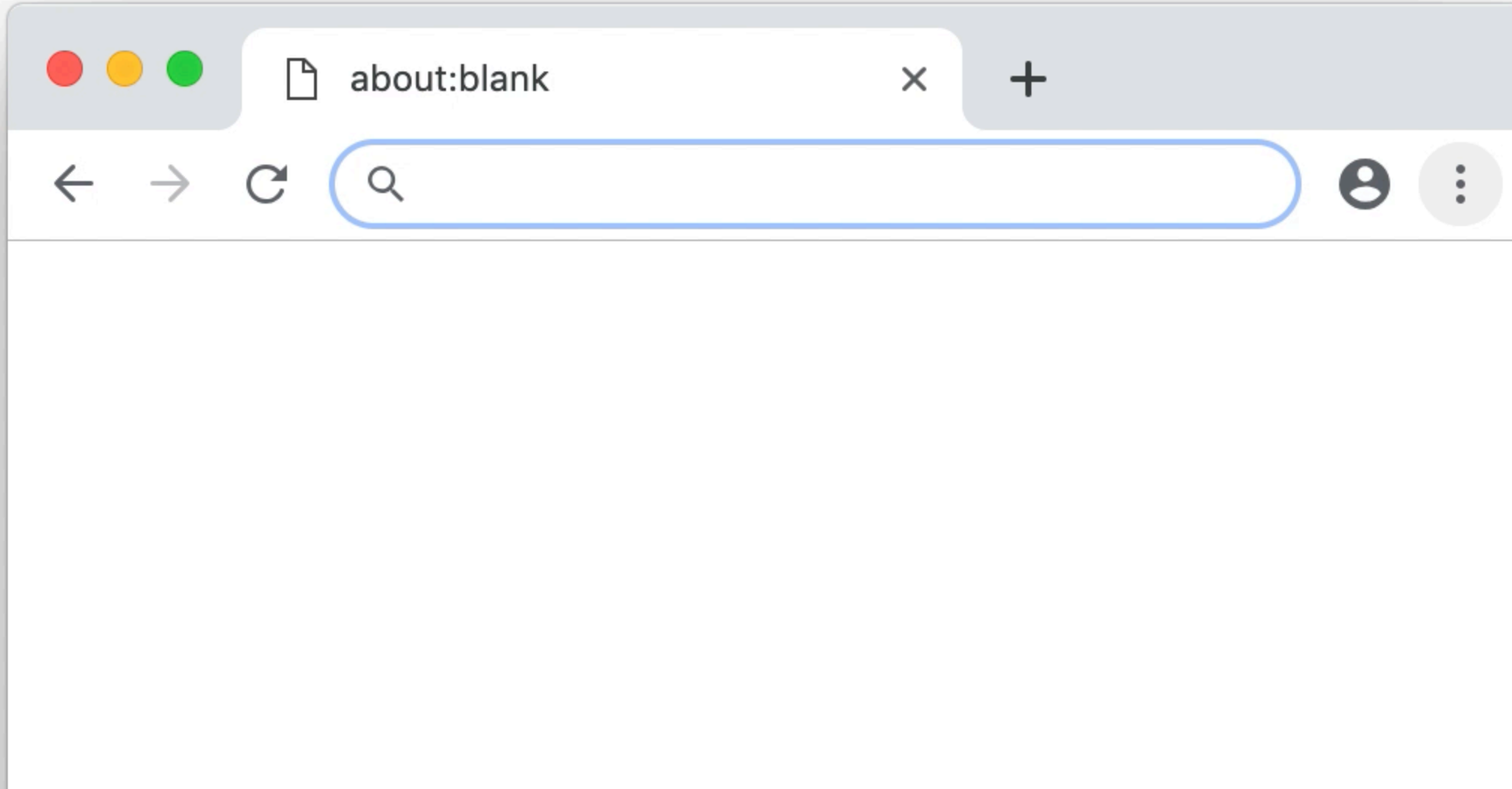
Challenge #1

1. Web applications have grown extremely complex.

https://github.com/jsoverson/workshop-reverse-engineering

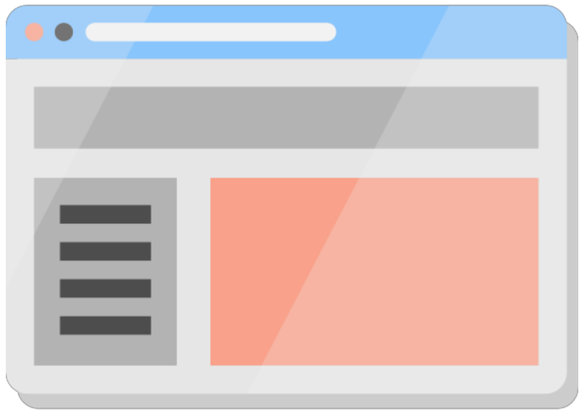
Web 101





https://github.com/jsoverson/workshop-reverse-engineering

Web 101



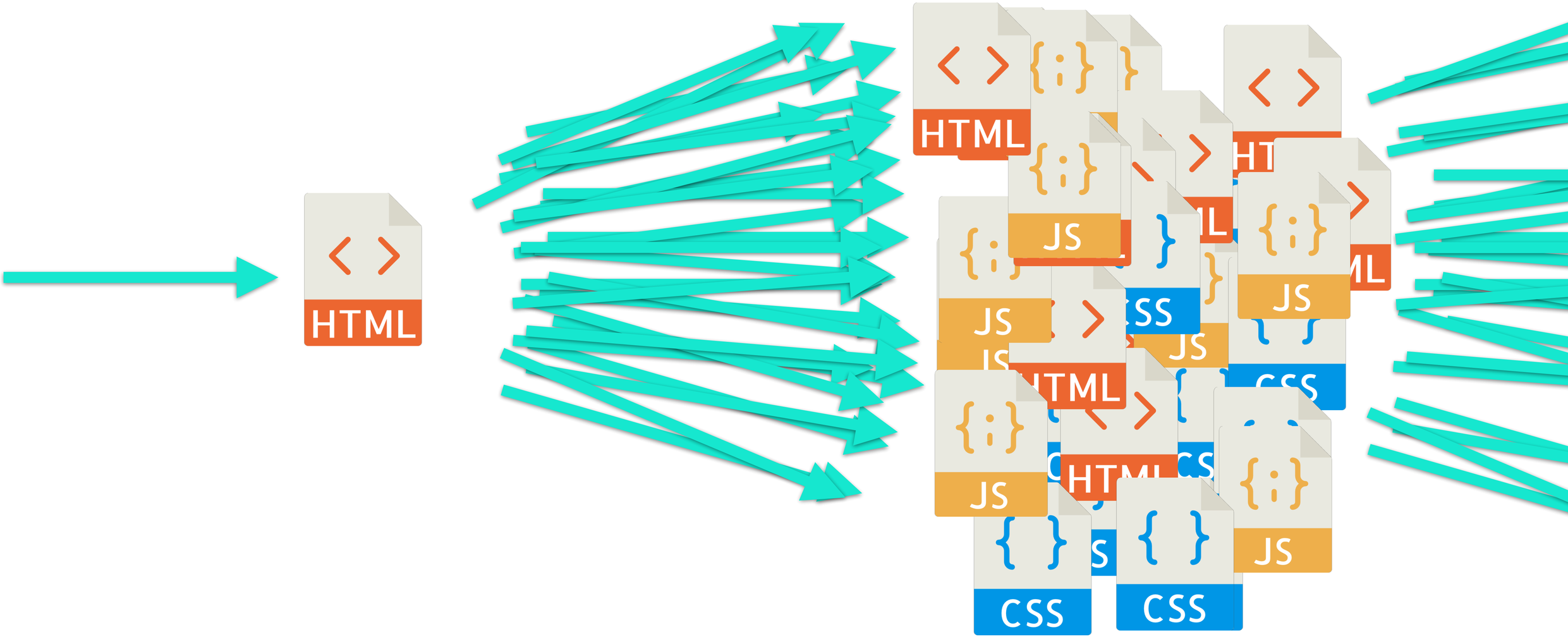
```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
User-Agent: Chrome/72.0.3626.96 Safari/537.36 (…)
```


```
HTTP/1.1 200 OK
Date: Tue, 19 Feb 2019 21:34:08 GMT
Content-Type: text/html
Content-Length: 1932

<!doctype HTML>(…)
```

