OWASP Top Ten Security Vulnerabilities in Node.js

Marco Ippolito

NearForm

Marco Ippolito

Senior Developer Experience Engineer
@NearForm

Node.js collaborator

Open Source enthusiast

@satanacchio

OWASP is a non profit foundation dedicated to improving software security

OWASP
Open Web Application Security Project

OWASP provides every year rankings for the top 10 most critical web applications security risks.

Criteria

- Frequency
- Severity
- Impact

Criteria

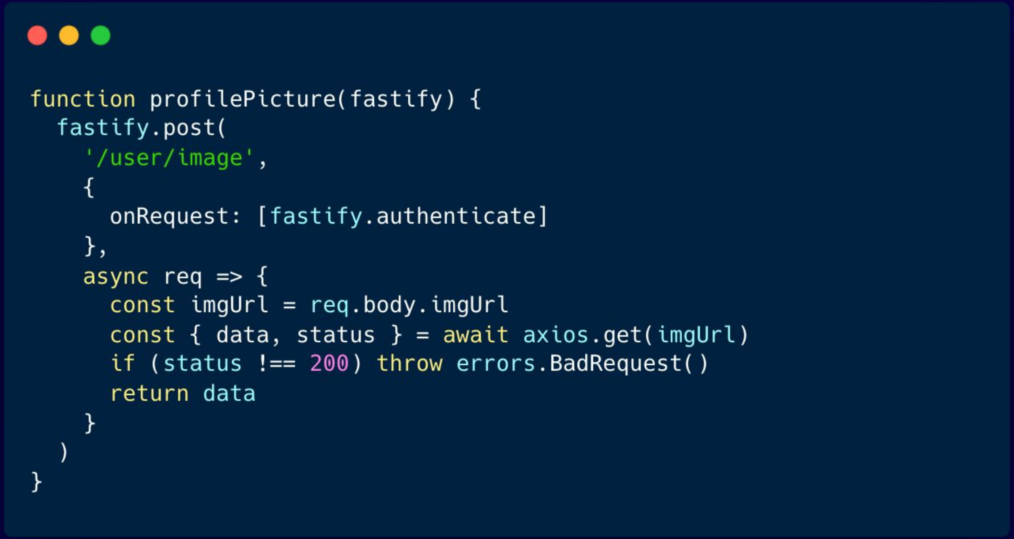
- Frequency
- **Severity**
- Impact

Criteria

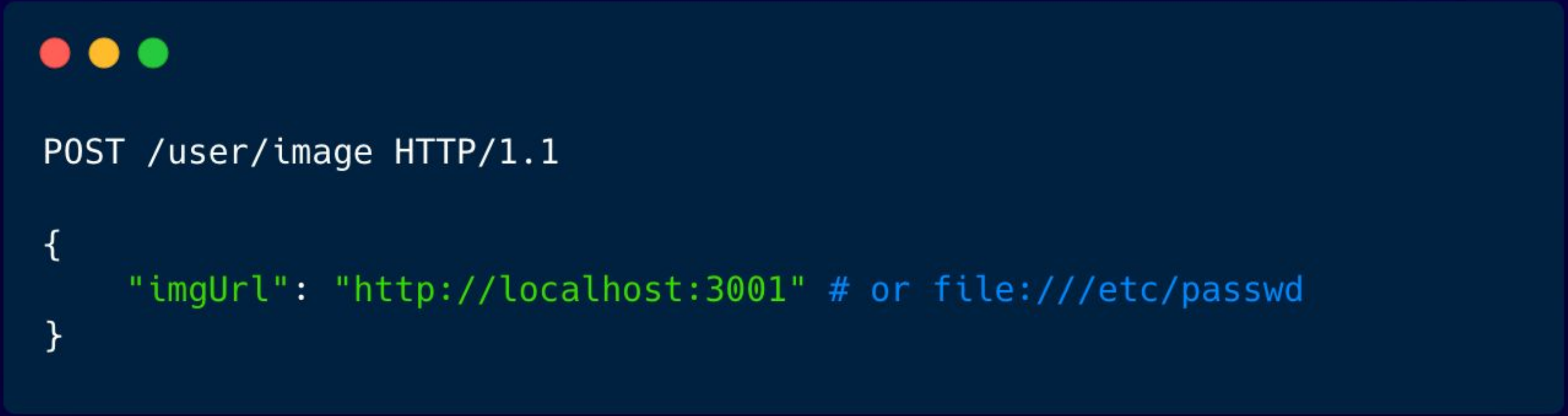
- Frequency
- Severity
- Impact

10 – Server Side Request Forgery

A web application is fetching remote resources without validating user supplied URL



```javascript
function profilePicture(fastify) {
  fastify.post(
    '/user/image',
    {
      onRequest: [fastify.authenticate]
    },
    async req => {
      const imgUrl = req.body.imgUrl
      const { data, status } = await axios.get(imgUrl)
      if (status !== 200) throw errors.BadRequest()
      return data
    }
  )
}
```



```
POST /user/image HTTP/1.1

{
    "imgUrl": "http://localhost:3001" # or file:///etc/passwd
}
```

