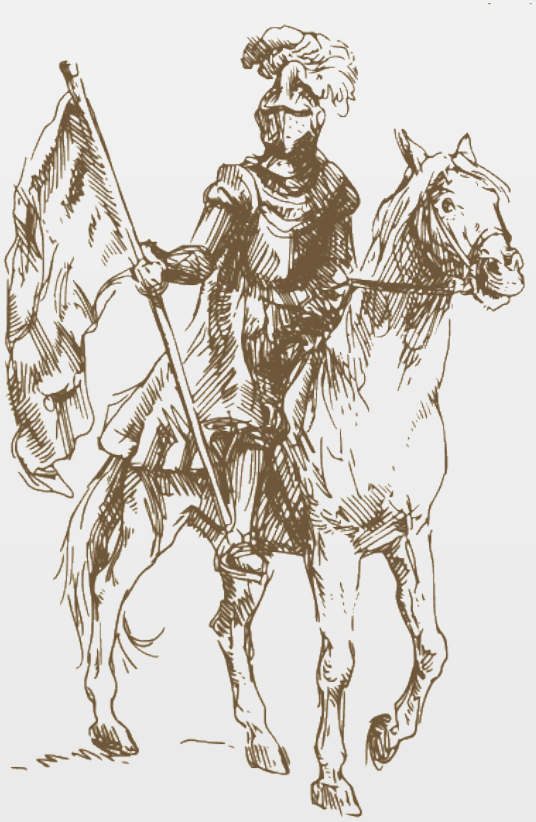


Battling

Top Overlooked Security Threats to Node.js Web Applications

Chetan Karande, Omgeo, OWASP

1. Fortify Our Defenses

   Addressing Overlooked Environment Configuration Issues

2. Engage in Warfare

   Mitigating Overlooked Security Attacks

Know thy self, know thy enemy.
A thousand battles, a thousand victories.

