Web Development

COMP 431 / COMP 531

Lecture 20: Authorization

Instructor: Mack Joyner

Department of Computer Science, Rice University

mjoyner@rice.edu

http://www.clear.rice.edu/comp431

Part II – Back End Development

Quiz #3
Back-End Web Server
Due Thursday 11/9

Homework Assignment 6
(Draft Back-End)
Due Thursday 11/16

COMP 531 Paper
Due Tuesday 11/28

Cookies

```
POST /login
{ username and password }
```

in plain sight!

Server returns a cookie

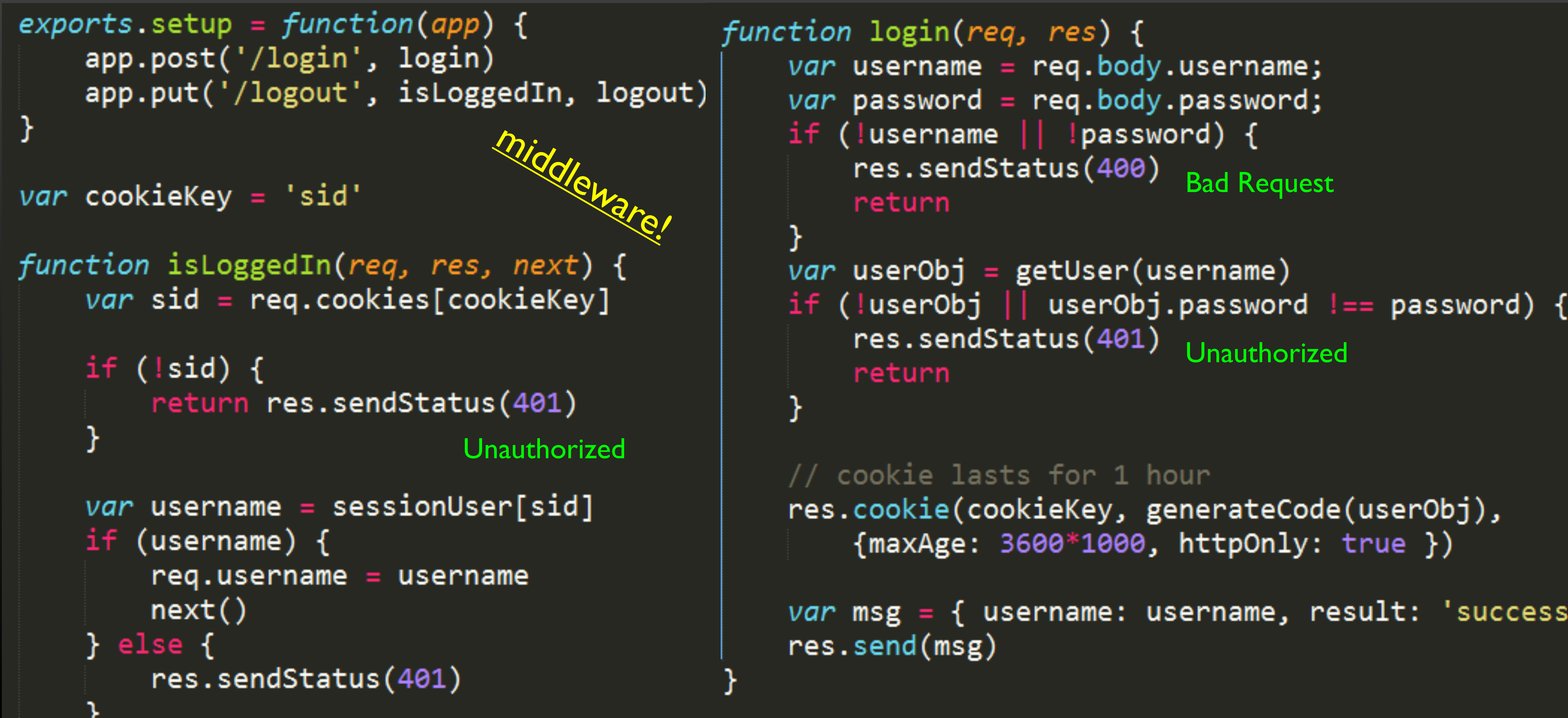
Browser "eats" the cookie and returns it with all subsequent requests

```
PUT /logout
```

Server returns "emptied" cookie for browser to eat

```
npm install cookie-parser –save
var cookieParser = require(‘cookie-parser’)
app.use(cookieParser())
```

What’s it look like in Node?



```
exports.setup = function(app) {
    app.post('/login', login)
    app.put('/logout', isLoggedIn, logout)
}

var cookieKey = 'sid'

function isLoggedIn(req, res, next) {
    var sid = req.cookies[cookieKey]

    if (!sid) {
        return res.sendStatus(401)
    }

    var username = sessionUser[sid]
    if (username) {
        req.username = username
        next()
    } else {
        res.sendStatus(401)
    }
```

middleware!

Unauthorized

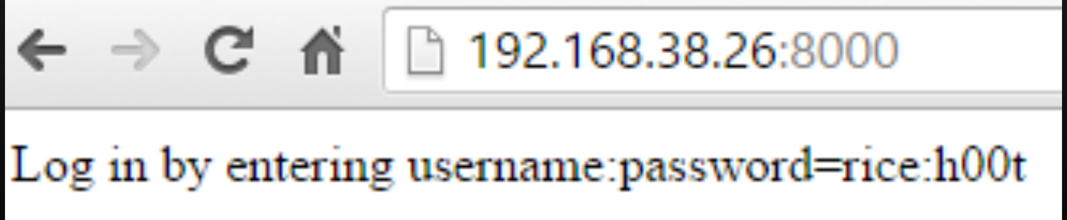
```
function login(req, res) {
    var username = req.body.username;
    var password = req.body.password;
    if (!username || !password) {
        res.sendStatus(400)
        return
    }
    var userObj = getUser(username)
    if (!userObj || userObj.password !== password) {
        res.sendStatus(401)
        return
    }

    // cookie lasts for 1 hour
    res.cookie(cookieKey, generateCode(userObj),
        {maxAge: 3600*1000, httpOnly: true })

    var msg = { username: username, result: 'success
    res.send(msg)
}
```

Bad Request

Unauthorized

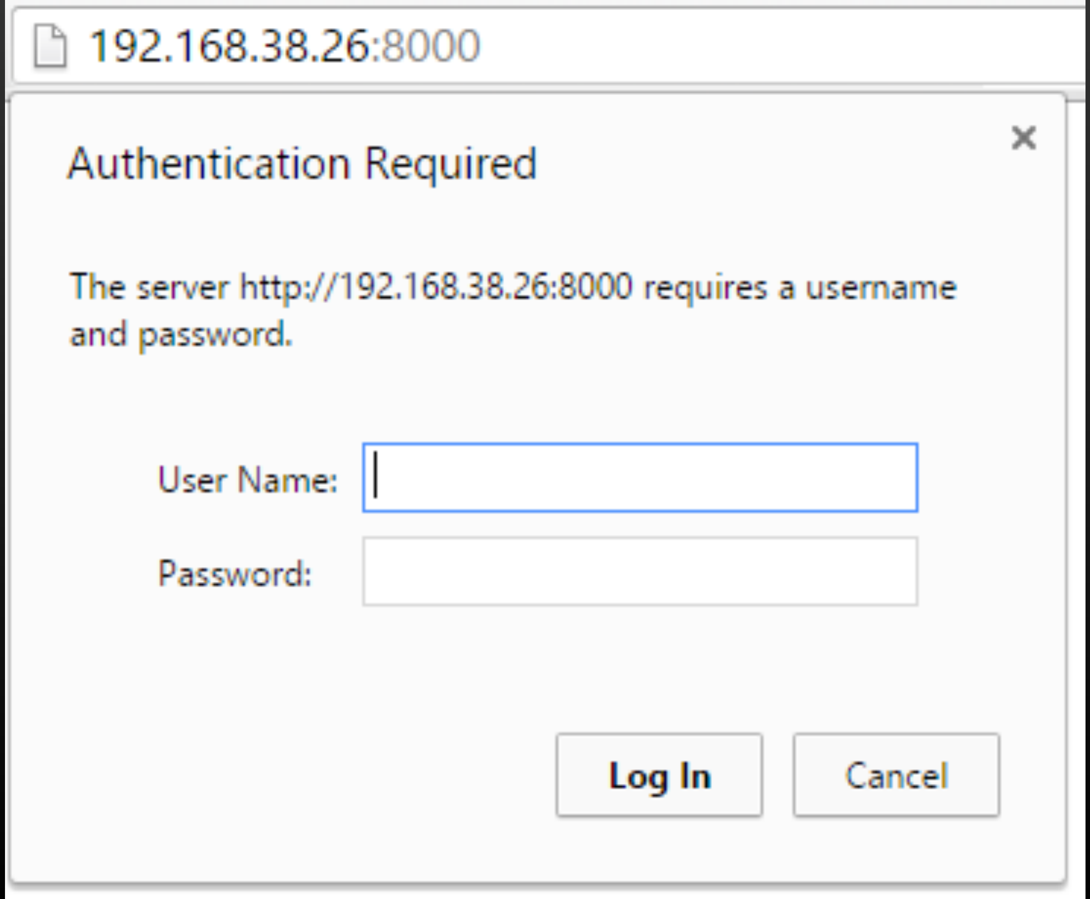
HTTP AUTH

- User makes request without Authorization
- Server responds 401 and sets WWW-Authenticate with a “challenge”
- User attempts challenge by filling in username and password
- Server then accepts or issues another challenge