Remediation 🪄

- Sanitize URLs
- Do not send raw responses to client
- Disable HTTP redirections

Remediation 🪄

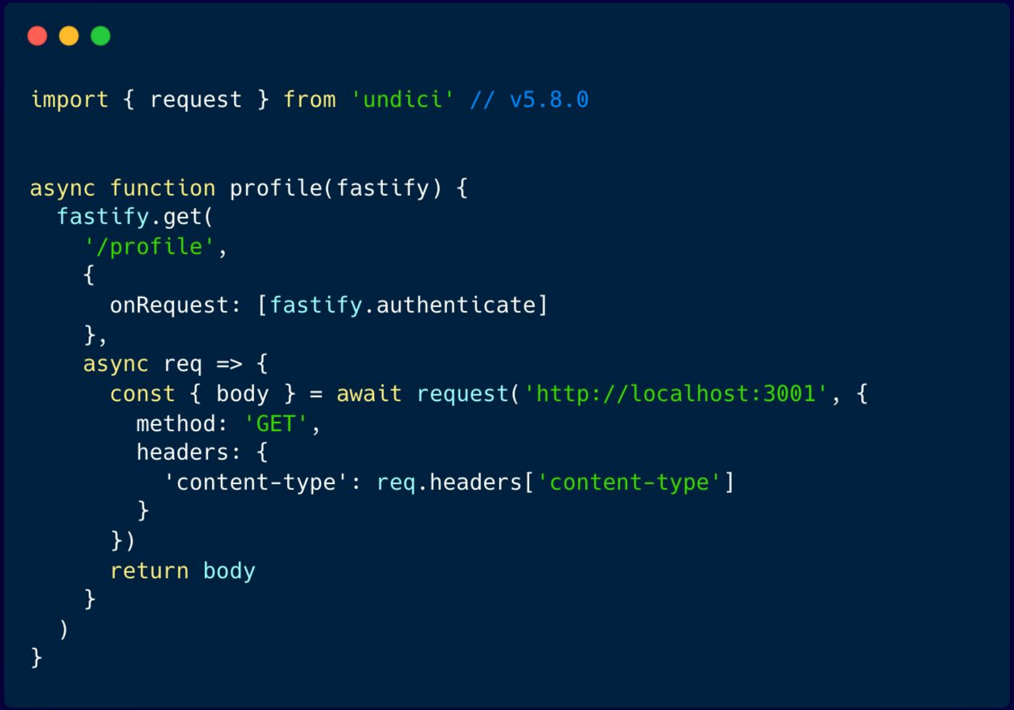
- Sanitize URLs
- **Do not send raw responses to client**
- Disable HTTP redirections

Remediation 🪄

- Sanitize URLs
- Do not send raw responses to client
- Disable HTTP redirections

9 - Security Logging and Monitoring Failures

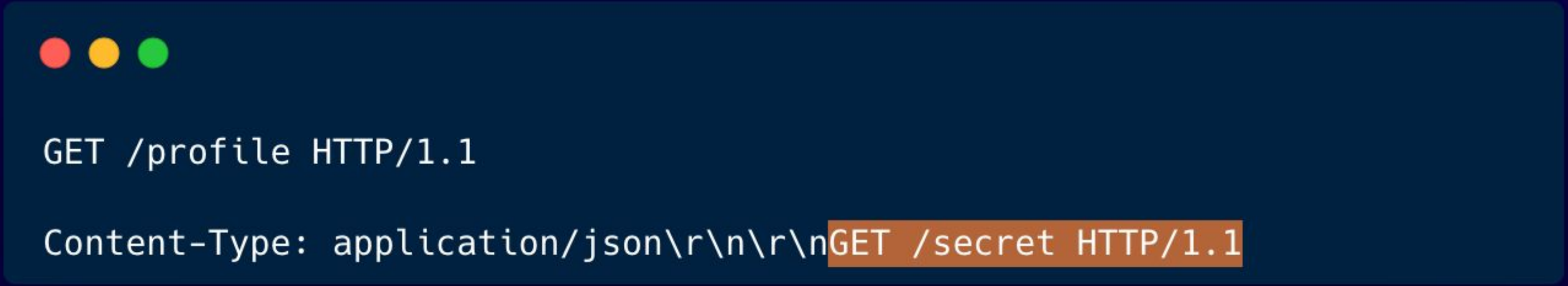
Insufficient logging, monitoring and detection of events such as logins, failed logins, high value transactions, suspicious activities