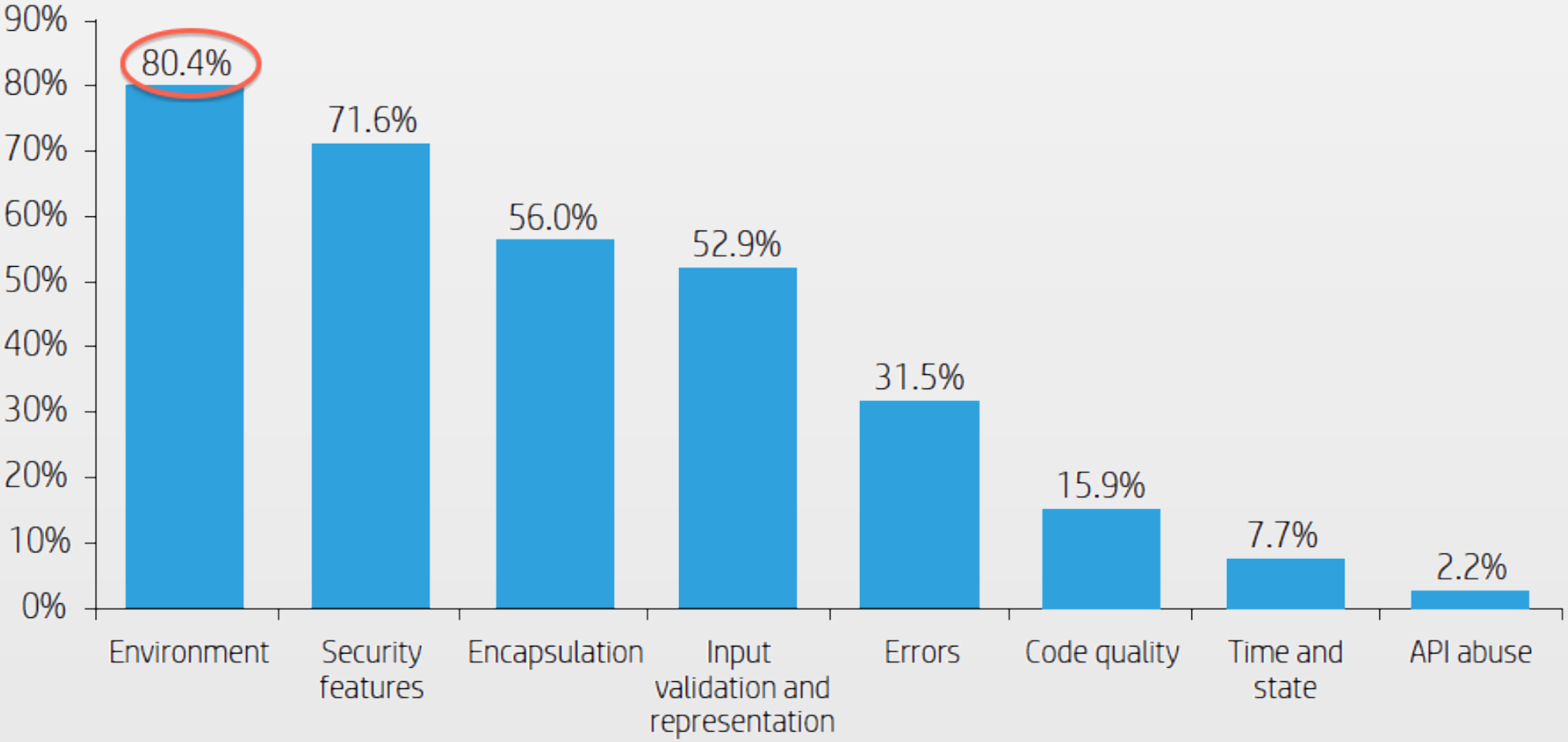
- Sun Tzu, The Art of War

Quiz

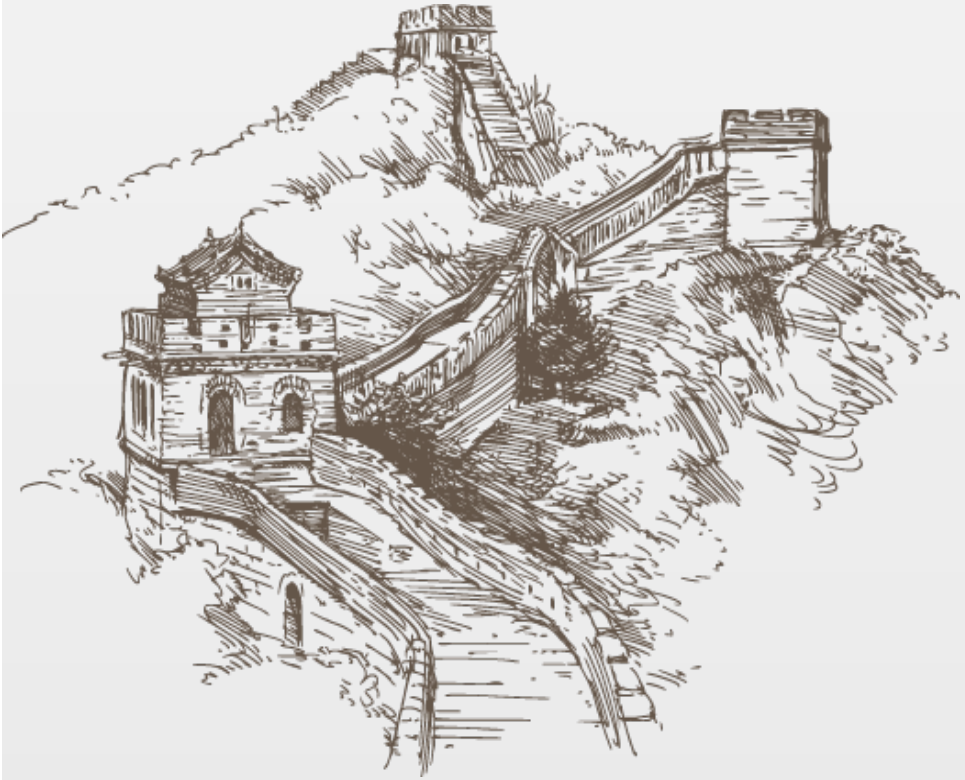
Identify the weakest area in a web application, where an attacker is most likely to find vulnerabilities?