192.168.38.26:8000

Log in by entering username:password=rice:h00t



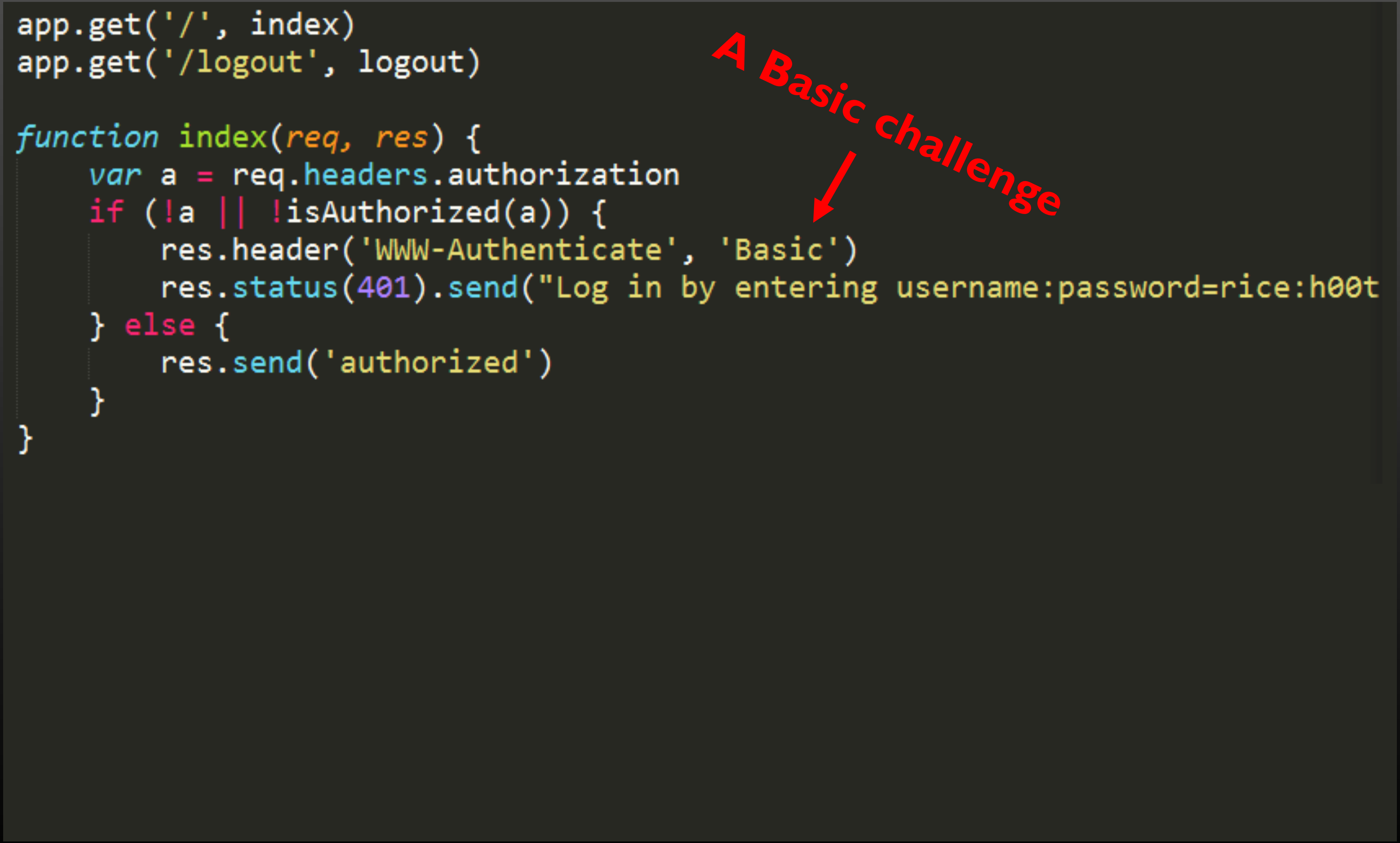
192.168.38.26:8000

Authentication Required

The server http://192.168.38.26:8000 requires a username and password.

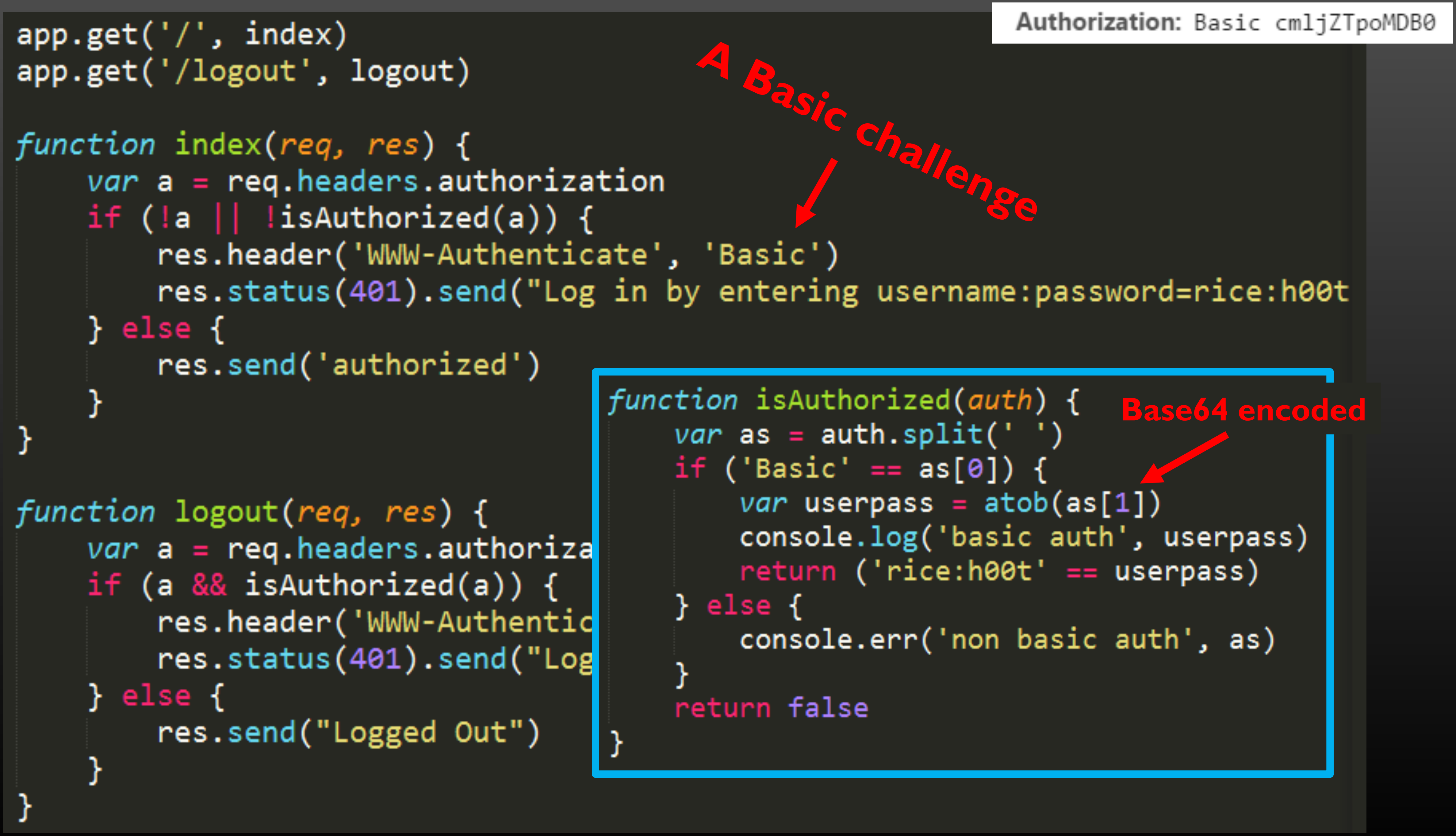
User Name:

Password:



```
app.get('/', index)
app.get('/logout', logout)

function index(req, res) {
    var a = req.headers.authorization
    if (!a || !isAuthorized(a)) {
        res.header('WWW-Authenticate', 'Basic')
        res.status(401).send("Log in by entering username:password=rice:h00t
    } else {
        res.send('authorized')
    }
}
```

A Basic challenge



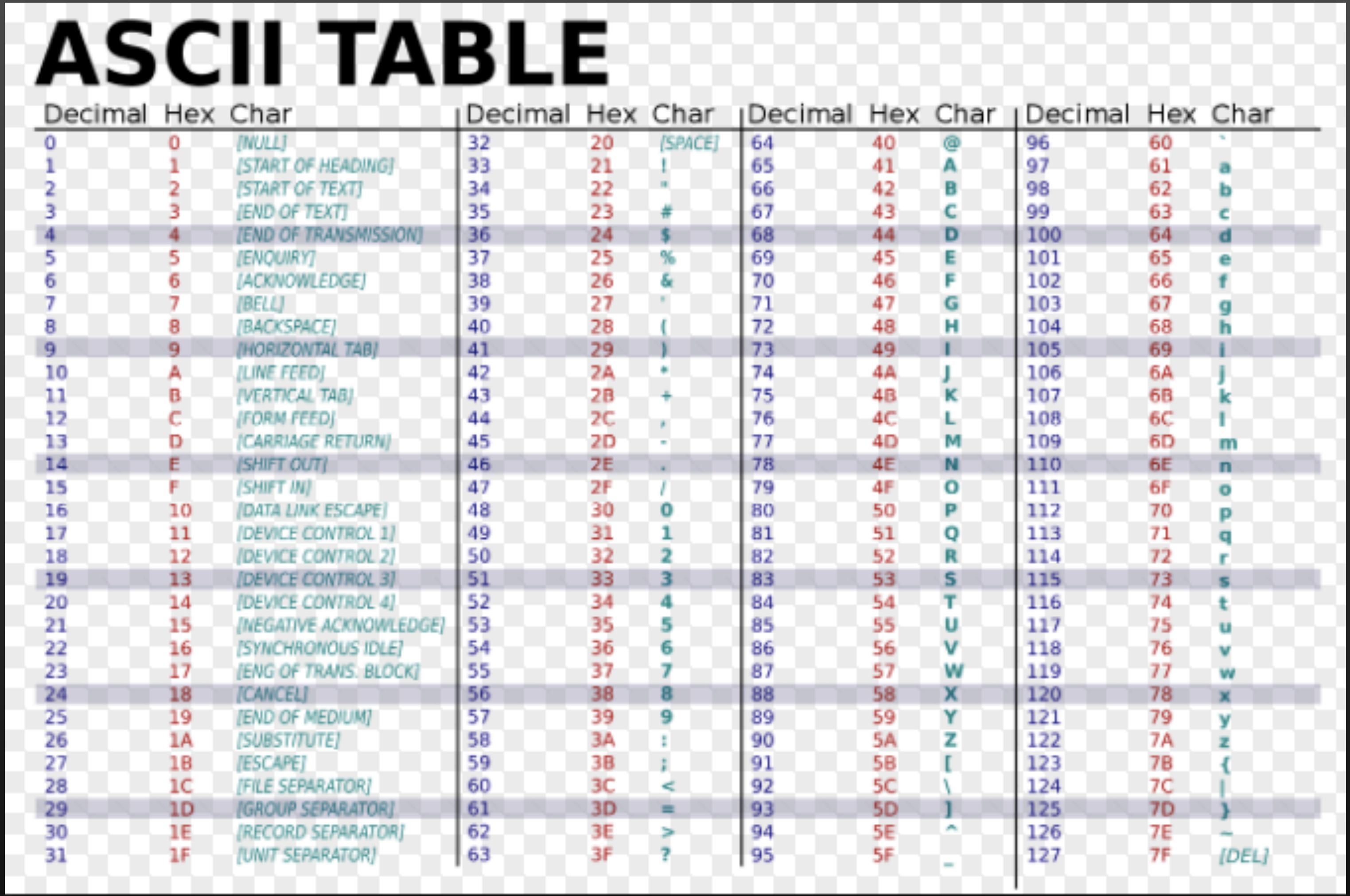
Authorization: Basic cmljZTpoMDB0

A Basic challenge

```
app.get('/', index)
app.get('/logout', logout)

function index(req, res) {
    var a = req.headers.authorization
    if (!a || !isAuthorized(a)) {
        res.header('WWW-Authenticate', 'Basic')
        res.status(401).send("Log in by entering username:password=rice:h00t
    } else {
        res.send('authorized')
    }
}

function logout(req, res) {
    var a = req.headers.authoriza
    if (a && isAuthorized(a)) {
        res.header('WWW-Authentic
        res.status(401).send("Log
    } else {
        res.send("Logged Out")
    }
}
```

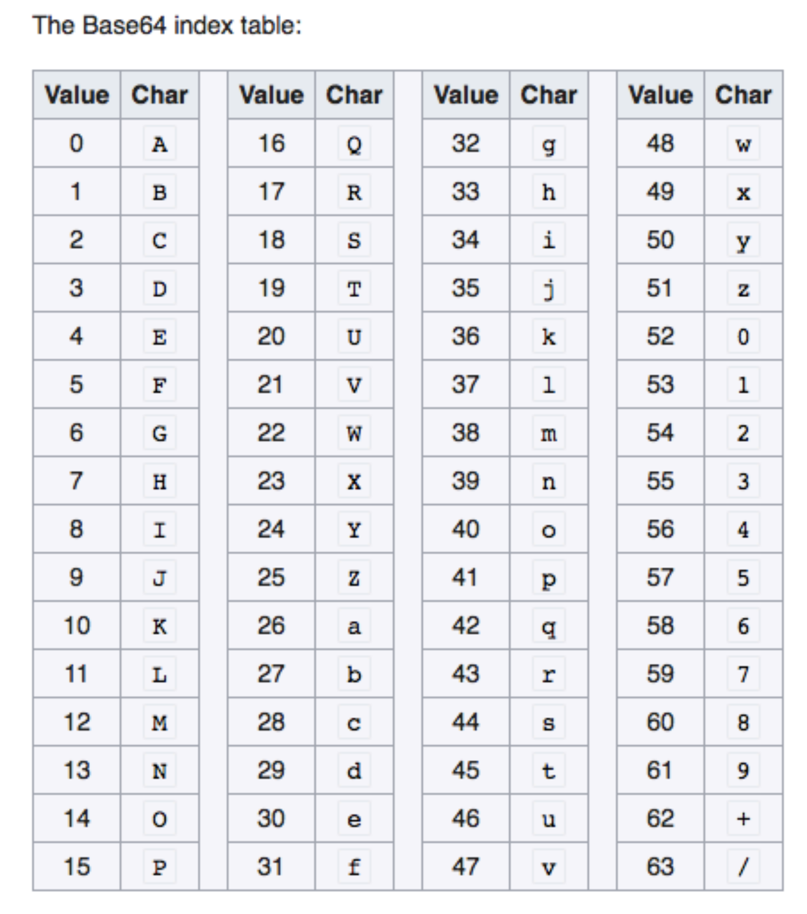
```
function isAuthorized(auth) {
    var as = auth.split(' ')
    if ('Basic' == as[0]) {
        var userpass = atob(as[1])
        console.log('basic auth', userpass)
        return ('rice:h00t' == userpass)
    } else {
        console.err('non basic auth', as)
    }
    return false
}
```