https://github.com/jsoverson/workshop-reverse-engineering

Web 101



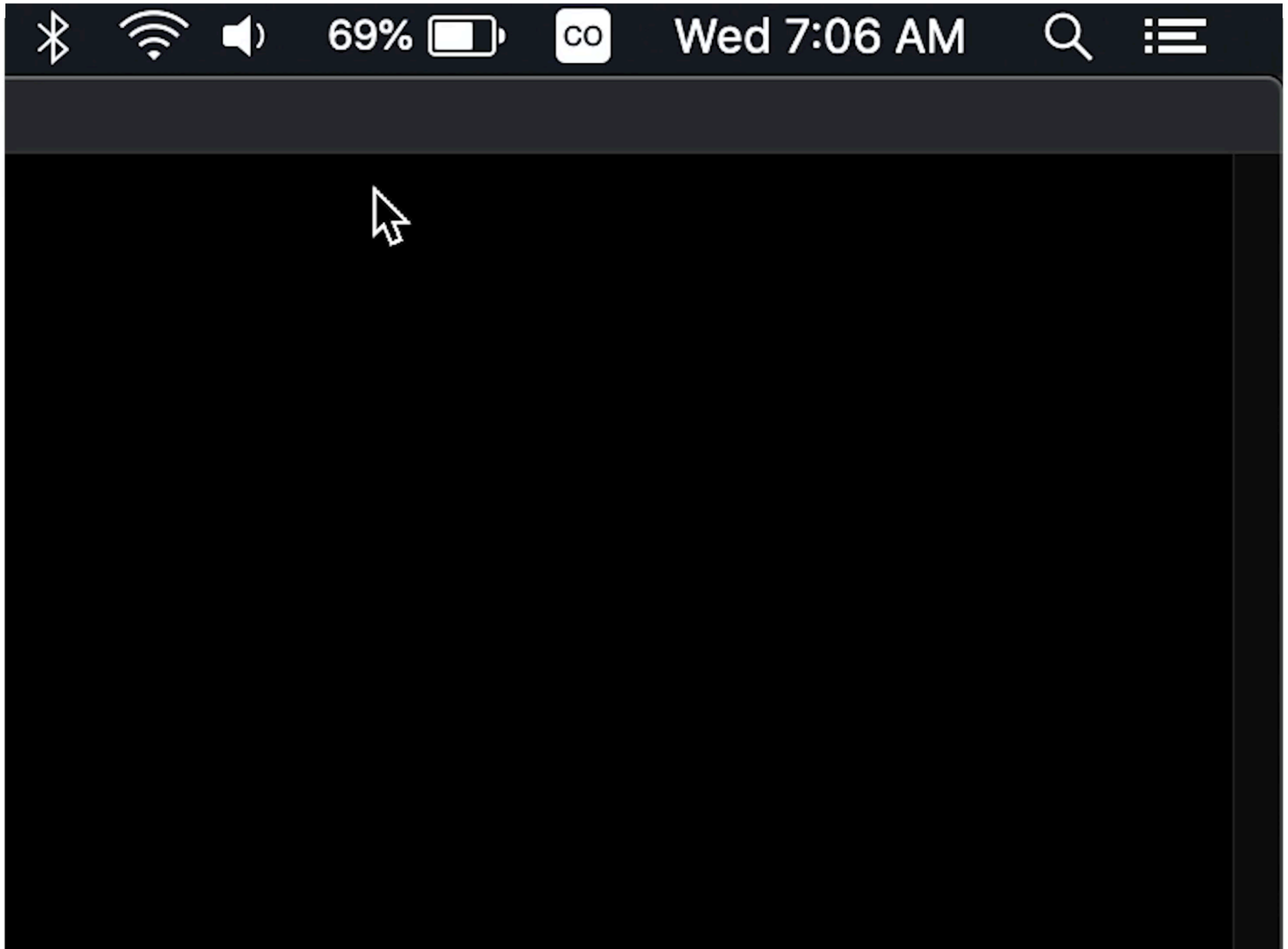
https://github.com/jsoverson/workshop-reverse-engineering

Challenge #2

1. Web applications have grown extremely complex.
2. Limited ability to run old versions of resources or use old APIs.

https://github.com/jsoverson/workshop-reverse-engineering

99% of websites assume connectivity



Challenge #3

1. Web applications have grown extremely complex.
2. Limited ability to run old versions of resources or use old APIs.
3. Web site resources can change *frequently*.

https://github.com/jsoverson/workshop-reverse-engineering

Going to Market Faster: Most Companies Are Deploying Code Weekly, Daily, or Hourly

