ZAP by Checkmarx Scanning Report

Site: http://localhost:4000

Generated on Mon, 9 Jun 2025 15:47:58

ZAP Version: 2.16.1

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	5
Low	5
Informational	6

Alerts

Name	Risk Level	Number of Instances
Cross Site Scripting (Reflected)	High	2
CSP: Failure to Define Directive with No Fallback	Medium	16
Content Security Policy (CSP) Header Not Set	Medium	18
Directory Browsing	Medium	2
Missing Anti-clickjacking Header	Medium	17
Vulnerable JS Library	Medium	2
Application Error Disclosure	Low	2
Cookie without SameSite Attribute	Low	4
Cross-Domain JavaScript Source File Inclusion	Low	17
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	47
X-Content-Type-Options Header Missing	Low	26
Authentication Request Identified	Informational	2
Information Disclosure - Suspicious Comments	Informational	7
Modern Web Application	Informational	15
Session Management Response Identified	Informational	14
User Agent Fuzzer	Informational	120
User Controllable HTML Element Attribute (Potential XSS)	Informational	4

Alert Detail

High	Cross Site Scripting (Reflected)
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML /JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://localhost:4000/signup
Method	POST
Attack	"><scrIpt>alert(1);</scRipt>
Evidence	"><scrIpt>alert(1);</scRipt>
Other Info	
URL	http://localhost:4000/signup
Method	POST
Attack	"><scrIpt>alert(1);</scRipt>
Evidence	"><scrIpt>alert(1);</scRipt>
Other Info	
Instances	2
	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Solution	Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	https://owasp.org/www-community/attacks/xss/ https://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Plugin Id	40012

Medium	CSP: Failure to Define Directive with No Fallback
Description	The Content Security Policy fails to define one of the directives that has no fallback. Missing /excluding them is the same as allowing anything.
URL	http://localhost:4000/a
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/allocations/
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/app/views
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/body
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/div
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/form
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/h1
Method	GET
Attack	
Evidence	default-src 'self'
Other	The directive(s): frame-ancestors, form-action is/are among the directives that do not

Info	fallback to default-src.
URL	http://localhost:4000/h2
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/head
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/html
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/robots.txt
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/script
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/server.js
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/site/search?value
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/sitemap.xml

Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
URL	http://localhost:4000/span
Method	GET
Attack	
Evidence	default-src 'self'
Other Info	The directive(s): frame-ancestors, form-action is/are among the directives that do not fallback to default-src.
Instances	16
Solution	Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.
Reference	https://www.w3.org/TR/CSP/ https://caniuse.com/#search=content+security+policy https://content-security-policy.com/ https://github.com/HtmlUnit/htmlunit-csp https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources
CWE Id	693
WASC Id	15
Plugin Id	10055

Medium	**Content Security Policy (CSP) Header Not Set**
Description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/login
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/signup
Method	GET
Attack	
Evidence	

Other Info	
URL	http://localhost:4000/tutorial
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a1
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a10
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a2
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a3
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a4
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a5
Method	GET
Attack	
Evidence	
Other Info	

URL	http://localhost:4000/tutorial/a6
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a8
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/redos
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/ssrf
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/login
Method	POST
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/signup
Method	POST

Attack	
Evidence	
Other Info	
Instances	18
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.
Reference	https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html https://www.w3.org/TR/CSP/ https://w3c.github.io/webappsec-csp/ https://web.dev/articles/csp https://caniuse.com/#feat=contentsecuritypolicy https://content-security-policy.com/
CWE Id	693
WASC Id	15
Plugin Id	10038

Medium	**Directory Browsing**
Description	It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files, etc. which can be accessed to read sensitive information.
URL	http://localhost:4000/tutorial/
Method	GET
Attack	http://localhost:4000/tutorial/
Evidence	parent directory
Other Info	
URL	http://localhost:4000/tutorial/a1/
Method	GET
Attack	http://localhost:4000/tutorial/a1/
Evidence	parent directory
Other Info	
Instances	2
Solution	Disable directory browsing. If this is required, make sure the listed files does not induce risks.
Reference	https://httpd.apache.org/docs/mod/core.html#options
CWE Id	548
WASC Id	48
Plugin Id	0

Medium	**Missing Anti-clickjacking Header**
Description	The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.
URL	http://localhost:4000
Method	GET
Attack	

Evidence	
Other Info	
URL	http://localhost:4000/login
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/signup
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a1
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a10
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a2
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a3
Method	GET
Attack	
Evidence	
Other	