Base64 encoded



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



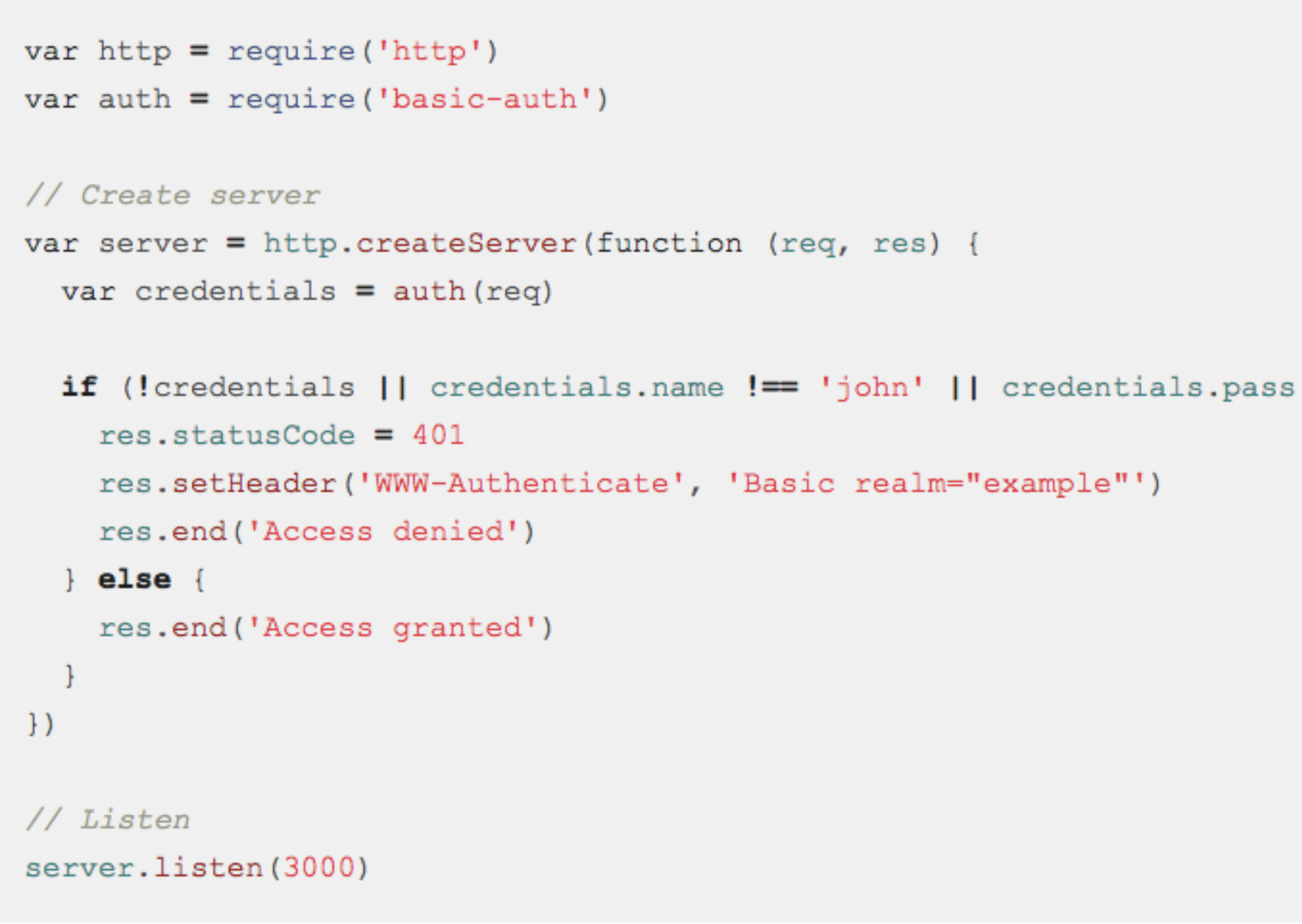
The Base64 index table:

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Example: Translation to Base64 encoding

r = 114 (ascii) in binary is 01110010, i = 105 (ascii) in binary is 01101001, c = 99 (ascii) in binary is 01100011,
Translate left to right (groups of 6 bits) 011100 = 28 = c, 100110 = 38 = m, 100101=37 = l, 100011 = 35 = j

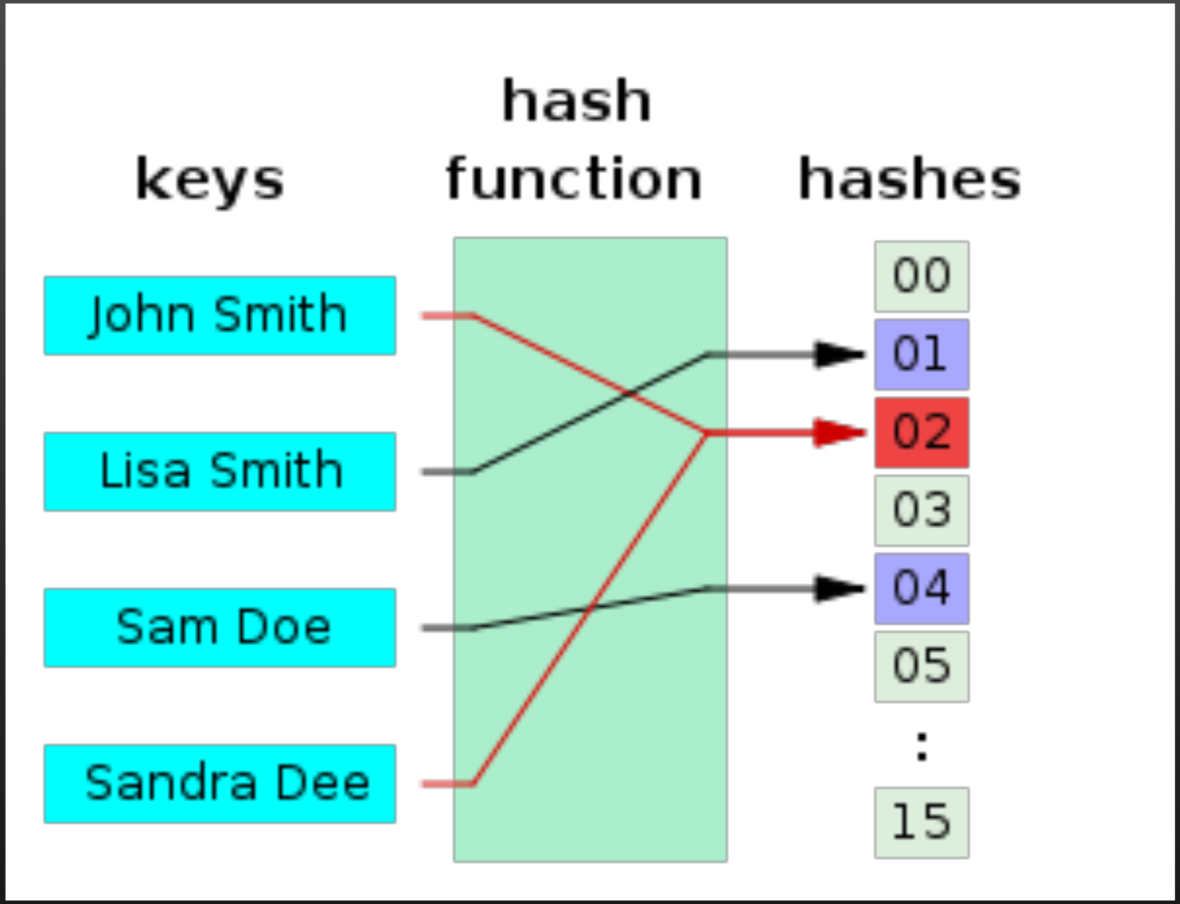
Basic Auth Node Module

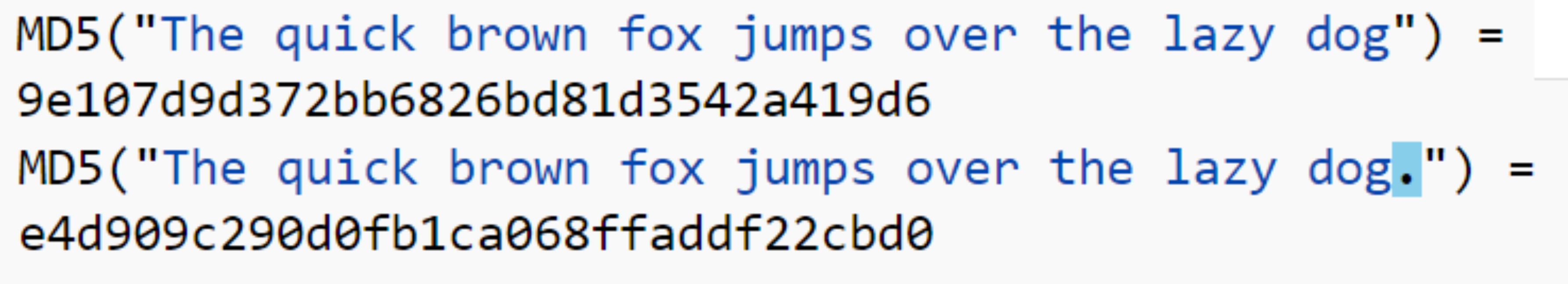


```
var http = require('http')
var auth = require('basic-auth')

// Create server
var server = http.createServer(function (req, res) {
  var credentials = auth(req)

  if (!credentials || credentials.name !== 'john' || credentials.pass
    res.statusCode = 401
    res.setHeader('WWW-Authenticate', 'Basic realm="example"')
    res.end('Access denied')
  } else {
    res.end('Access granted')
  }
})

// Listen
server.listen(3000)
```