https://github.com/jsoverson/workshop-reverse-engineering

Etsy Deploy Stats: 2012

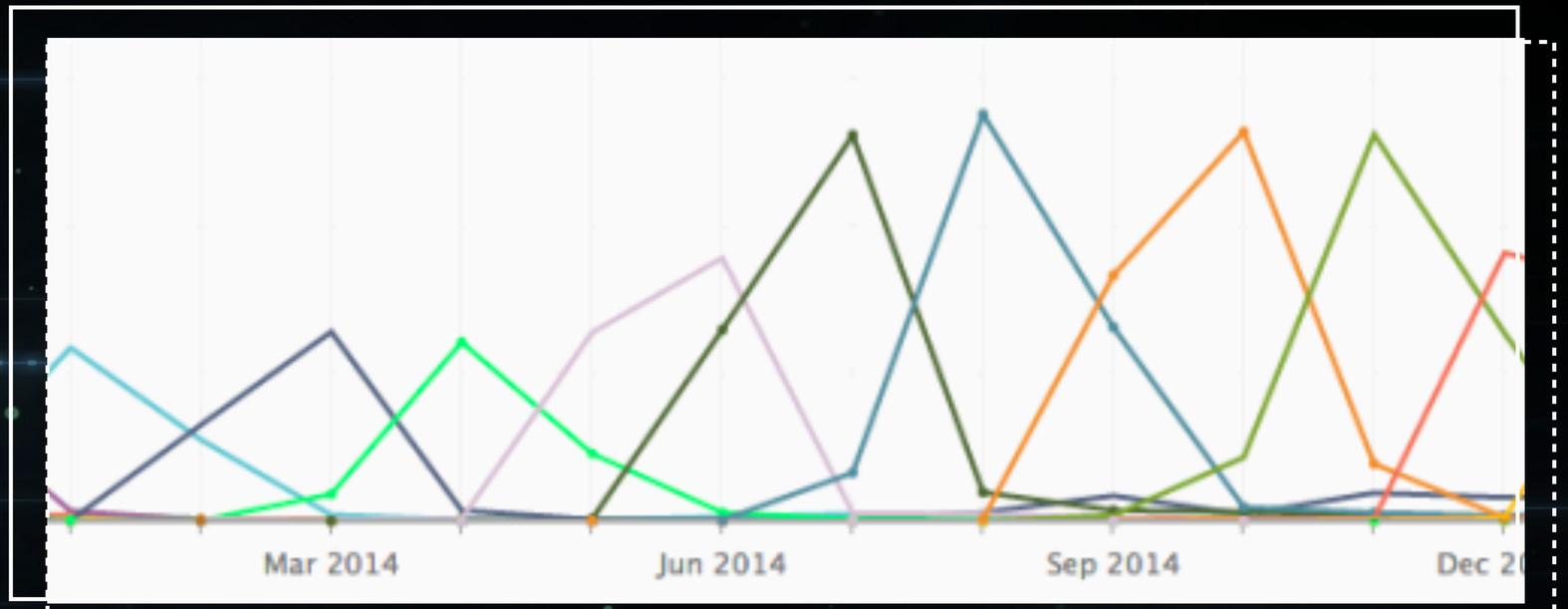
- Deployed to production 6,419 times
- On average, 535/month, 25/day

Challenge #4

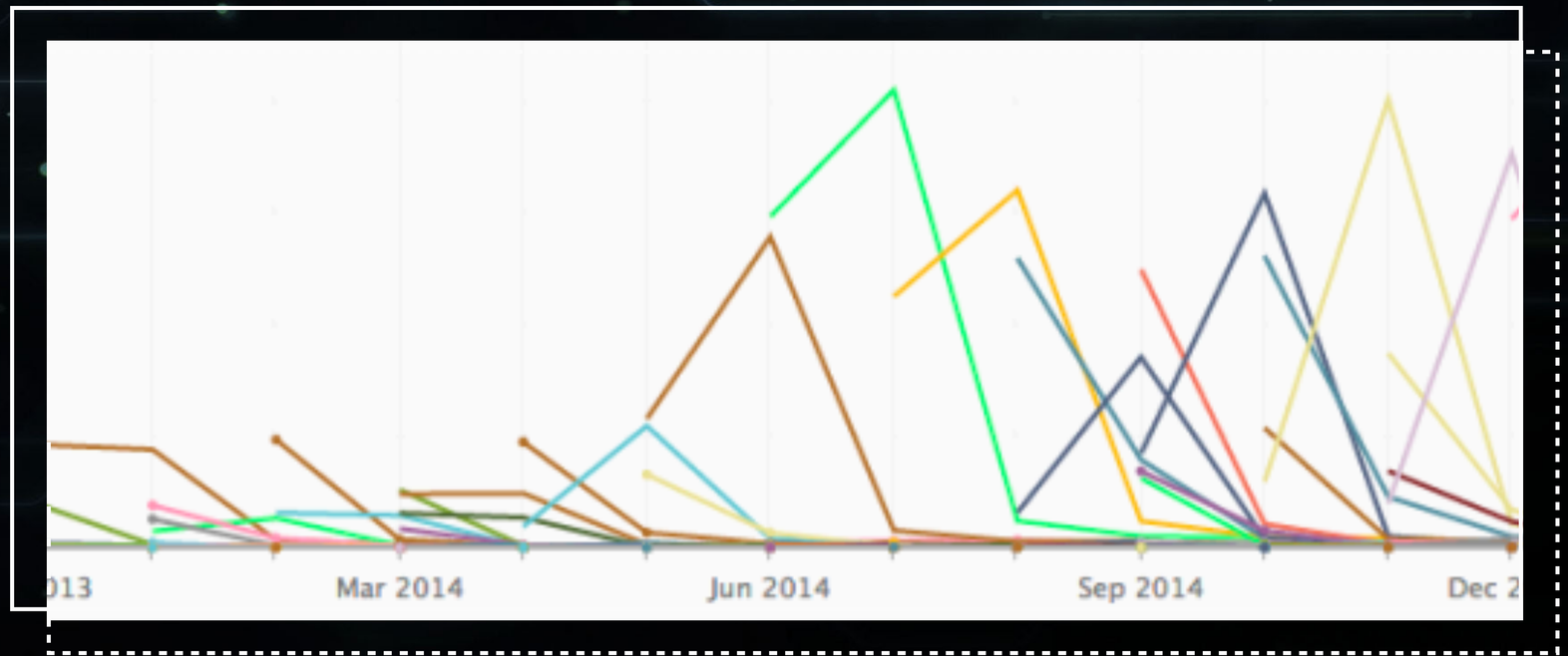
1. Web applications have grown extremely complex.
2. Limited ability to run old versions of resources or use old APIs.
3. Web sites and APIs can change *frequently*.
4. Web browsers change *frequently*.