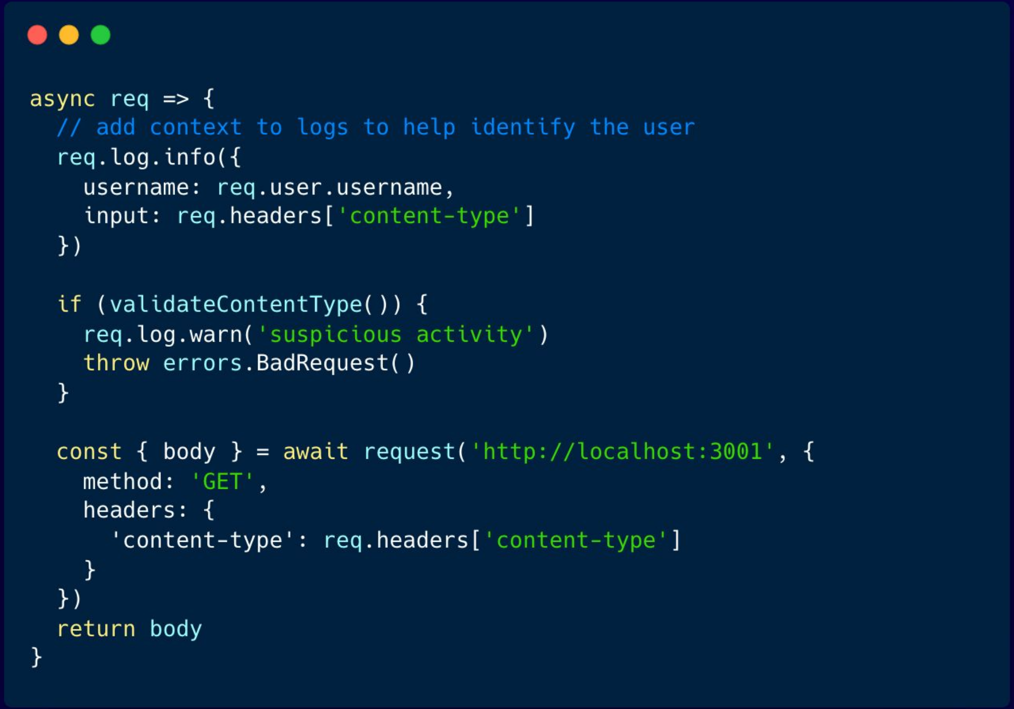
```
import { request } from 'undici' // v5.8.0


async function profile(fastify) {
  fastify.get(
    '/profile',
    {
      onRequest: [fastify.authenticate]
    },
    async req => {
      const { body } = await request('http://localhost:3001', {
        method: 'GET',
        headers: {
          'content-type': req.headers['content-type']
        }
      })
      return body
    }
  )
}
```

@satanacchio

CVE-2022-35948



```
GET /profile HTTP/1.1

Content-Type: application/json\r\n\r\nGET /secret HTTP/1.1
```



```
async req => {
  // add context to logs to help identify the user
  req.log.info({
    username: req.user.username,
    input: req.headers['content-type']
  })

  if (validateContentType()) {
    req.log.warn('suspicious activity')
    throw errors.BadRequest()
  }

  const { body } = await request('http://localhost:3001', {
    method: 'GET',
    headers: {
      'content-type': req.headers['content-type']
    }
  })
  return body
}
```

@satanacchio

Remediation 🪄

- **Ensure all login, access control, input validation failures are logged**
- **Ensure logs are generated in a format that log manager can easily consume**
- **Establish effective monitoring and alerting**