npm install basic-auth --save

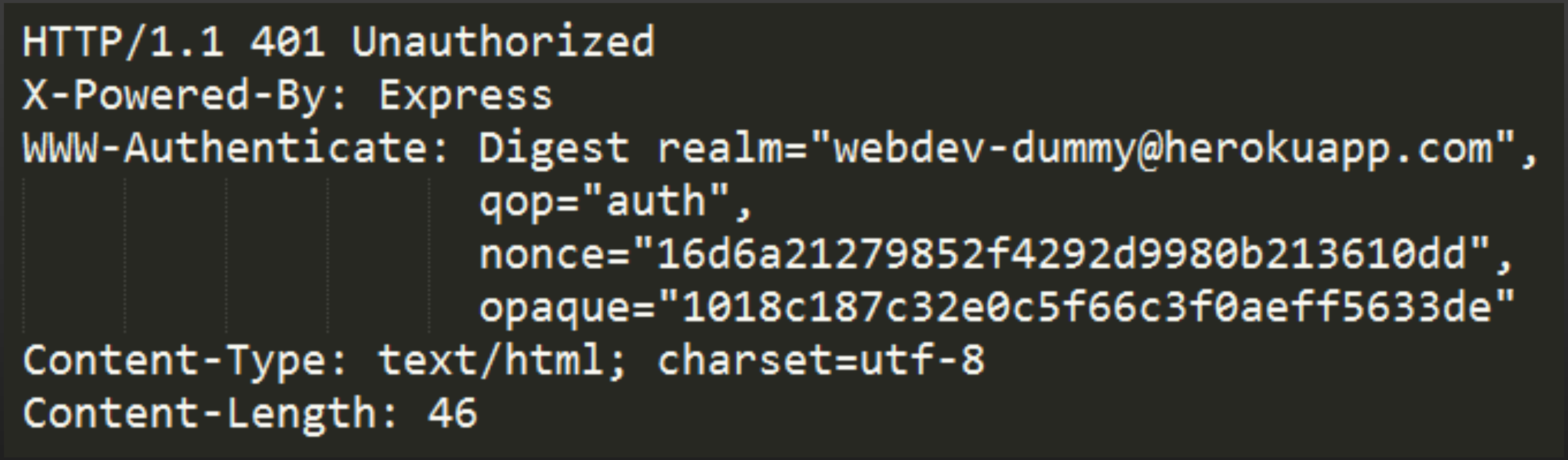
Hashing



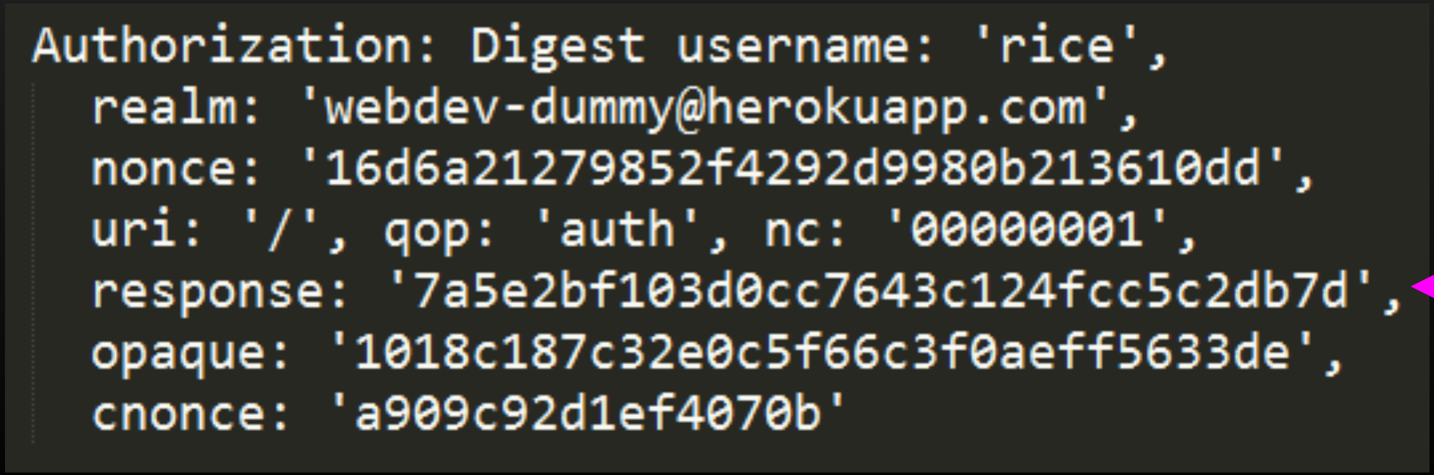


```
MD5("The quick brown fox jumps over the lazy dog") =
9e107d9d372bb6826bd81d3542a419d6
MD5("The quick brown fox jumps over the lazy dog.") =
e4d909c290d0fb1ca068ffaddf22cbd0
```

HTTP AUTH Digest Challenge

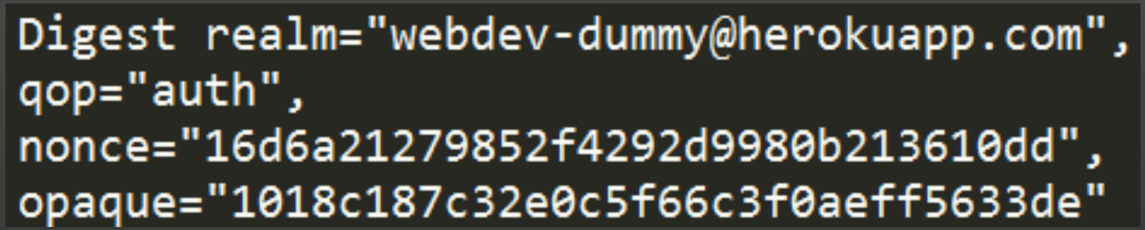


```
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
WWW-Authenticate: Digest realm="webdev-dummy@herokuapp.com",
                  qop="auth",
                  nonce="16d6a21279852f4292d9980b213610dd",
                  opaque="1018c187c32e0c5f66c3f0aeff5633de"
Content-Type: text/html; charset=utf-8
Content-Length: 46
```



```
Authorization: Digest username: 'rice',
  realm: 'webdev-dummy@herokuapp.com',
  nonce: '16d6a21279852f4292d9980b213610dd',
  uri: '/', qop: 'auth', nc: '00000001',
  response: '7a5e2bf103d0cc7643c124fcc5c2db7d',
  opaque: '1018c187c32e0c5f66c3f0aeff5633de',
  cnonce: 'a909c92d1ef4070b'
```