https://github.com/jsoverson/workshop-reverse-engineering

A year of Chrome



A year of FireFox



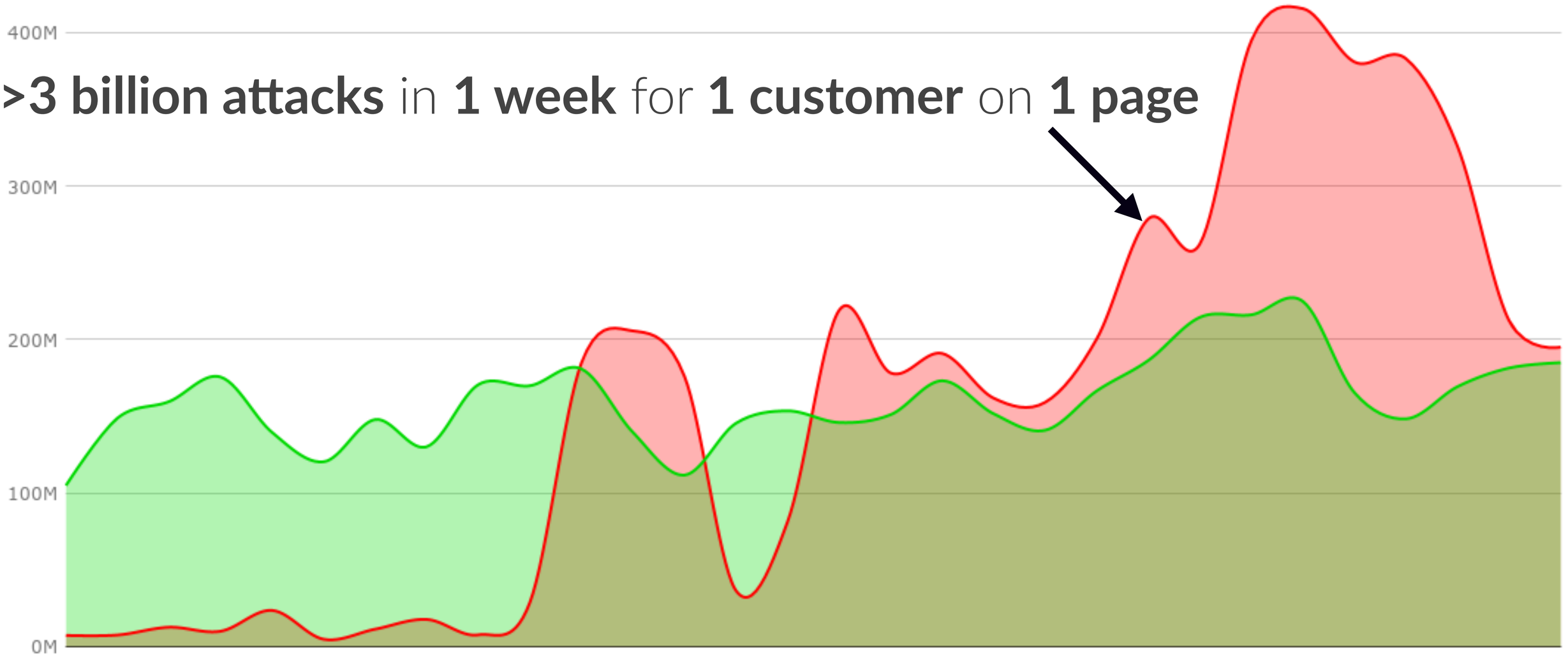
https://github.com/jsoverson/workshop-reverse-engineering

Challenge #5

1. Web applications have grown extremely complex.
2. Limited ability to run old versions of resources or use old APIs.
3. Web sites and APIs can change *frequently*.
4. Web browsers change *frequently*.
5. Actual attackers are leading to more effective countermeasures.

https://github.com/jsoverson/workshop-reverse-engineering

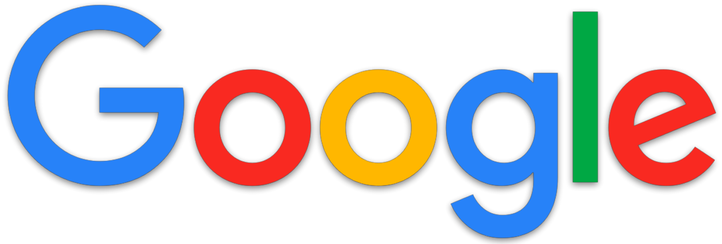
Website attacks are a serious problem



>3 billion attacks in **1 week** for **1 customer** on **1 page**

https://github.com/jsoverson/workshop-reverse-engineering

And have an industry around protecting them



https://github.com/jsoverson/workshop-reverse-engineering

And have an industry around protecting them

Google

distil networks

Akamai

perimeterX

So how do you hack web apps?

1. Drive the browser programmatically
2. Simulate and intercept "system" calls
3. Reuse as much application code as possible

Lab work

Lab `0`

1. Understanding Intent in JavaScript

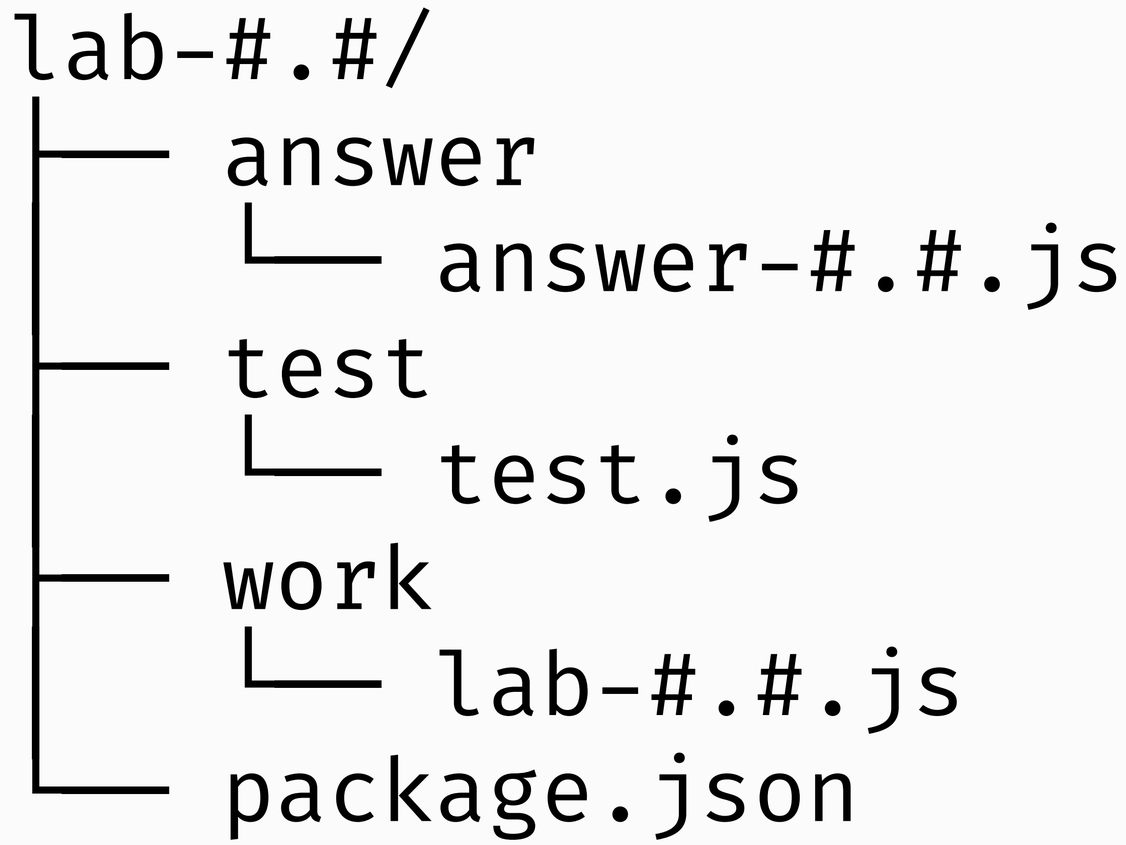
Lab 1

1. Extracting logic
2. Extracting logic resiliently
3. Transforming with scope awareness

Lab 2

1. Automating a browser
2. Intercepting requests
3. Modifying responses
4. Rewriting JS on the fly with a mocked environment