A. Data Encryption
B. Environment Configuration
C. Input Validation
D. Error Handling

Year 2013 Vulnerabilities Sampling by Category

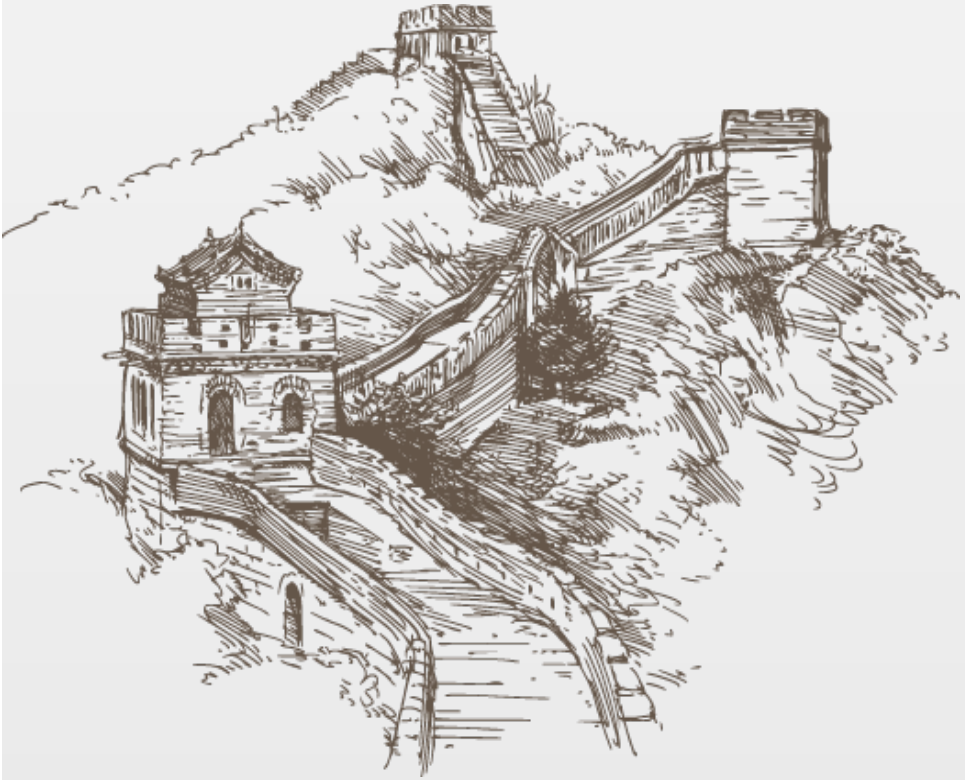


Source: HP 2013 cyber risk report



1. Fortify Our Defenses

Addressing Overlooked Environment Configuration Issues



Preventing Internal Implementation Disclosure

- The X-Powered-By header can be extremely useful to an attacker for building site's risk profile.

HTTP Response Headers

- X-Powered-By header has no functional value. It can be removed safely.

server.js

```
var express = require("express");
var app = express();
…
app.disable("x-powered-by");
```

Other ways to remove X-Powered-By –

server.js

```
…
app.use(helmet.hidePoweredBy());
```

Other ways to remove X-Powered-By –

server.js

```
…
app.use(helmet.hidePoweredBy({
      setTo: "PHP 4.2.0"
}));
```