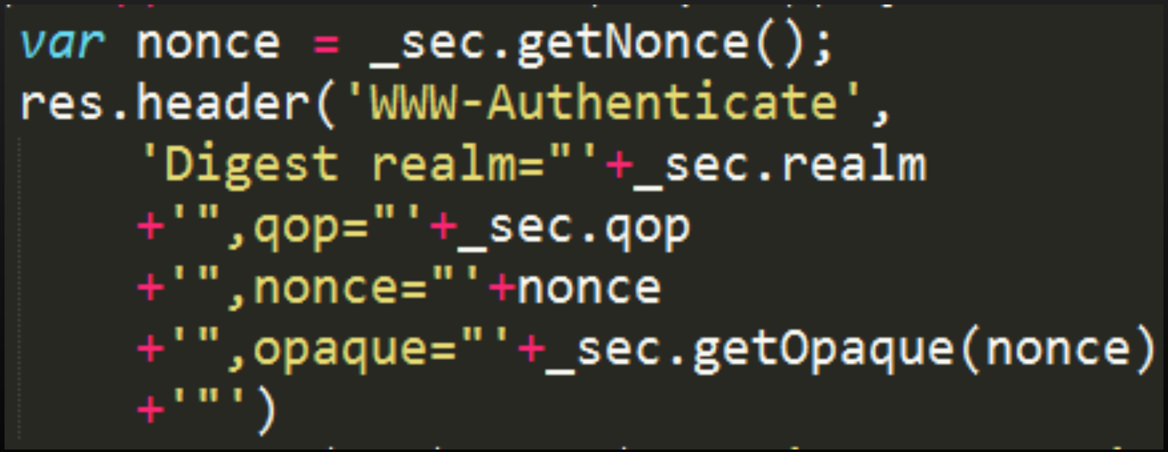
Password is in response



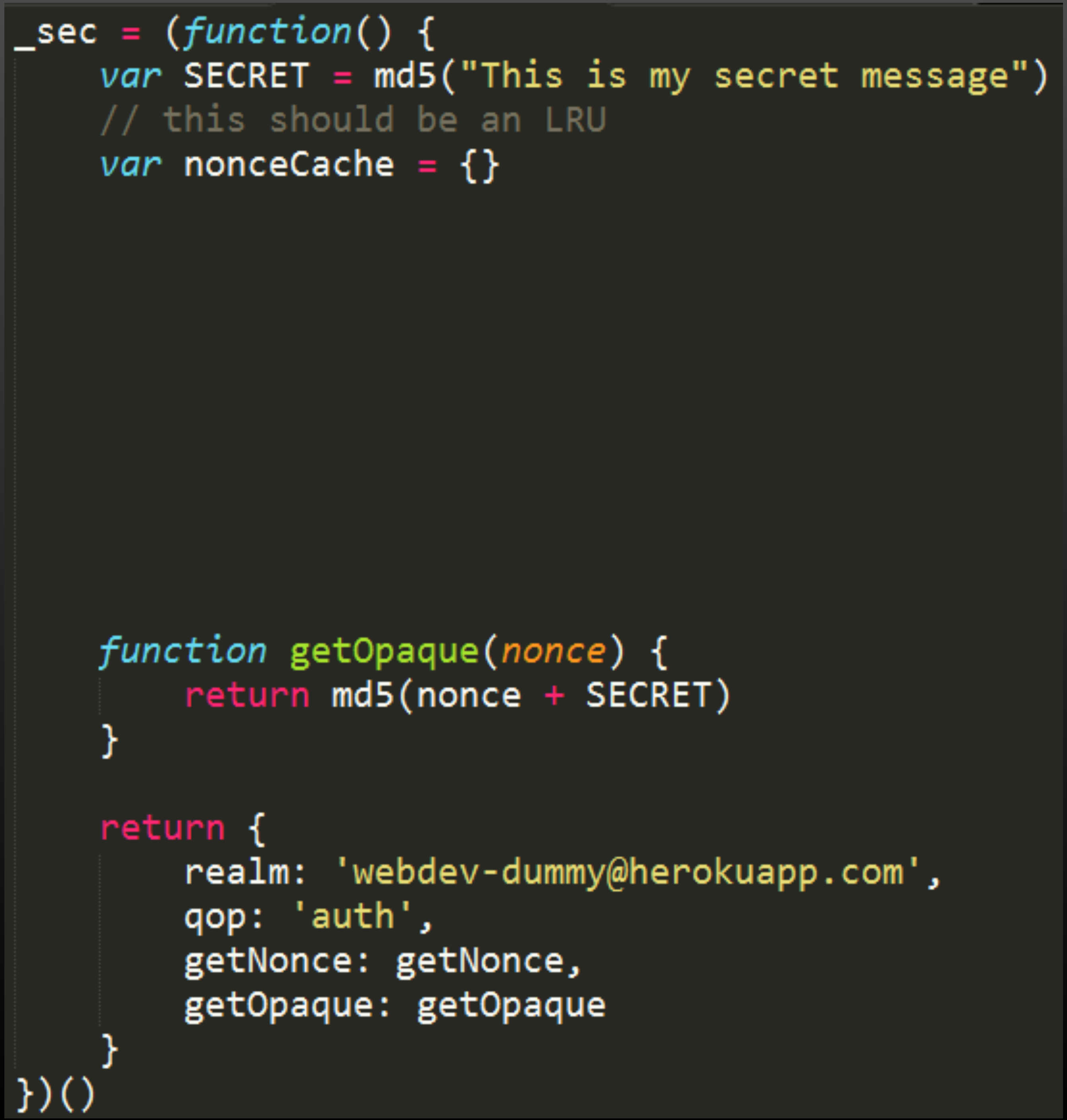
```
Digest realm="webdev-dummy@herokuapp.com",
qop="auth",
nonce="16d6a21279852f4292d9980b213610dd",
opaque="1018c187c32e0c5f66c3f0aeff5633de"
```

I “tied” together opaque and nonce.

This way you must know both the nonce and the opaque value to hack into the system.

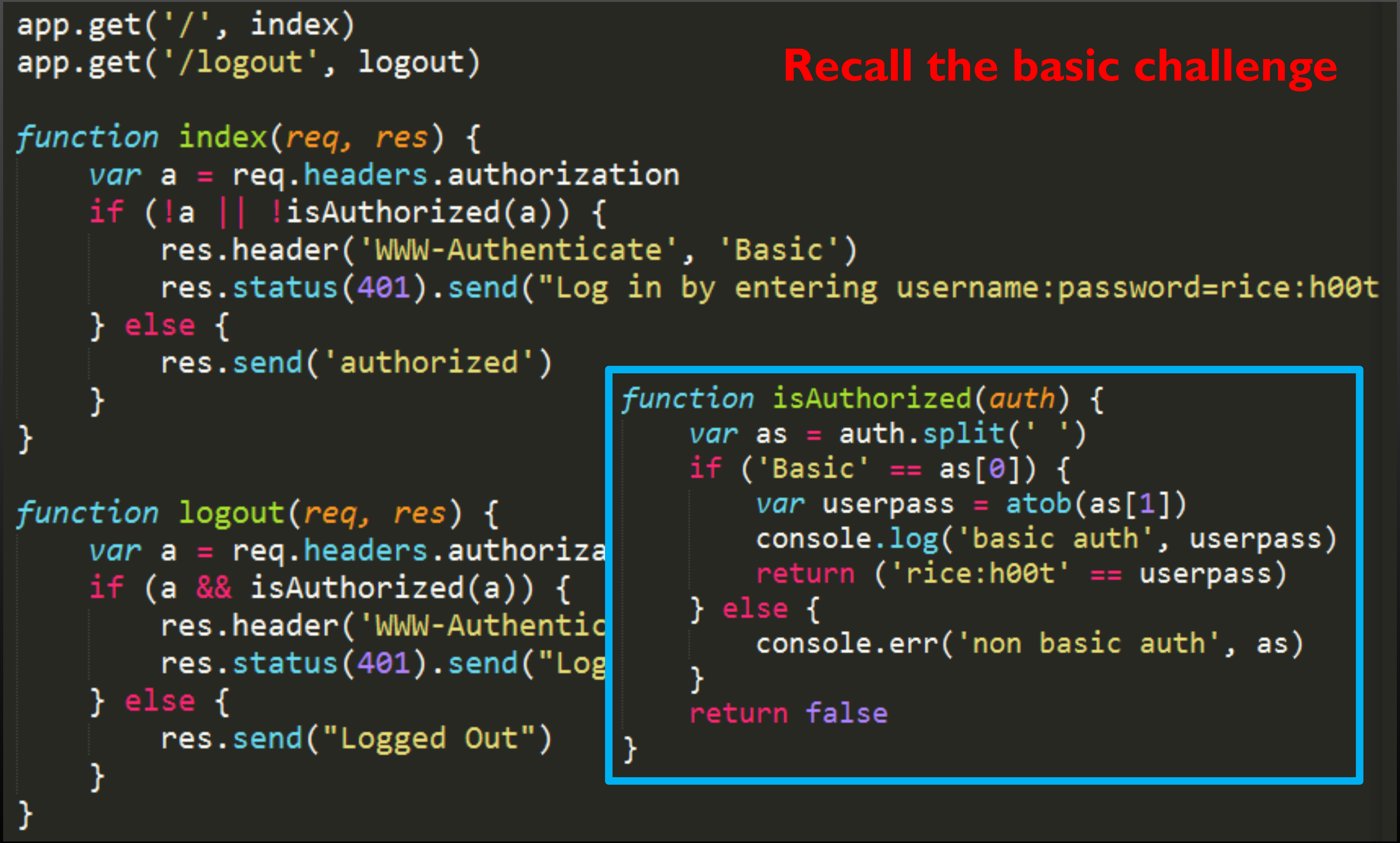


```
var nonce = _sec.getNonce();
res.header('WWW-Authenticate',
    'Digest realm="'+_sec.realm
    +'",qop="'+_sec.qop
    +'",nonce="'+nonce
    +'",opaque="'+_sec.getOpaque(nonce)
    +'"')
```



```
_sec = (function() {
    var SECRET = md5("This is my secret message")
    // this should be an LRU
    var nonceCache = {}








    function getOpaque(nonce) {
        return md5(nonce + SECRET)
    }

    return {
        realm: 'webdev-dummy@herokuapp.com',
        qop: 'auth',
        getNonce: getNonce,
        getOpaque: getOpaque
    }
})()
```