Lab Format



Node/npm basics

```
node [script.js]

npm install

npm install [specific package]
```

https://gitpod.io/

Gitpod BETA

Features Business Docs Blog Login

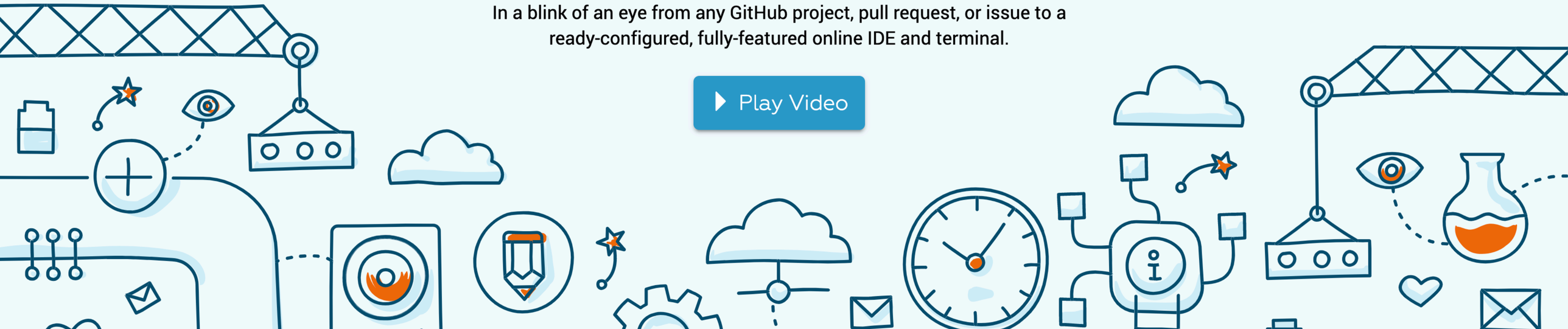
Stay in the flow!

One-Click Online IDE for GitHub

and other code hosting platforms.

In a blink of an eye from any GitHub project, pull request, or issue to a ready-configured, fully-featured online IDE and terminal.

Play Video



https://try-puppeteer.appspot.com/

Try Puppeteer v1.9.0 Run an example: screenshot.js

```
1 const browser = await puppeteer.launch();
2
3 const page = await browser.newPage();
4 await page.goto('https://example.com');
5
6 console.log(await page.content());
7 await page.screenshot({path: 'screenshot.png'});
8
9 await browser.close();
10
```

RUN IT

LOG

Lab 0.1

Understanding Intent in JavaScript

Background

Payload A is the first script of three found in an exploited dependency of npm package event-stream.

Goal

De-obfuscate payload-A.js and identify how it is loading the second payload.

Lab 0.1

Understanding Intent in JavaScript

Tips

- Format your code with ⌘+⇧+P or ^+⇧+P to open command palette then "Format Document"
- Rename variables by pressing F2 when cursor is over an identifier.
- There is a helper file that includes additional encoded strings.
  (The first and second strings in the helper file are beyond the scope of this lab.)
- process.env contains the environment variables at time of execution

Lab Series 1

Programmatically manipulating JavaScript

Lab 1.1

Programmatically extract logic from JavaScript

Background

Common JavaScript best practices and bundlers produce code that has a minimal public footprint, limiting the ability to hook into existing logic.

Goal

Use the Shift parser and code generator to **read payload-A.js** and **extract its e function** and **export it as an unhex method** in our node script.

Lab 1.1

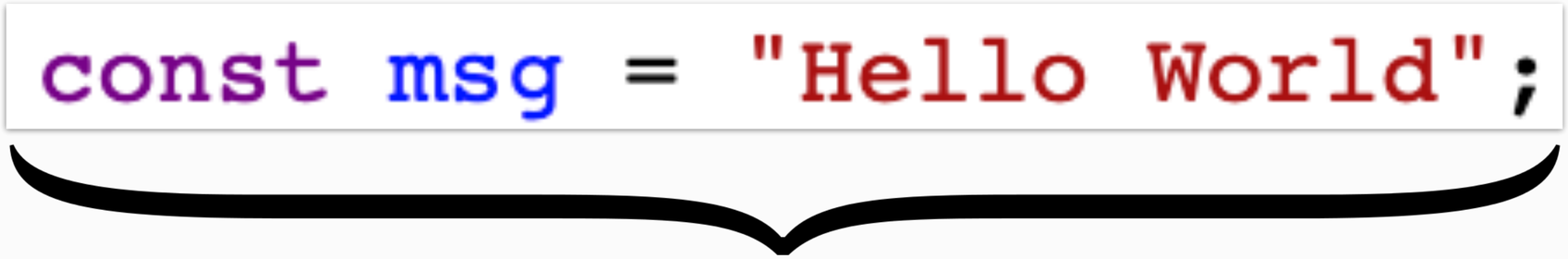
Using parsers

```
parse(originalSource) → Abstract Syntax Tree (AST)

codegen(AST) → newSource
```

Lab 1.1

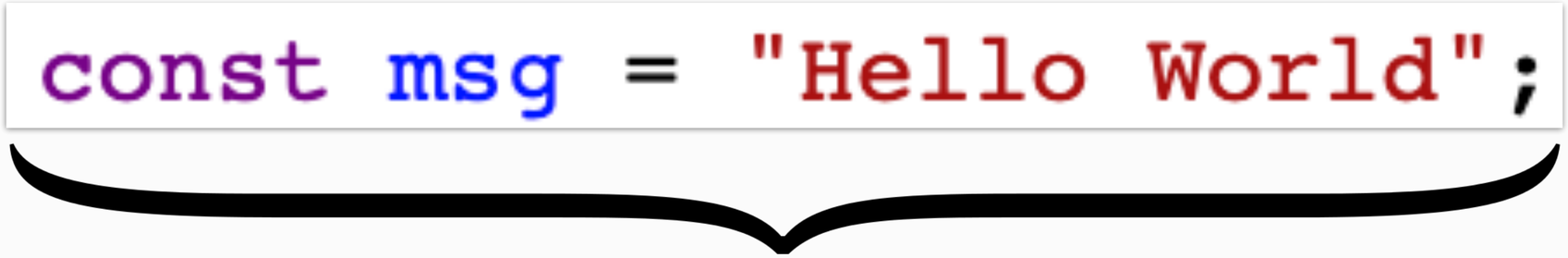
What is an AST?



VariableDeclarationStatement

Lab 1.1

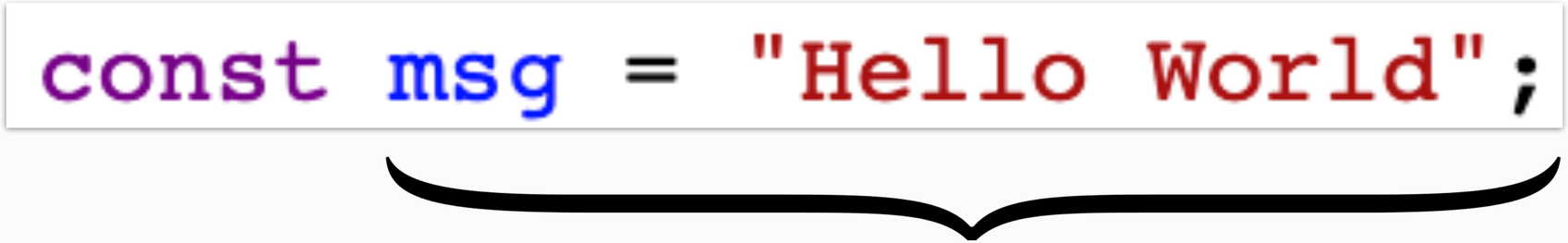
What is an AST?