Info	
URL	http://localhost:4000/tutorial/a4
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a5
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a6
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a8
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/redos
Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/tutorial/ssrf

Method	GET
Attack	
Evidence	
Other Info	
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	
Other Info	
Instances	17
Solution	Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.
Reference	https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
CWE Id	1021
WASC Id	15
Plugin Id	10020

Medium	**Vulnerable JS Library**
Description	The identified library appears to be vulnerable.
URL	http://localhost:4000/vendor/bootstrap/bootstrap.js
Method	GET
Attack	
Evidence	* Bootstrap v3.0.0
Other Info	The identified library bootstrap, version 3.0.0 is vulnerable. CVE-2018-14041 CVE-2019-8331 CVE-2018-20677 CVE-2018-20676 CVE-2018-14042 CVE-2016-10735 CVE-2024-6484 CVE-2024-6485 https://nvd.nist.gov/vuln/detail/CVE-2024-6485 https://nvd.nist.gov/vuln/detail/CVE-2024-6484 https://github.com/advisories/GHSA-pj7m-g53m-7638 https://github.com/advisories/GHSA-vxmc-5x29-h64v https://github.com/rubysec/ruby-advisory-db/blob/master/gems/bootstrap-sass/CVE-2024-6484.yml https://github.com/twbs/bootstrap/issues/20631 https://github.com/advisories/GHSA-9mvj-f7w8-pvh2 https://github.com/advisories/GHSA-9v3m-8fp8-mj99 https://github.com/twbs/bootstrap/issues/28236 https://www.herodevs.com/vulnerability-directory/cve-2024-6485 https://github.com/twbs/bootstrap/issues/20184 https://www.herodevs.com/vulnerability-directory/cve-2024-6484 https://github.com/advisories/GHSA-ph58-4vrj-w6hr https://github.com/twbs/bootstrap https://github.com/advisories/GHSA-4p24-vmcr-4gqj https://github.com/rubysec/ruby-advisory-db/blob/master/gems/bootstrap/CVE-2024-6484.yml https://nvd.nist.gov/vuln/detail/CVE-2018-20676
URL	http://localhost:4000/vendor/jquery.min.js
Method	GET
Attack	
Evidence	/*! jQuery v1.10.2
	The identified library jquery, version 1.10.2 is vulnerable. CVE-2020-11023 CVE-2020-11022 CVE-2015-9251 CVE-2019-11358 https://github.com/jquery/jquery/issues/2432 http://blog.jquery.com/2016/01/08/jquery-2-2-and-1-12-released/ http://research.

Other Info	insecurelabs.org/jquery/test/ https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ https://nvd.nist.gov/vuln/detail/CVE-2019-11358 https://github.com/advisories/GHSA-rmxg-73gg-4p98 https://nvd.nist.gov/vuln/detail/CVE-2015-9251 https://github.com/jquery/jquery/commit/753d591aea698e57d6db58c9f722cd0808619b1b https://bugs.jquery.com/ticket/11974 https://github.com/jquery/jquery.com/issues/162 https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/
Instances	2
Solution	Upgrade to the latest version of the affected library.
Reference	https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
CWE Id	1395
WASC Id	
Plugin Id	10003

Low	**Application Error Disclosure**
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://localhost:4000/login
Method	POST
Attack	
Evidence	HTTP/1.1 500 Internal Server Error
Other Info	
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	HTTP/1.1 500 Internal Server Error
Other Info	
Instances	2
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	550
WASC Id	13
Plugin Id	90022

Low	**Cookie without SameSite Attribute**
Description	A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	set-cookie: connect.sid
Other Info	

URL	http://localhost:4000/
Method	GET
Attack	
Evidence	set-cookie: connect.sid
Other Info	
URL	http://localhost:4000/robots.txt
Method	GET
Attack	
Evidence	set-cookie: connect.sid
Other Info	
URL	http://localhost:4000/sitemap.xml
Method	GET
Attack	
Evidence	set-cookie: connect.sid
Other Info	
Instances	4
Solution	Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.
Reference	https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site
CWE Id	1275
WASC Id	13
Plugin Id	10054

Low	**Cross-Domain JavaScript Source File Inclusion**
Description	The page includes one or more script files from a third-party domain.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/login
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/signup
Method	GET
Attack	
	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" +

Evidence	"script>");</script>
Other Info	
URL	http://localhost:4000/tutorial
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a1
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a10
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a2
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a3
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a4
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a5
Method	GET

Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a6
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a8
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/redos
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
URL	http://localhost:4000/tutorial/ssrf
Method	GET
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	

URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>
Other Info	
Instances	17
Solution	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.
Reference	
CWE Id	829
WASC Id	15
Plugin Id	10017

Low	**Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)**
Description	The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/a
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/allocations/
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/app/views
Method	GET

Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/body
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/div
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/favicon.ico
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/form
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/h1
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/h2
Method	GET
Attack	
Evidence	X-Powered-By: Express

Other Info	
URL	http://localhost:4000/head
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/html
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/images/owasplogo.png
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/login
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/robots.txt
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	

URL	http://localhost:4000/script
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/server.js
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/signup
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/site/search?value
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/sitemap.xml
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/span
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a1

Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a10
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a2
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a3
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a4
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a5
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a6
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	

Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a8
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/redos
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/tutorial/ssrf
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/vendor/bootstrap/bootstrap.css
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/vendor/bootstrap/bootstrap.js
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/vendor/html5shiv.js
Method	GET
Attack	
Evidence	X-Powered-By: Express

Other Info	
URL	http://localhost:4000/vendor/jquery.min.js
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css/font-awesome.min.css
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts/fontawesome-webfont.woff?v=4.0.1
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/vendor/theme/sb-admin.css
Method	GET
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/login
Method	POST
Attack	
Evidence	X-Powered-By: Express
Other Info	
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	X-Powered-By: Express
Other Info	
Instances	47
Solution	Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.
Reference	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/08-Fingerprint_Web_Application_Framework https://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html
CWE Id	497

WASC Id	13
Plugin Id	10037

Low	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/favicon.ico
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/images/owasplogo.png
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/login
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/signup
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.

URL	http://localhost:4000/tutorial
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a1
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a10
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a2
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a3
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a4
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.

URL	http://localhost:4000/tutorial/a5
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a6
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a8
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/redos
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/tutorial/ssrf

Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/bootstrap/bootstrap.css
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/bootstrap/bootstrap.js
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/html5shiv.js
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/jquery.min.js
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/theme/font-awesome/css/font-awesome.min.css
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/theme/font-awesome/fonts/fontawesome-webfont.woff?v=4.0.1

Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/vendor/theme/sb-admin.css
Method	GET
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	
Other Info	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
Instances	26
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application /web server to not perform MIME-sniffing.
Reference	https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85) https://owasp.org/www-community/Security_Headers
CWE Id	693
WASC Id	15
Plugin Id	10021

Informational	**Authentication Request Identified**
Description	The given request has been identified as an authentication request. The 'Other Info' field contains a set of key=value lines which identify any relevant fields. If the request is in a context which has an Authentication Method set to "Auto-Detect" then this rule will change the authentication to match the request identified.
URL	http://localhost:4000/login
Method	POST
Attack	
Evidence	password
Other Info	userParam=userName userValue=QgnoyoPYvDjcocSGsIdouzwv passwordParam=password referer=http://localhost:4000/login csrfToken=_csrf
URL	http://localhost:4000/login
Method	POST

Attack	
Evidence	password
Other Info	userParam=userName userValue=ZAP passwordParam=password referer=http://localhost:4000/login csrfToken=_csrf
Instances	2
Solution	This is an informational alert rather than a vulnerability and so there is nothing to fix.
Reference	https://www.zaproxy.org/docs/desktop/addons/authentication-helper/auth-req-id/
CWE Id	
WASC Id	
Plugin Id	10111

Informational	**Information Disclosure - Suspicious Comments**
Description	The response appears to contain suspicious comments which may help an attacker.
URL	http://localhost:4000/tutorial
Method	GET
Attack	
Evidence	userName
Other Info	The following pattern was used: \bUSERNAME\b and was detected in likely comment: "//localhost:4000/login -X POST --data 'userName=vyva%0aError: alex moldovan failed $1,000,000 transaction&password=Admin_123&_cs", see evidence field for the suspicious comment/snippet.
URL	http://localhost:4000/tutorial/a1
Method	GET
Attack	
Evidence	userName
Other Info	The following pattern was used: \bUSERNAME\b and was detected in likely comment: "//localhost:4000/login -X POST --data 'userName=vyva%0aError: alex moldovan failed $1,000,000 transaction&password=Admin_123&_cs", see evidence field for the suspicious comment/snippet.
URL	http://localhost:4000/tutorial/a2
Method	GET
Attack	
Evidence	user
Other Info	The following pattern was used: \bUSER\b and was detected in likely comment: "// Create user document", see evidence field for the suspicious comment/snippet.
URL	http://localhost:4000/tutorial/a5
Method	GET
Attack	
Evidence	from
Other Info	The following pattern was used: \bFROM\b and was detected in likely comment: "// Prevent opening page in frame or iframe to protect from clickjacking", see evidence field for the suspicious comment/snippet.
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	
Evidence	admin