Remediation 🪄

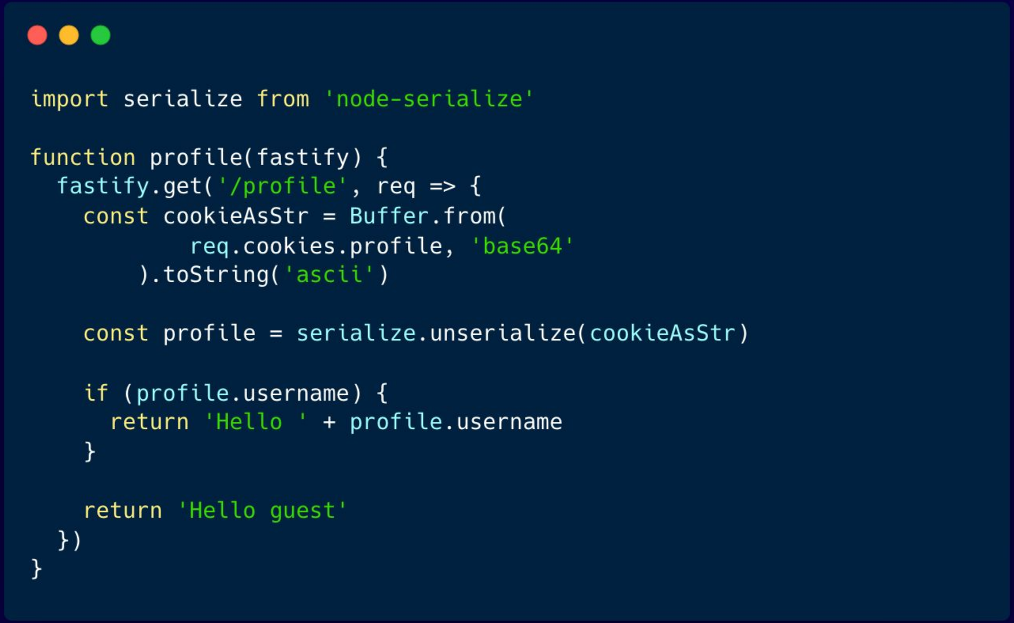
- Ensure all login, access control, input validation failures are logged
- **Ensure logs are generated in a format that log manager can easily consume**
- Establish effective monitoring and alerting

Remediation 🪄

- Ensure all login, access control, input validation failures are logged
- Ensure logs are generated in a format that log manager can easily consume
- **Establish effective monitoring and alerting**

8 - Software Data and Integrity Failures

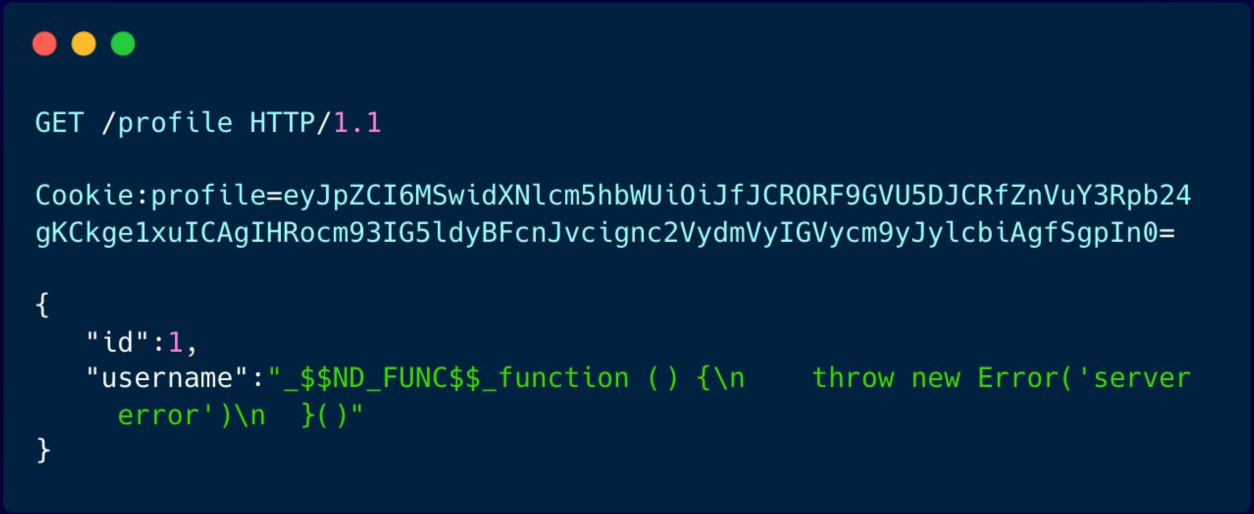
Software and data integrity failures occur when an attacker can modify or delete data in an unauthorized manner



```
import serialize from 'node-serialize'

function profile(fastify) {
  fastify.get('/profile', req => {
    const cookieAsStr = Buffer.from(
            req.cookies.profile, 'base64'
        ).toString('ascii')

    const profile = serialize.unserialize(cookieAsStr)

    if (profile.username) {
      return 'Hello ' + profile.username
    }

    return 'Hello guest'
  })
}
```