- Another source of implementation disclosure - default session cookie name

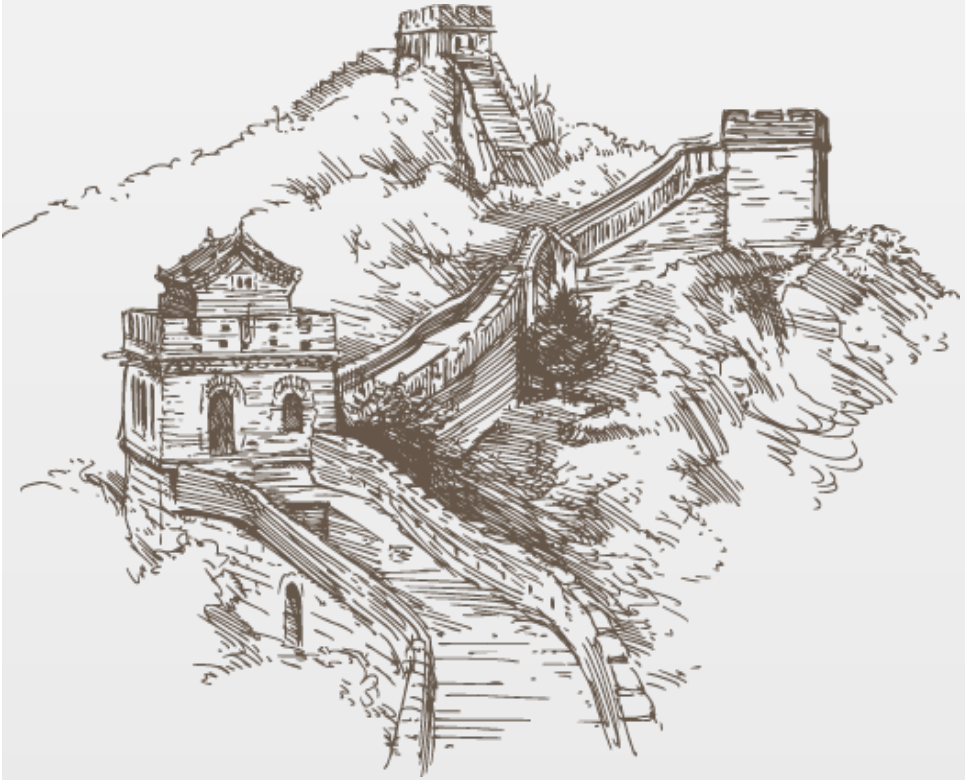
HTTP Response Headers

Use generic cookie names

server.js

```
var session = require("express-session");

app.use(session({
    secret: "s3Cur3",
    key: "sessionId",
    …
}));
```



Configuring Protection against CSRF

Enable CSRF Protection

server.js

```
var csrf= require("csurf");

app.use(csrf());
```

Enable CSRF Protection

server.js

```
var csrf= require("csurf");

app.use(csrf());
…
app.use(function(req, res, next) {
    res.locals.csrftoken = req.csrfToken();
    next();
});
```

Enable CSRF Protection

server.js

```
var csrf= require("csurf");

app.use(csrf());
…
app.use(function(req, res, next) {
    res.locals.csrftoken = req.csrfToken();
    next();
});
```

Form Template

```
…

<input type="hidden" name="_csrf" value="{{csrftoken}}">
```

- Express CSRF middleware ignores verifying tokens on HTTP GET, OPTIONS, and HEAD requests (which is a correct behavior)

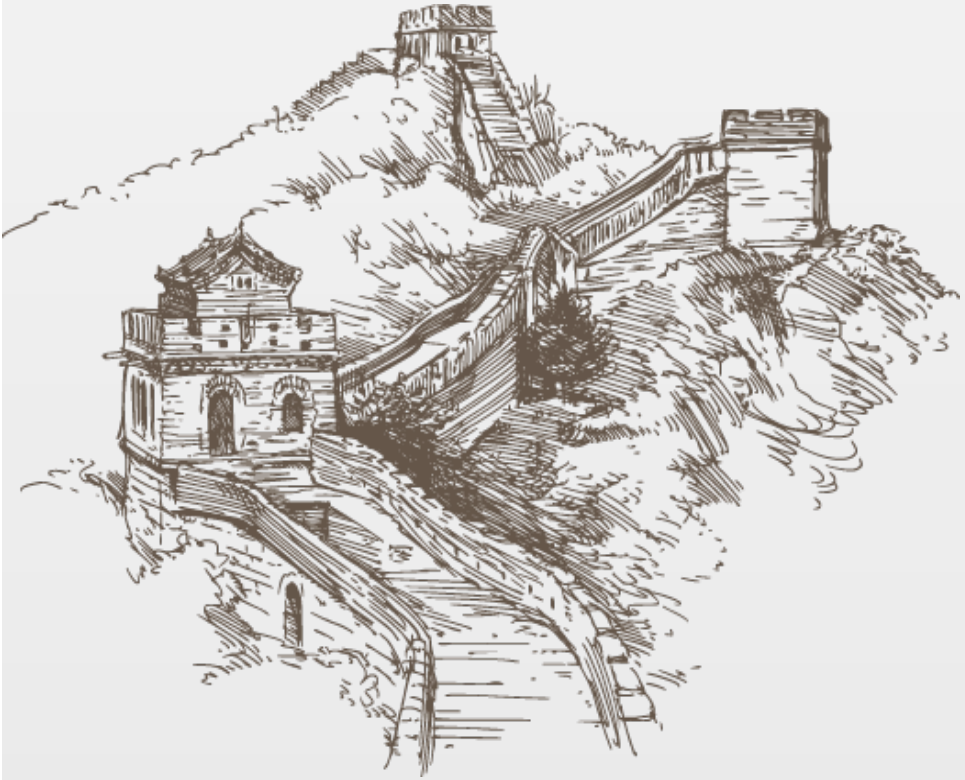
- Ensure GET APIs are coded not to mutate states.