Other Info	The following pattern was used: \bADMIN\b and was detected in likely comment: " //Middleware to check if user has admin rights", see evidence field for the suspicious comment/snippet.
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
Evidence	later
Other Info	The following pattern was used: \bLATER\b and was detected in likely comment: "//docs. npmjs.com/cli/v6/commands/npm-audit">npm audit is a vulnerability scanner built into the npm CLI (version 6 or later)", see evidence field for the suspicious comment/snippet.
URL	http://localhost:4000/vendor/jquery.min.js
Method	GET
Attack	
Evidence	username
Other Info	The following pattern was used: \bUSERNAME\b and was detected in likely comment: "//, En=/^([\w.+-]+:)(?:\/\/([^\/?#:]*)(?::(\d+)|)|)/,Sn=x.fn.load,An={},jn={},Dn="*/".concat("*");try {yn=o.href}catch(Ln){yn=a.cr", see evidence field for the suspicious comment/snippet.
Instances	7
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Reference	
CWE Id	615
WASC Id	13
Plugin Id	10027

Informational	**Modern Web Application**
Description	The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	
Other Info	Links have been found that do not have traditional href attributes, which is an indication that this is a modern web application.
URL	http://localhost:4000/login
Method	GET
Attack	
Evidence	
Other Info	Links have been found that do not have traditional href attributes, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial
Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display -->

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation   </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection   A2 Broken Auth   A3 XSS   A4 Insecure DOR   A5 Misconfig   A6 Sensitive Data   A7 Access Controls   A8 CSRF   A9 Insecure Components   A10 Redirects   ReDoS Attacks   SSRF   <ul class="nav navbar-nav navbar-right navbar-user"> Exit   <!-- /.navbar-collapse --> </nav>   <h1>A1 - Injection <small></small> </h1>   <!-- /.row -->   Exploitability: EASY Prevalence: COMMON Detectability: AVERAGE Technical Impact: SEVERE < /div>   <h3 class="panel-title">Description</h3>   Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.   <!--   <h3 class="panel-title">Real World Attack Incident Examples</h3>   Screencast here ... < /div> -->   <!-- accordions -->   <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href="#collapseOne"> A1 - 1 Server Side JS Injection </h4>   <h3 class="panel-title">Description</h3>   When <code>eval()</code>, <code>setTimeout()</code>, <code>setInterval()</code>, <code>Function()</code>are used to process user provided inputs, it can be exploited by an attacker to inject and execute malicious JavaScript code on server.   <h3 class="panel-title">Attack Mechanics</h3>   Web applications using the JavaScript <code>eval()</code>function to parse the incoming data without any type of input validation are vulnerable to this attack. An attacker can inject arbitrary JavaScript code to be executed on the server. Similarly <code>setTimeout()< /code>, and <code>setInterval()</code>functions can take code in string format as a first argument causing same issues as <code>eval()</code>. This vulnerability can be very critical and damaging by allowing attacker to send various types of commands. Denial of Service Attack: <iframe width="560" height="315" src="//www.youtube.com/embed/krOx9QWwcYw?rel=0" frameborder="0" allowfullscreen></iframe> An effective denial-of-service attack can be executed simply by sending the commands below to <code>eval()</code>function: <pre>while(1)</pre> This input will cause the target server's event loop to use 100% of its processor time and unable to process any other incoming requests until process is restarted.   An alternative DoS attack would be to simply exit or kill the running process: <pre>process.exit()</pre> or <pre>process.kill(process.pid) </pre>   File System Access
  <iframe width="560" height="315" src="//www.youtube.com/embed/Mr-Jh9bjSLo?rel=0" frameborder="0" allowfullscreen></iframe> Another potential goal of an attacker might be to read the contents of files from the server. For example, following two commands list the contents of the current directory and parent directory respectively: <pre>res.end(require('fs').readdirSync('.').toString())</pre> <pre>res.end(require('fs').readdirSync('..').toString()) </pre>   Once file names are obtained, an attacker can issue the command below to view the actual contents of a file: <pre>res.end(require('fs').readFileSync(filename))</pre>   An attacker can further exploit this vulnerability by writing and executing harmful binary files using <code>fs</code>and <code>child_process< /code>modules.   <h3 class="panel-title">How Do I Prevent It?</h3>   To prevent server-side js injection attacks: <ul> Validate user inputs on