`VariableDeclaration`

Lab 1.1

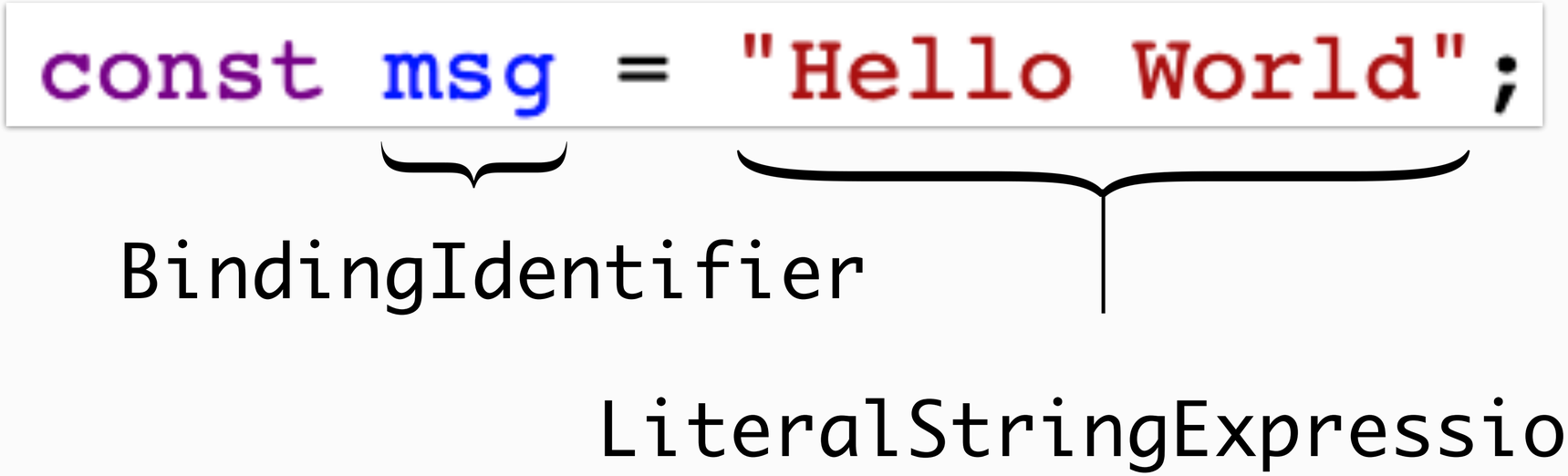
What is an AST?



VariableDeclarator

Lab 1.1

What is an AST?



Lab 1.1

What is an AST?

```
1 const msg = "Hello World";
2
3 log(msg);
```

```
{
  "type": "Script",
  "directives": [],
  "statements": [
    {
      "type": "VariableDeclarationStatement",
      "declaration": {
        "type": "VariableDeclaration",
        "kind": "const",
        "declarators": [
          {
            "type": "VariableDeclarator",
            "binding": {
              "type": "BindingIdentifier",
              "name": "msg"
            },
            "init": {
              "type": "LiteralStringExpression",
              "value": "Hello World"
            }
          }
        ]
      }
```

Lab 1.1

What is an AST?

```
1 const msg = "Hello WOPRs";
2
3 log(msg);
```

Lab 1.1

Programmatically extract logic from JavaScript

Background

Common JavaScript best practices and bundlers produce code that has a minimal public footprint, limiting the ability to hook into existing logic.

Goal

Use the Shift parser and code generator to **read payload-A.js** and **extract its e function** and **export it as an unhex method** in our node script.

Lab 1.1

Programmatically extract logic from JavaScript

Tips

- Don't overthink it - you're just deeply accessing a property in an object.
- `console.log()` as you make your way down the tree, e.g.

  ```
  console.log(tree.expressions[0]);
  ```
- Copy/paste `payload-A.js` into astexplorer.net for an interactive UI

Lab 1.2

Resiliently extract logic from JavaScript

Background

Extracting and manipulating JavaScript requires that the code be resilient to changes in the input source.

Goal

Use a traversal method to pick out the same function from lab 1.1 by inspecting the AST node of the function and identifying it by its shape or attributes.

Lab 1.2

Resiliently extract logic from JavaScript

Tips

- `allNodes` contains a list of all nodes in the AST.
- You can iterate over that list and check every node for properties that represent the node you want to target.
- You probably want to check if `node.type === 'FunctionDeclaration'`
- You can then check the function name to make sure it is equal to `'e'`

Lab 1.3

Rewrite JavaScript taking scope and context into account

Background

Rewriting JavaScript requires tools that are aware of the entire program's scope and context.

Goal

Use `shift-scope` to rename global variables "`a`" and "`b`" to "`first`" and "`second`"

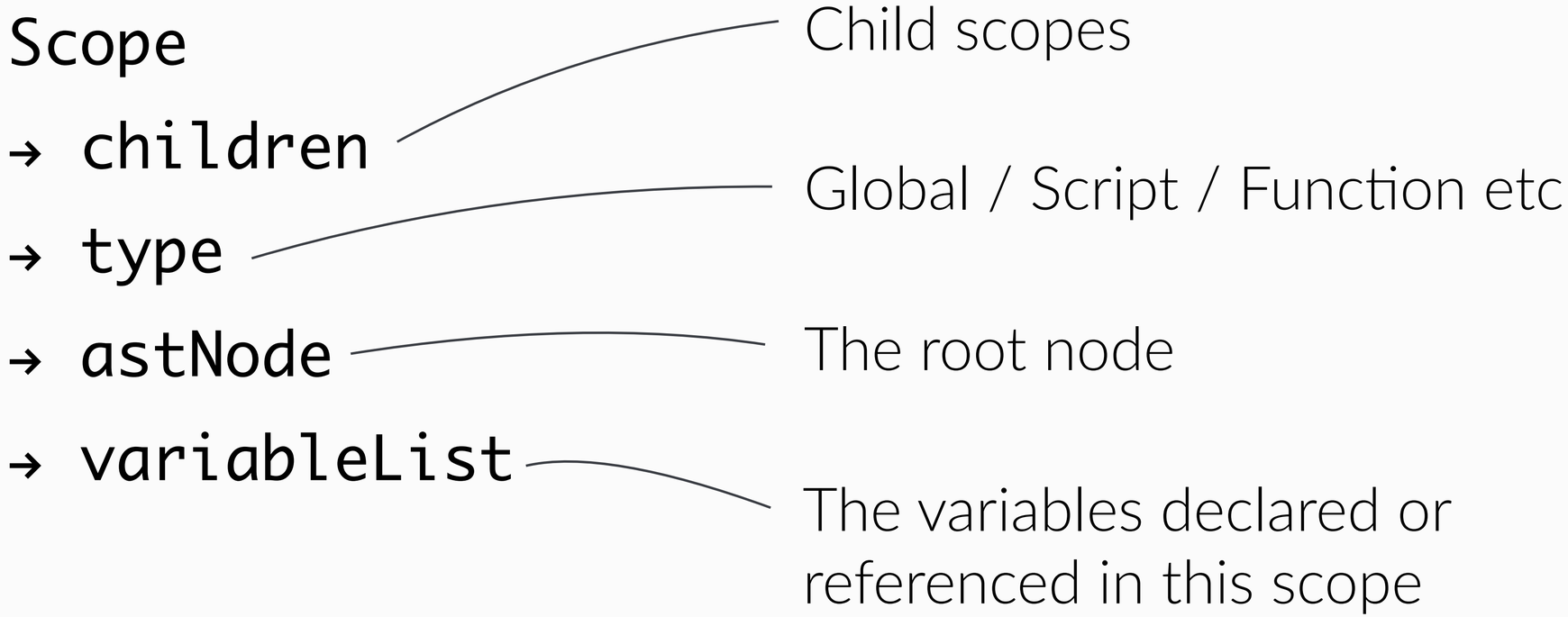
Analyzing Scope

```
1
2   /* rename a to first and b to second */
3   const a = "Hello";
4   const b = "World";
5
6   function join(a, b) {
7     /* these should not be renamed */
8     return a + b;
9   }
10
11  module.exports = function() {
12    /* these should be renamed */
13    return join(a, b);
14  }
15
```