@satanacchio

CVE-2017-5941



```
GET /profile HTTP/1.1

Cookie:profile=eyJpZCI6MSwidXNlcm5hbWUiOiJfJCRORF9GVU5DJCRfZnVuY3Rpb24
gKCkge1xuICAgIHRocm93IG5ldyBFcnJvcignc2VydmVyIGVycm9yJylcbiAgfSgpIn0=

{
   "id":1,
   "username":"_$$ND_FUNC$$_function () {\n    throw new Error('server
     error')\n  }()"
}
```

Remediation 🪄

- **Ensure unsigned or serialized data is not tampered**
- **Ensure libraries and dependencies come from trusted repositories**
- **Check digital signatures to verify that software has not been altered**

Remediation 🪄

- Ensure unsigned or serialized data is not tampered
- **Ensure libraries and dependencies come from trusted repositories**
- Check digital signatures to verify that software has not been altered

Remediation 🪄

- Ensure unsigned or serialized data is not tampered
- Ensure libraries and dependencies come from trusted repositories
- **Check digital signatures to verify that software has not been altered**

7 - Identification and Authentication failures

System inability to identify the user or validate the identity of their user as their own

There is 1 Impostor among us
imgflip.com

Remediation 🪄

- Use multi factor authentication
- Limit failed logins
- Implement weak password checks
- Align password length, complexity and rotation with NIST

Remediation 🪄

- Use multi factor authentication
- **Limit failed logins**
- Implement weak password checks
- Align password length, complexity and rotation with NIST

Remediation 🪄

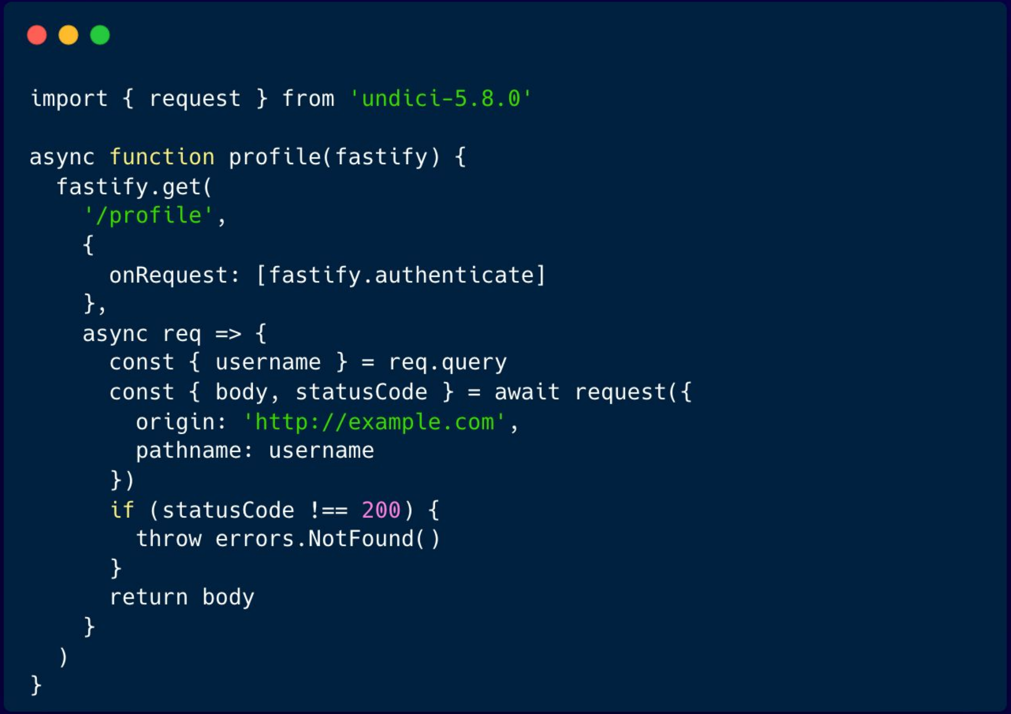
- Use multi factor authentication
- Limit failed logins
- **Implement weak password checks**
- Align password length, complexity and rotation with NIST

Remediation 🪄

- Use multi factor authentication
- Limit failed logins
- Implement weak password checks
- **Align password length, complexity and rotation policies with NIST**