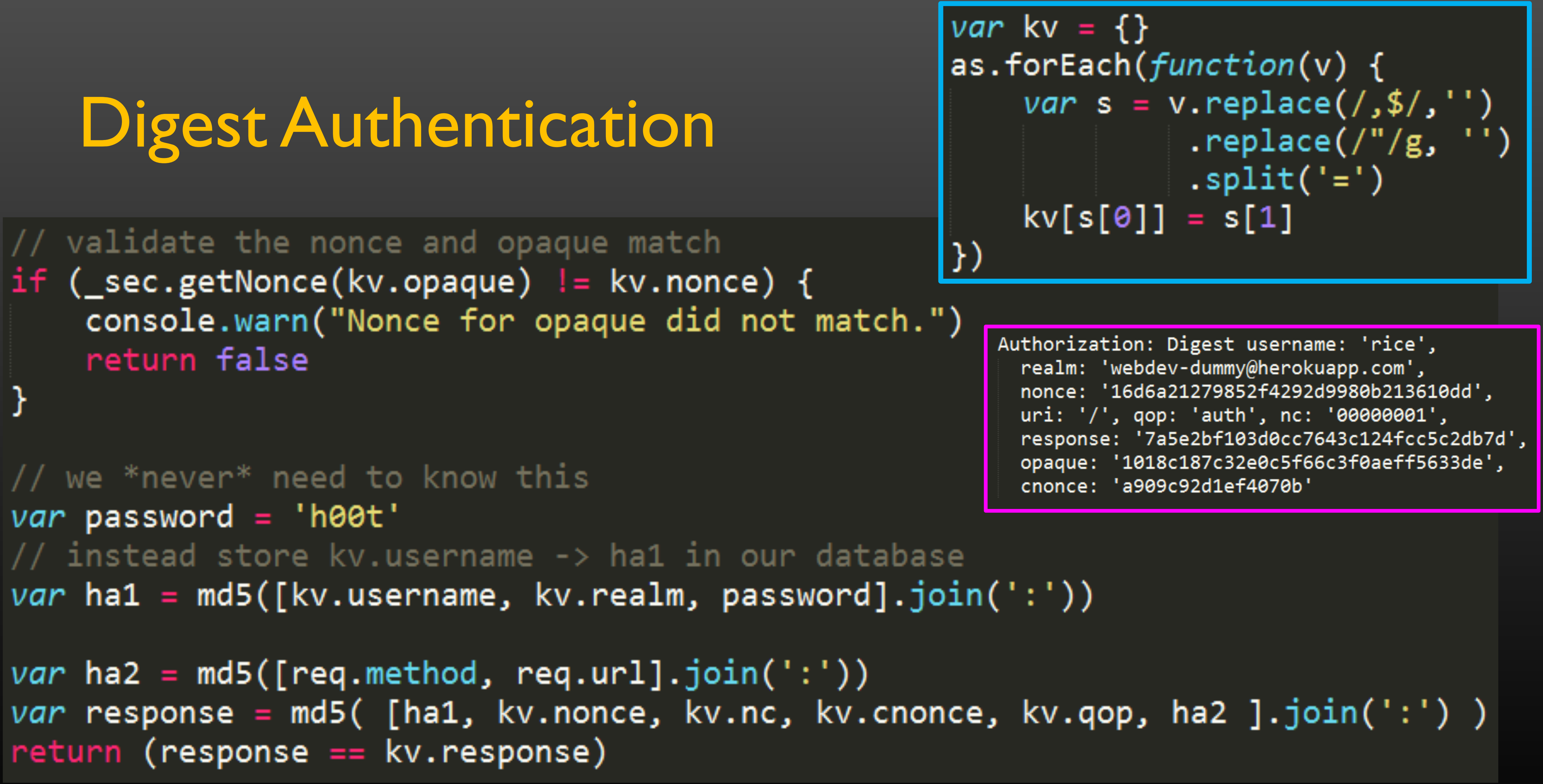
Recall the basic challenge



```
app.get('/', index)
app.get('/logout', logout)

function index(req, res) {
    var a = req.headers.authorization
    if (!a || !isAuthorized(a)) {
        res.header('WWW-Authenticate', 'Basic')
        res.status(401).send("Log in by entering username:password=rice:h00t
    } else {
        res.send('authorized')
    }
}

function logout(req, res) {
    var a = req.headers.authoriza
    if (a && isAuthorized(a)) {
        res.header('WWW-Authentic
        res.status(401).send("Log
    } else {
        res.send("Logged Out")
    }
}
```

```
function isAuthorized(auth) {
    var as = auth.split(' ')
    if ('Basic' == as[0]) {
        var userpass = atob(as[1])
        console.log('basic auth', userpass)
        return ('rice:h00t' == userpass)
    } else {
        console.err('non basic auth', as)
    }
    return false
}
```

Digest Authentication

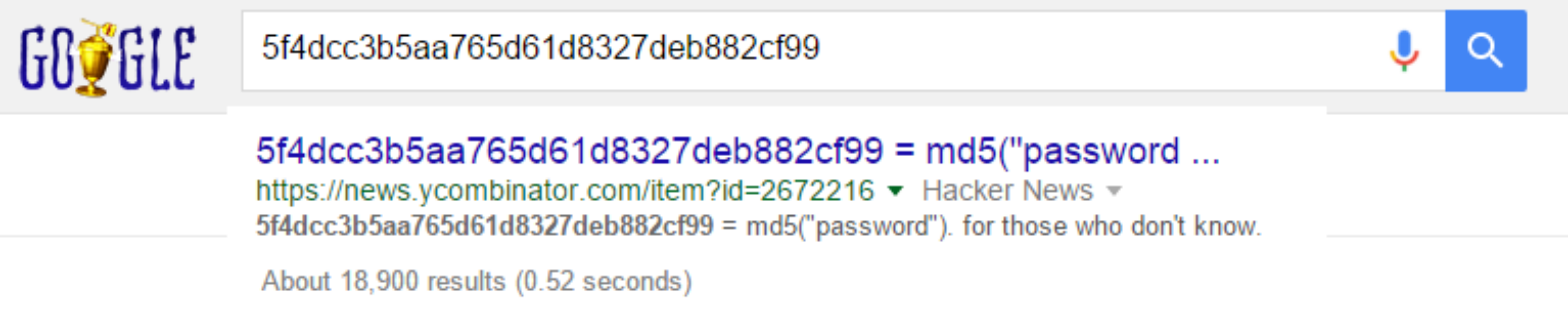


```
var kv = {}
as.forEach(function(v) {
    var s = v.replace(/,$/,'')
             .replace(/"/g, '')
             .split('=')
    kv[s[0]] = s[1]
})
```

```
// validate the nonce and opaque match
if (_sec.getNonce(kv.opaque) != kv.nonce) {
    console.warn("Nonce for opaque did not match.")
    return false
}

// we *never* need to know this
var password = 'h00t'
// instead store kv.username -> ha1 in our database
var ha1 = md5([kv.username, kv.realm, password].join(':'))

var ha2 = md5([req.method, req.url].join(':'))
var response = md5( [ha1, kv.nonce, kv.nc, kv.cnonce, kv.qop, ha2 ].join(':') )
return (response == kv.response)
```

```
Authorization: Digest username: 'rice',
  realm: 'webdev-dummy@herokuapp.com',
  nonce: '16d6a21279852f4292d9980b213610dd',
  uri: '/', qop: 'auth', nc: '00000001',
  response: '7a5e2bf103d0cc7643c124fcc5c2db7d',
  opaque: '1018c187c32e0c5f66c3f0aeff5633de',
  cnonce: 'a909c92d1ef4070b'
```

Hash lookup



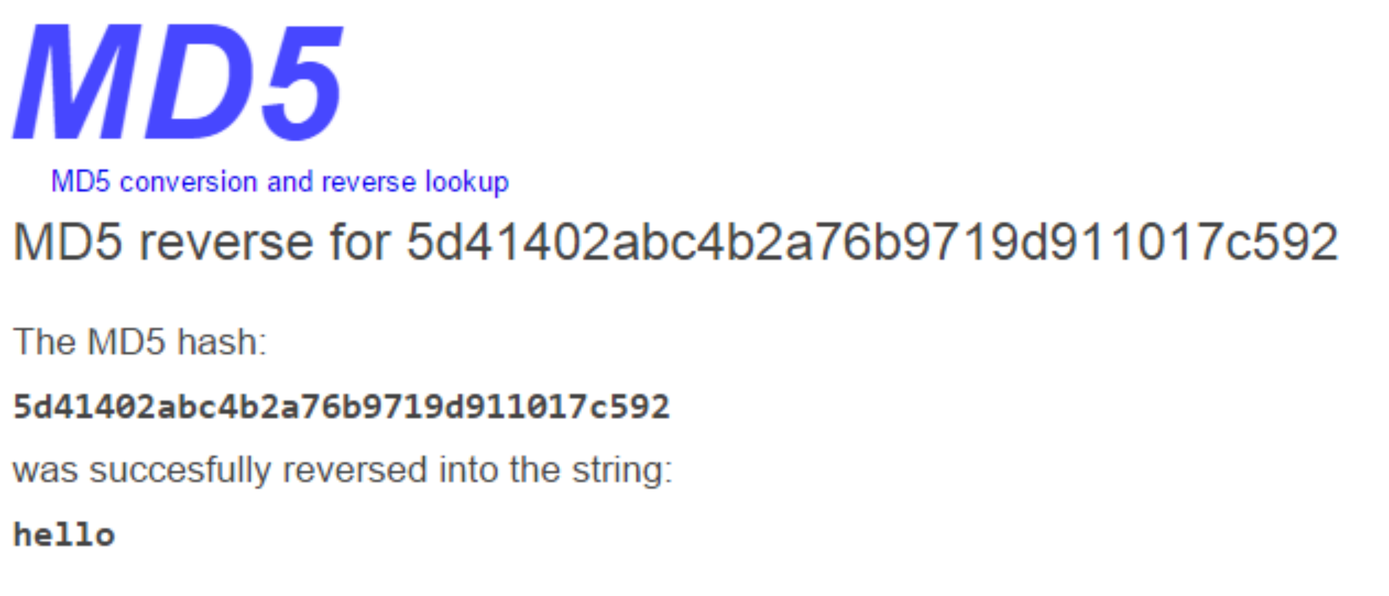
5f4dcc3b5aa765d61d8327deb882cf99

5f4dcc3b5aa765d61d8327deb882cf99 = md5("password ...

https://news.ycombinator.com/item?id=2672216 ▾ Hacker News ▾

5f4dcc3b5aa765d61d8327deb882cf99 = md5("password"). for those who don't know.