Use method-override module before CSRF

server.js

```
var methodOverride = require("method-override");
var csrf= require("csurf");

app.use(methodOverride("X-HTTP-Method-Override"));
app.use(csrf());
```



Using Secure Version of Software Dependencies

- Use the latest stable version of Node.js and frameworks.

  Node.js security vulnerabilities

  Express security updates

- Stay up to date on npm module versions and known vulnerbailities

- Useful tools:

  npm outdated

  Node Security Project

  Retire.js

2. Engaging in Warfare

Mitigating Overlooked Security Attacks

Cross Site Scripting (XSS) Attack

An attacker can exploit XSS vulnerability to -

- Steal session cookies, and then impersonate the user.
- Redirect user to malicious sites.

- Myth: Template libraries handle output encoding by default, making application safe against XSS attacks

- Myth: Template libraries handle output encoding by default, making application safe against XSS attacks

- Encode untrusted data for correct context depending on where it will be placed

Encode for HTML Body

```
<div> Untrusted Data </div>
```

&	→	&amp;	"	→	&quot;
<	→	&lt;	'	→	&#x27;
>	→	&gt;	/	→	&#x2F;

Encode for HTML Attributes

```
<input type="text" name="firstname" value="Untrusted Data">
```

Non-alphanumeric characters → &#xHH; format

Enclose attribute value in quotes

Encode for CSS

```
<div style="width= Untrusted Data ;">contents</div>
```

Untrusted data → CSS Hex Encoding (\HH or \HHHHHH)

Encode for JavaScript

```
<script> var firstName=" Untrusted Data "; </script>
```

Non-alphanumeric characters → \uXXXX; unicode format

Encode for URL

```
<a href="Untrusted Data">Show Details</a>
```

Untrusted data → encodeURI()

Encode for URL Parameter

```
<a href="/account?id= Untrusted Data ">Show Details</a>
```

Untrusted data → encodeURIComponent()

DOM Based XSS: Encode on both server and client

```
<a href="/reviews#Untrusted Data">Movie Reviews</a>
```

```
<script>
 document.write("<h1>"+ document.location.hash +"</h1>");
</script>
```

- Use proven utilities for encoding (e.g. OWASP ESAPI)

Add HTTPOnly, Secure attributes on Session Cookie

server.js

```
var session = require("express-session");

app.use(session({
    secret: "s3Cur3",
    key: "sessionId",
    cookie: {
            httpOnly: true,
            secure: true
    }
}));
```

Add Content Security Policy header

server.js

```
var policy = {
  defaultPolicy: {
    "default-src": ["'self'"],
    "img-src": ["static.example.com"]
  }
}

helmet.csp.policy(policy);
```

Regular Expression Denial of Service (ReDoS) Attack

- Evil regex can take exponential execution time when applied to certain non-matching inputs.

- Evil regex can take exponential execution time when applied to certain non-matching inputs.
- By default, regex gets executed in event loop thread, so could be exploited for DoS attack.

- Evil regex pattern requirements:

( )+

1. Grouping with repetition, and
2. Inside repeated group, repeatation or alternation with operlapping

- Evil regex pattern requirements:

(a+)+

1. Grouping with repetition, and
2. Inside repeated group, repeatation or alternation with operlapping

- Evil regex pattern requirements:

(a|aa)+

1. Grouping with repetition, and
2. Inside repeated group, repeatation or alternation with overlapping

Example: Commonly used URL validator regex

```
/^(?!mailto:)(?:(?:https?|ftp):\/\/)?(?:\S+(?::\S*)?@)?(?:(?:(?:[1-9]\d?|1\d
\d|2[01]\d|22[0-3])(?:\.(?:1?\d{1,2}|2[0-4]\d|25[0-5])){2}(?:\.(?:
[0-9]\d?|1\d\d|2[0-4]\d|25[0-4]))|(?:(?:[a-z\u00a1-
\uffff0-9]+-?)*[a-z\u00a1-\uffff0-9]+)(?:\.(?:[a-z\u00a1-
\uffff0-9]+-?)*[a-z\u00a1-\uffff0-9]+)*(?:\.(?:[a-z\u00a1-\uffff]{2,})))|
localhost)(?::\d{2,5})?(?:\/[^\s]*)?$/i
```

Input pattern: aaaaaaaaaaaaaaa!