6 - Vulnerable and outdated components

Third-party libraries or frameworks that have known vulnerabilities or are no longer supported by maintainers



```
import { request } from 'undici-5.8.0'

async function profile(fastify) {
  fastify.get(
    '/profile',
    {
      onRequest: [fastify.authenticate]
    },
    async req => {
      const { username } = req.query
      const { body, statusCode } = await request({
        origin: 'http://example.com',
        pathname: username
      })
      if (statusCode !== 200) {
        throw errors.NotFound()
      }
      return body
    }
  )
}
```

Remediation 🪄

- Use tools to monitor the status of your dependencies
- Automate dependency update workflow
- Remove unused dependencies

Remediation 🪄

- Use tools to monitor the status of your dependencies
- **Automate dependency update workflow**
- Remove unused dependencies

Remediation 🪄

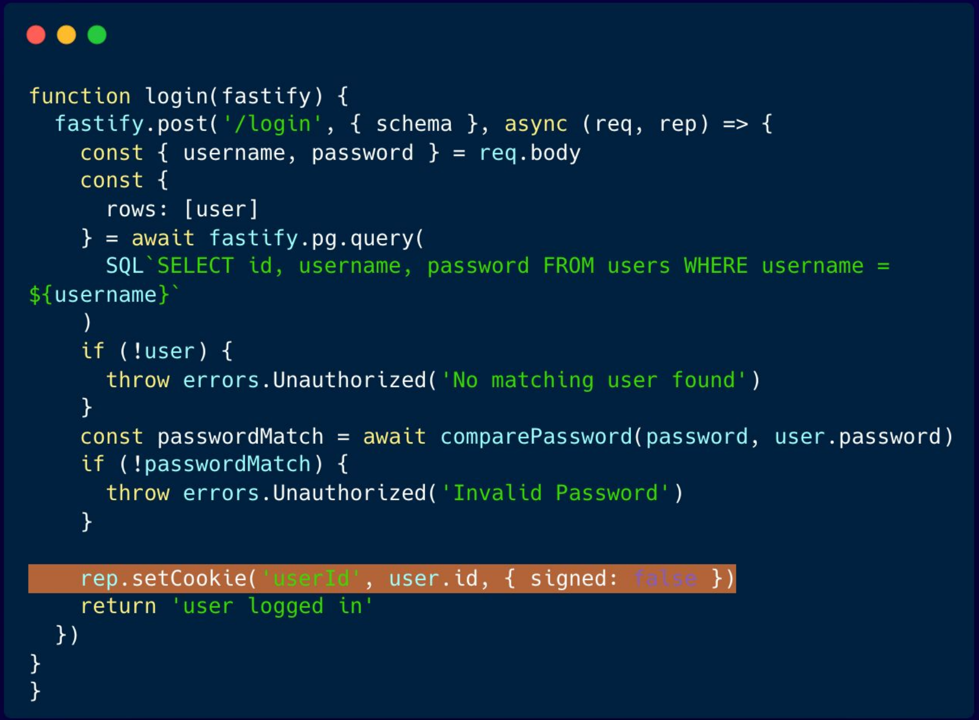
- Use tools to monitor the status of your dependencies
- Automate dependency update workflow
- **Remove unused dependencies**

- `npx is-my-node-vulnerable`
- Snyk vulnerability scanner
- retire.js

5 - Security Misconfiguration

Security settings are not adequately defined in the configuration process or maintained and deployed with default settings

@satanacchio



```
function login(fastify) {
  fastify.post('/login', { schema }, async (req, rep) => {
    const { username, password } = req.body
    const {
      rows: [user]
    } = await fastify.pg.query(
      SQL`SELECT id, username, password FROM users WHERE username =
${username}`
    )
    if (!user) {
      throw errors.Unauthorized('No matching user found')
    }
    const passwordMatch = await comparePassword(password, user.password)
    if (!passwordMatch) {
      throw errors.Unauthorized('Invalid Password')
    }

    rep.setCookie('userId', user.id, { signed: false })
    return 'user logged in'
  })
}
}
```

Sign your cookies and use httpOnly flag

Remediation 🪄

- Different credentials should be used in each environment
- Repeatable, automated and fast to deploy environments
- Tests to verify effectiveness of configuration

Remediation 🪄

- Different credentials should be used in each environment
- **Repeatable, automated and fast to deploy environments**
- Tests to verify effectiveness of configuration

Remediation 🪄

- Different credentials should be used in each environment
- Repeatable, automated and fast to deploy environments
- Tests to verify effectiveness of configuration