About 18,900 results (0.52 seconds)



MD5

MD5 conversion and reverse lookup

MD5 reverse for 5d41402abc4b2a76b9719d911017c592

The MD5 hash:

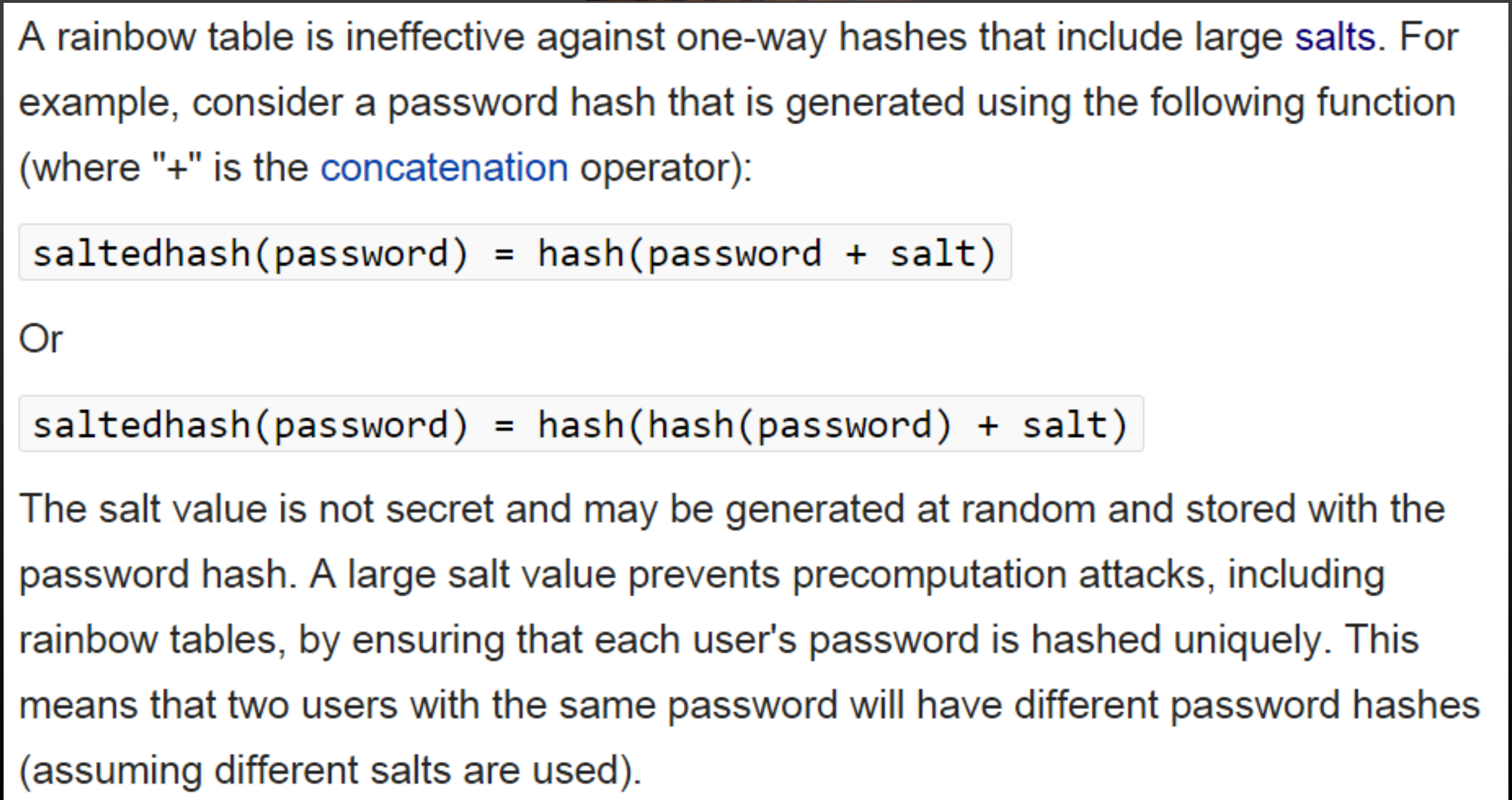
5d41402abc4b2a76b9719d911017c592

was succesfully reversed into the string:

hello

https://en.wikipedia.org/wiki/Rainbow_table

Defense: Salting



A rainbow table is ineffective against one-way hashes that include large salts. For example, consider a password hash that is generated using the following function (where "+" is the concatenation operator):

```
saltedhash(password) = hash(password + salt)
```

Or

```
saltedhash(password) = hash(hash(password) + salt)
```

The salt value is not secret and may be generated at random and stored with the password hash. A large salt value prevents precomputation attacks, including rainbow tables, by ensuring that each user's password is hashed uniquely. This means that two users with the same password will have different password hashes (assuming different salts are used).

Salted Passwords

- Pre-Salt Plan of Attack:
  - Create a look up table of every *n*-character password to hash (slow)

    OR
  - Use a rainbow table of every *n*-character password to hash (faster)

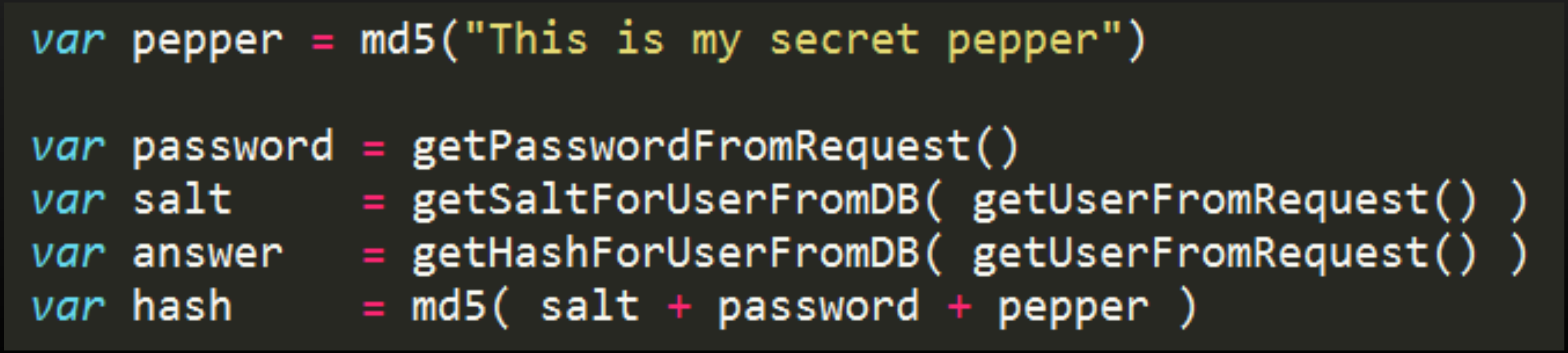
- The salt is typically public
  - Now they have to have a larger *n*-character lookup table
- Salted Plan of attack:
  - Take the salt, generate a table from it
  - We're in!

It just takes time…

Peppering

…security through obscurity

- Note that we have a different salt for each user
- This salt is in the database
- If the database is compromised an attacker can get it by making a lookup table
- Pepper is a secret code on the server, not in the database



```
var pepper = md5("This is my secret pepper")

var password = getPasswordFromRequest()
var salt     = getSaltForUserFromDB( getUserFromRequest() )
var answer   = getHashForUserFromDB( getUserFromRequest() )
var hash     = md5( salt + password + pepper )
```

Security, security, security

You don’t want to be hacked

- Hash on the browser? Sure.
- Hash on the server? Definitely

- MD5 and SHA-1 are now “trivial” do not use them in production
- H(H(H(H(H(H(H(….. H(password + salt) + salt) + salt) …..)

- Instead use a Key-Derivation Function (KDF) such as PBKDF2 or bcrypt / scrypt