Evidence	server side before processing - Do not use <code>eval()</code>function to parse user inputs. Avoid using other commands with similar effect, such as <code>setTimeOut()< /code>, <code>setInterval()</code>, and <code>Function()</code>. - For parsing JSON input, instead of using <code>eval()</code>, use a safer alternative such as <code>JSON.parse()</code>. For type conversions use type related <code>parseXXX()< /code>methods. - Include <code>"use strict"</code>at the beginning of a function, which enables strict mode within the enclosing function scope. <h3 class="panel-title">Source Code Example</h3> In <code>routes/contributions.js</code>, the <code>handleContributionsUpdate()</code>function insecurely uses <code>eval()< /code>to convert user supplied contribution amounts to integer. <pre> // Insecure use of eval() to parse inputs var preTax = eval(req.body.preTax); var afterTax = eval(req.body. afterTax); var roth = eval(req.body.roth); </pre> This makes application vulnerable to SSJS attack. It can fixed simply by using <code>parseInt()</code>instead. <pre> //Fix for A1 -1 SSJS Injection attacks - uses alternate method to eval var preTax = parseInt(req.body. preTax); var afterTax = parseInt(req.body.afterTax); var roth = parseInt(req.body.roth); < /pre> In addition, all functions begin with <code>use strict</code>pragma. < /div> <h3 class="panel-title" >Further Reading</h3> “ServerSide JavaScript Injection: Attacking NoSQL and Node.js" a whitepaper by Bryan Sullivan. <!-- /ssjs --> <!-- DB Injection --> <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href="#collapseTwo"> A1 - 2 SQL and NoSQL Injection </h4> <h3 class="panel-title">Description</h3> SQL and NoSQL injections enable an attacker to inject code into the query that would be executed by the database. These flaws are introduced when software developers create dynamic database queries that include user supplied input. <h3 class="panel-title">Attack Mechanics</h3> Both SQL and NoSQL databases are vulnerable to injection attack. Here is an example of equivalent attack in both cases, where attacker manages to retrieve admin user's record without knowing password: <h5>1. SQL Injection</h5> Lets consider an example SQL statement used to authenticate the user with username and password <pre>SELECT * FROM accounts WHERE username = '$username' AND password = '$password'</pre> If this statement is not prepared or properly handled when constructed, an attacker may be able to supply <code>admin' --</code>in the username field to access the admin user's account bypassing the condition that checks for the password. The resultant SQL query would looks like: <pre>SELECT * FROM accounts WHERE username = 'admin' -- AND password = ''</pre> <h5>2. NoSQL Injection</h5> The equivalent of above query for NoSQL MongoDB database is: <pre>db.accounts.find({username: username, password: password});</pre> While here we are no longer dealing with query language, an attacker can still achieve the same results as SQL injection by supplying JSON input object as below: <pre> { "username": "admin", "password": {$gt: ""} } </pre> In MongoDB, <code>$gt</code>selects those documents where the value of the field is greater than (i.e. >) the specified value. Thus above statement compares password in database with empty string for greatness, which returns <code>true</code>. The same results can be achieved using other comparison operator such as <code>$ne</code>. <h3 class="panel-title" >SSJS Attack Mechanics</h3> Server-side JavaScript Injection (SSJS) is an attack where JavaScript code is injected and executed in a server component. MongoDB specifically, is vulnerable to this attack when queries are run without proper sanitization. <h5>$where operator</h5> MongoDB's <code>$where< /code> operator performs JavaScript expression evaluation on the MongoDB server. If the user is able to inject direct code into such queries then such an attack can take place Lets consider an example query: <pre> db.allocationsCollection.find({ $where: "this.userId == '" + parsedUserId + "' && " + "this.stocks > " + "'" + threshold + "'" }); </pre> The code will match all documents which have a <code>userId</code> field as specified by <code>parsedUserId</code> and a <code>stocks</code> field as specified by <code>threshold</code>. The problem is that these parameters are not validated, filtered, or sanitised, and vulnerable to SSJS Injection. <h3 class="panel-title">How Do I Prevent It?</h3> < /div> Here are some measures to prevent SQL / NoSQL injection attacks, or minimize impact if it happens: - Prepared Statements: For SQL calls, use