4 - Insecure Design

Lack of security controls being integrated in the application during the development cycle

MY WEBAPP
HOPE
DREAMS
PRAYERS
imgflip.com

Remediation 🪄

- Model threats for the application, flows and business logic
- Use unit and integration tests to verify threat model
- Re-evaluate security requirements and design during development lifecycle

Remediation 🪄

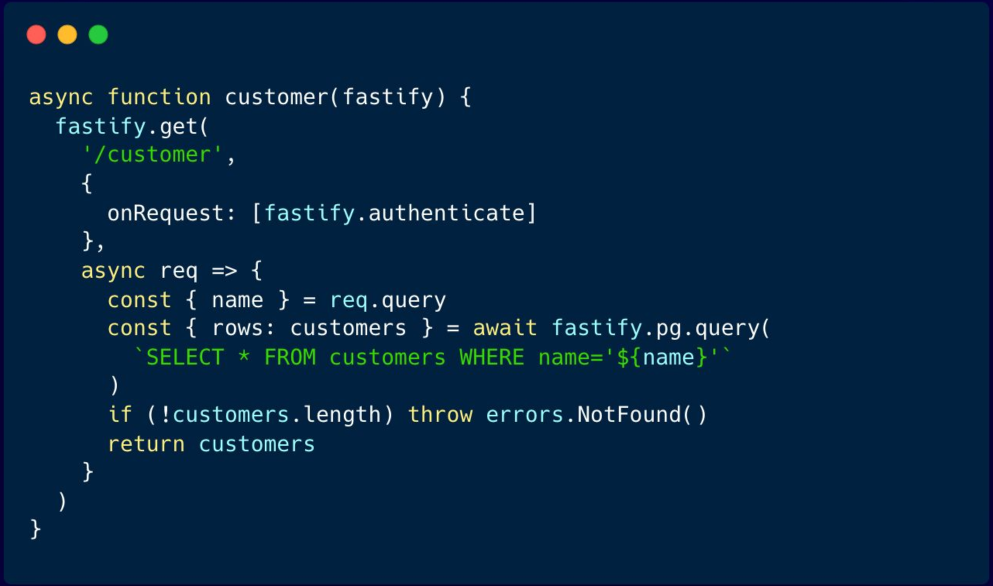
- Model threats for the application, flows and business logic
- **Use unit and integration tests to verify threat model**
- Re-evaluate security requirements and design during development lifecycle

Remediation 🪄

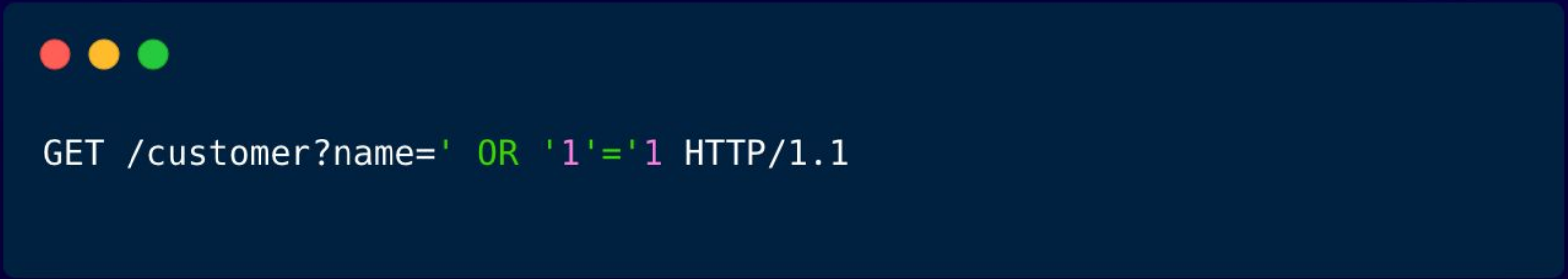
- Model threats for the application, flows and business logic
- Use unit and integration tests to verify threat model
- **Re-evaluate security requirements and design during development lifecycle**

3 - Injection

Malicious payload is able to inject an arbitrary bit of query or code on the target server


```
async function customer(fastify) {
  fastify.get(
    '/customer',
    {
      onRequest: [fastify.authenticate]
    },
    async req => {
      const { name } = req.query
      const { rows: customers } = await fastify.pg.query(
        `SELECT * FROM customers WHERE name='${name}'`
      )
      if (!customers.length) throw errors.NotFound()
      return customers
    }
  )
}
```