Analyzing Scope

```
const scope = analyzeScope(AST)
```

Analyzing Scope



Analyzing Scope

```
const lookupTable = new ScopeLookup(scope)

const identifier = /* node from AST */

const lookup = lookupTable.variableMap.get(identifier)
```

Lab 1.3

Rewrite JavaScript taking scope and context into account

Background

Rewriting JavaScript requires tools that are aware of the entire program's scope and context.

Goal

Use `shift-scope` to rename global variables "`a`" and "`b`" to "`first`" and "`second`"

Lab 1.3

Rewrite JavaScript taking scope and context into account

Tips

- The argument to `lookupTable.variableMap.get()` should be the identifier node itself, not a string.
- There are variables already defined that reference the AST nodes.
- Each lookup returns a list of entries with declarations and references. You need to change *both* declarations and references to fully rename an identifier.

Lab Series 2

Programmatically controlling a browser

Lab Series 2

Notes:

The Puppeteer npm package downloads Chrome on every install. If internet is flakey, download it once and copy `node_modules/puppeteer` from one lab to another

Lab 2.1

Programmatically control a browser with Puppeteer

Background

Web application analysis needs to be completely automated, otherwise everything depending on a manual step risks breaking when anything changes.

Goal

Set up a base environment that controls Chrome with Puppeteer and nodejs.

What is Puppeteer?

> Puppeteer is a nodejs library that provides a high-level API to control Chrome over the DevTools Protocol.
>
> Puppeteer runs headless by default, but can be configured to run full (non-headless) Chrome or Chromium.

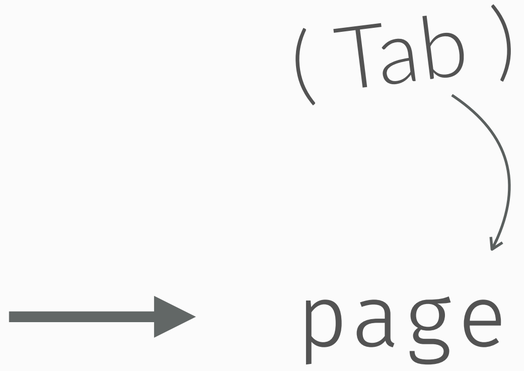
Puppeteer API

```
puppeteer.launch();
puppeteer.connect();
```

```
browser
```

Browser instance

```
browser.pages();
browser.newPage();
```



Page instance

```
page.goto();
page.evaluate();
```

```
page.$();
page.$$();
```

```
element
```

Element instance

```
element.click();
element.type(…);
element.tap();
```

Lab 2.1

Programmatically control a browser with Puppeteer

Tips

- The Puppeteer API is fantastic : http://bit.ly/puppeteer-api
- You need to get a `browser` instance from `puppeteer`
- You need to get a `page` instance from `browser`
- You need to *go to* a url of your choice, e.g. https://example.com

Lab 2.2

Intercept requests via the Chrome Devtools Protocol

Background

Intercepting, inspecting, and modifying inbound and outbound communication is a critical part of any reverse engineering effort.