Example: Commonly used URL validator regex

# of Input Characters	Execution Time
30	6 sec
35	3min
36	6 min
37	13 min
38	25 min
39	1hr 28 min
40	3 hr 46 min

- Review regex in our own or external code for evil pattern
  Tools: RXRR, SDL Regex Fuzzer

- Review regex in our own or external code for evil pattern
  Tools: RXRR, SDL Regex Fuzzer

- Do not use user supplied inputs as regex

HTTP Parameter Pollution (HPP)

Quiz

```
// GET /search?firstname=John&firstname=John

req.query.firstname

//=>
```

```
// GET /search?firstname=John&firstname=John

req.query.firstname

//=> [“John”, “John”]
```

```
// POST firstname=John&firstname=John
```

```
// POST firstname=John&firstname=John

req.body.firstname

//=> [“John”, “John”]
```

Express populates HTTP request parameters with same name in an array

Express populates HTTP request parameters with same name in an array

Attacker can intentionally pollute request parameters to exploit this mechanism

An attacker can exploit HPP to:

- Trigger Type Errors in application

Server Console

```
Object John,John has no method 'trim'
TypeError: Object John,John has no method 'trim'
```

- Any uncaught errors in async code could crash the HTTP server causing DoS.

An attacker can exploit HPP to:

- Modify application behavior

DB Shell

```
> db.users.find({userName:"p"}).pretty()
{
        "_id" : ObjectId("53092b495ad132dfd56dbf70"),
        "address" : "",
        "dob" : "",
        "firstName" : [
                "John",
                "John"
        ],
        "lastName" : "Doe",
        "password" : "$2a$10$iZBDC45NulBco7s2GhCkJ.jU1Kr
        "ssn" : "1234",
        "userId" : 39,
        "userName" : "p"
}
```

An attacker can exploit HPP to:

- Bypass input validations applied on strings in our own code, WAF, browser filters.

An attacker can exploit HPP to:

- Bypass input validations applied on strings in our own code, WAF, browser filters.

```
> ["John", "John"] + " Doe"
"John,John Doe"
```

Check expected type as part of the input validation

- Check expected type as part of the input validation
- Implement robust error handling mechanism using try/catch, domain, and cluster.

OWASP Top 10

- Educate developers about OWASP Top 10 Risks

- Educate developers about OWASP Top 10 risks

OWASP Node Goat Project

Quick Recap

- Remove X-Powered-By response header and use generic session cookie names
- Keep watch on security vulnerabilities in dependencies

- Ensure HTTP GET requests are idempotent
- Include method-override module before any module that depends on method of the request

- Encode for all contexts on both server and client to protect against XSS attack.
- Use HTTPOnly and Secure attributes on session cookie, include CSP headers.

- Review regex for evil pattern to mitigate ReDoS attack.
- Verify input types as part of the validation

May Victory Be Yours.

Twitter:@ karande_c

Links

HP 2013 cyber risk report (http://www8.hp.com/h20195/v2/GetPDF.aspx/4AA5-0858ENW.pdf)
Node.js security vulnerabilities (http://blog.nodejs.org/vulnerability/)
Express security updates (http://expressjs.com/advanced/security-updates.html)
npm outdated (https://www.npmjs.org/doc/cli/npm-outdated.html)
Node Security Project (https://nodesecurity.io/advisories)
Retire.js(http://open.bekk.no/retire-js-what-you-require-you-must-also-retire)
RXRR (http://www.cs.bham.ac.uk/~hxt/research/rxxr-download.shtml)
SDL Regex Fuzzer (http://www.microsoft.com/en-us/download/details.aspx?id=20095)
OWASP ESAPI (https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API)
OWASP Node Goat Project (https://www.owasp.org/index.php/Projects/OWASP_Node_js_Goat_Project)

Image Credits

http://www.shutterstock.com/pic.mhtml?id=93406768
http://www.shutterstock.com/pic.mhtml?id=67916401
http://www.shutterstock.com/pic.mhtml?id=97398575
http://www.bigstockphoto.com/image-36498607
http://openclipart.org/detail/169260/medieval-cannon-by-helm42