@satanacchio



```
GET /customer?name=' OR '1'='1 HTTP/1.1
```

Remediation 🪄

- Validate user input
- Escape special characters
- Avoid user supplied tables names or columns

Remediation 🪄

- Validate user input
- Escape special characters
- Avoid user supplied tables names or columns

Remediation 🪄

- Validate user input
- Escape special characters
- **Avoid user supplied tables names or columns**

2 - Cryptographic Failures

Exposing sensitive data on a weak or non existent cryptographic algorithm

MD5
imgflip.com

Remediation 🪄

- **Use up to date and strong encryption algorithms**
- **Proper key secrets management**
- **Disable caching for data that contains sensitive information**

Remediation 🪄

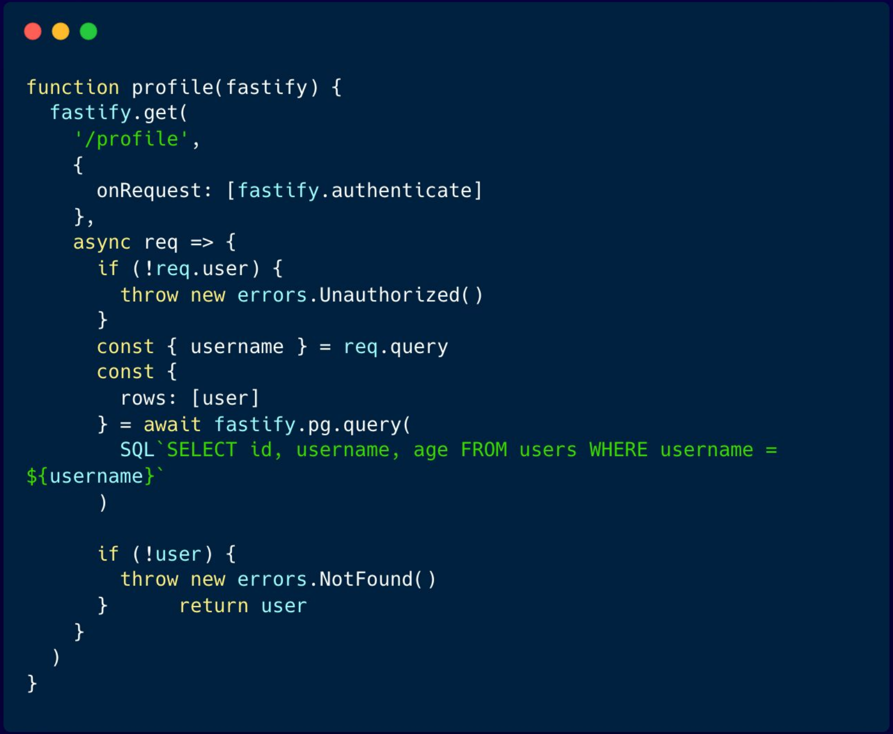
- Use up to date and strong encryption algorithms
- **Proper key secrets management**
- Disable caching for data that contains sensitive information

Remediation 🪄

- Use up to date and strong encryption algorithms
- Proper key secrets management
- **Disable caching for data that contains sensitive information**

1 - Broken Access Control

Users can access resources or perform actions that they are not supposed to be able to access



```
function profile(fastify) {
  fastify.get(
    '/profile',
    {
      onRequest: [fastify.authenticate]
    },
    async req => {
      if (!req.user) {
        throw new errors.Unauthorized()
      }
      const { username } = req.query
      const {
        rows: [user]
      } = await fastify.pg.query(
        SQL`SELECT id, username, age FROM users WHERE username =
${username}`
      )

      if (!user) {
        throw new errors.NotFound()
      }      return user
    }
  )
}
```

Remediation 🪄

- Except for public resources, deny by default
- Implement access control once and reuse them throughout the application
- Tokens should be short-lived
- Log access control failures

Remediation 🪄

- Except for public resources, deny by default
- **Implement access control once and reuse them throughout the application**
- Tokens should be short-lived
- Log access control failures

Remediation 🪄

- Except for public resources, deny by default
- Implement access control once and reuse them throughout the application
- Tokens should be short-lived
- Log access control failures

Remediation 🪄

- Except for public resources, deny by default
- Implement access control once and reuse them throughout the application
- Tokens should be short-lived
- **Log access control failures**

Thanks for listening!!!

United States:	(916) 299-6882
International:	+353 51 330 290
United Kingdom:	0870 067 95569
Ireland:	051 330 290

nearform.com

sales@nearform.com

NearForm