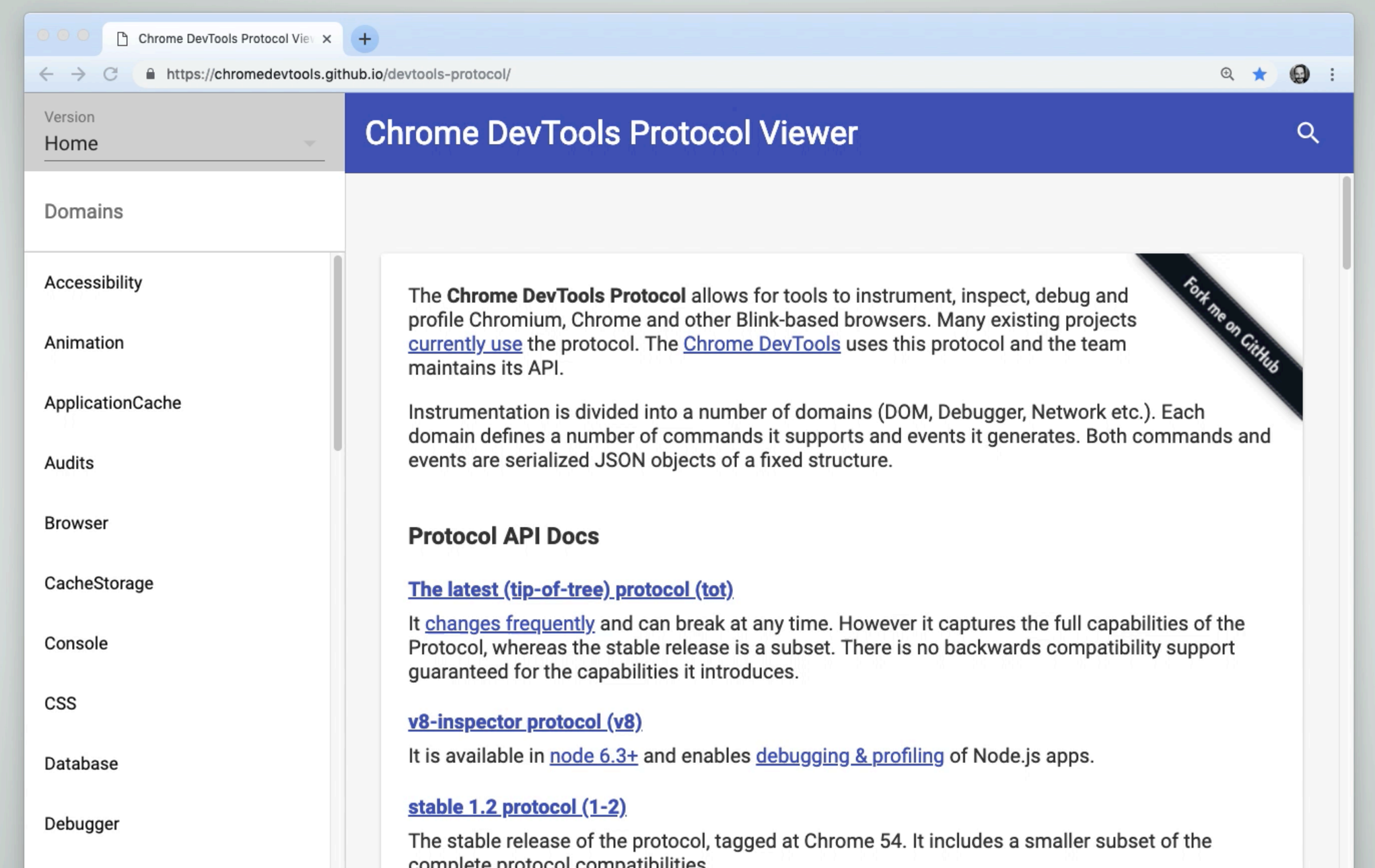
Goal

Use the Chrome Devtools Protocol directly to intercept all Script resources, log the URL to the terminal, and then continue the request.

What is Chrome Devtools Protocol?

> The Chrome DevTools Protocol allows for tools to instrument, inspect, debug and profile Chromium, Chrome and other Blink-based browsers.

http://bit.ly/chrome-devtools-protocol



Initiating a CDP Session

```
const client = page.target().createCDPSession();
```

Sending commands

```
client.send(“command”);
client.send(“command”, {options…});
client.on(“event”, listener);
```

Intercepting Network Traffic

(pseudo code)

```
send(“Network.enable”);
send(“Network.setRequestInterception”, patterns);
on(“Network.requestIntercepted”, (evt) ⇒ {
  /* modify request or response */

  send(
    “Network.continueInterceptedRequest”,
    request
  )
})
```

Lab 2.2

Intercept requests via the Chrome Devtools Protocol

Background

Intercepting, inspecting, and modifying inbound and outbound communication is a critical part of any reverse engineering effort.

Goal

Use the Chrome Devtools Protocol directly to intercept all Script resources, log the URL to the terminal, and then continue the request.

Lab 2.2

Intercept requests via the Chrome Devtools Protocol

Tips

- The Puppeteer API is fantastic : http://bit.ly/puppeteer-api
- This is purely in the `Network` domain
- Remind me to flip back to the pseudo-code if you're stuck.

Lab 2.3

Modify intercepted requests

Background

Modifying intercepted requests requires recreating an entire HTTP response and passing it along as the original.

Goal

Retrieve the original script body and append a `console.log()` statement to the end of the script that simply logs a message.

Intercepting Network Traffic

(pseudo code)

```
send(“Network.enable”);
send(“Network.setRequestInterception”, patterns);
on(“Network.requestIntercepted”, (evt) ⇒ {
  const response =
     send(“Network.getResponseBodyForInterception”, interceptionId);

  send(
    “Network.continueInterceptedRequest”,
    request
  )
})
```

Modifying a Response

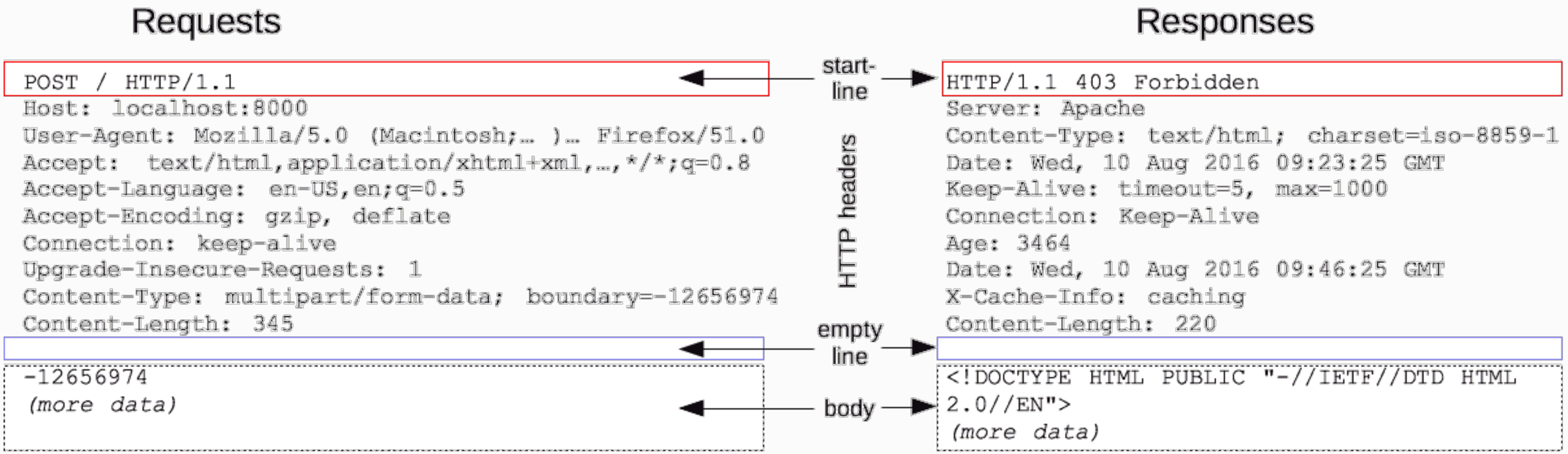
(pseudo code)

```
const response =
    send('Network.getResponseBodyForInterception', interceptionId);

const body = response.base64Encoded
  ? atob(response.body)
  : response.body;

send('Network.continueInterceptedRequest', {
  interceptionId,
  rawResponse: btoa(/* Complete HTTP Response */)
});
```

HTTP Requests & Responses



Lab 2.3

Modify intercepted requests

Background

Modifying intercepted requests requires recreating an entire HTTP response and passing it along as the original.

Goal

Retrieve the original script body and append a `console.log()` statement to the end of the script that simply logs a message.

Lab 2.3

Modify intercepted requests

Tips

- Rely on the Chrome Devtools Protocol documentation : https://bit.ly/chrome-devtools-protocol
- What you add to each script is up to you, it's user choice as long as it is observable.
- The last TODO requires reading the CDP documentation.

Final Lab

Meaningfully rewrite a script to intercept its access to browser APIs

Background

Intercepting a script's access to standard APIs allows you to guide its execution in the direction you want without modifying its internals.

Goal

Wrap the example site's script to intercept access to document.location to make the script think it is hosted elsewhere & inject code that exposes the seed