	prepared statements instead of building dynamic queries using string concatenation. - Input Validation: Validate inputs to detect malicious values. For NoSQL databases, also validate input types against expected types - Least Privilege: To minimize the potential damage of a successful injection attack, do not assign DBA or admin type access rights to your application accounts. Similarly minimize the privileges of the operating system account that the database process runs under. <h3 class="panel-title">Source Code Example</h3> Note: These vulnerabilities are not present when using an Atlas M0 cluster with NodeGoat. The Allocations page of the demo application is vulnerable to NoSQL Injection. For example, set the stocks threshold filter to: <pre>1'; return 1 == '1</pre> This will retrieve allocations for all the users in the database. An attacker could also send the following input for the <code>threshold</code> field in the request's query, which will create a valid JavaScript expression and satisfy the <code> $where</code> query as well, resulting in a DoS attack on the MongoDB server: <pre>http://localhost:4000 /allocations/2?threshold=5';while(true){};' </pre> You can also just drop the following into the Stocks Threshold input box: <pre>';while(true){};'</pre> For these vulnerabilities, bare minimum fixes can be found in <code>allocations.html</code> and <code>allocations-dao.js</code> <!-- /NoSQL Injection --> <!-- Log Injection --> <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href=" #collapseThree"> A1 - 3 Log Injection </h4> <h3 class="panel-title">Description</h3> Log injection vulnerabilities enable an attacker to forge and tamper with an application's logs. <h3 class="panel-title">Attack Mechanics</h3> An attacker may craft a malicious request that may deliberately fail, which the application will log, and when attacker's user input is unsanitized, the payload is sent as-is to the logging facility. Vulnerabilities may vary depending on the logging facility: <h5>1. Log Forging (CRLF) </h5> Lets consider an example where an application logs a failed attempt to login to the system. A very common example for this is as follows: < /p> <pre> var userName = req.body.userName; console.log('Error: attempt to login with invalid user: ', userName); </pre> When user input is unsanitized and the output mechanism is an ordinary terminal stdout facility then the application will be vulnerable to CRLF injection, where an attacker can create a malicious payload as follows: <pre> curl http://localhost:4000/login -X POST --data 'userName=vyva%0aError: alex moldovan failed $1,000,000 transaction&password=Admin_123&_csrf=' </pre> Where the <code>userName</code> parameter is encoding in the request the LF symbol which will result in a new line to begin. Resulting log output will look as follows: <pre> Error: attempt to login with invalid user: vyva Error: alex moldovan failed $1,000,000 transaction </pre> <h5>2. Log Injection Escalation </h5> An attacker may craft malicious input in hope of an escalated attack where the target isn't the logs themselves, but rather the actual logging system. For example, if an application has a back-office web app that manages viewing and tracking the logs, then an attacker may send an XSS payload into the log, which may not result in log forging on the log itself, but when viewed by a system administrator on the log viewing web app then it may compromise it and result in XSS injection that if the logs app is vulnerable. <h3 class="panel-title">How Do I Prevent It?</h3> As always when dealing with user input: - Do not allow user input into logs - Encode to proper context, or sanitize user input Encoding example: <pre> // Step 1: Require a module that supports encoding var ESAPI = require('node-esapi'); // - Step 2: Encode the user input that will be logged in the correct context // following are a few examples: console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder(). encodeForHTML(userName)); console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForJavaScript(userName)); console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForURL(userName)); </pre> <h3 class="panel-title">Source Code Example</h3> For the above Log Injection vulnerability, example and fix can be found at <code>routes/session.js</code> < /div> <!-- /Log Injection --> <!-- end accordions --> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a1
Method	GET

Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav"> A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /.navbar-collapse --> </nav> <h1>A1 - Injection <small></small> </h1> <!-- /.row --> Exploitability: EASY Prevalence: COMMON Detectability: AVERAGE Technical Impact: SEVERE <h3 class="panel-title">Description</h3> Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. <!-- <h3 class="panel-title">Real World Attack Incident Examples</h3> Screencast here ... --> <!-- accordions --> <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href="#collapseOne"> A1 - 1 Server Side JS Injection </h4> <h3 class="panel-title">Description</h3> When <code>eval()</code>, <code>setTimeout()</code>, <code>setInterval()</code>, <code>Function()</code>are used to process user provided inputs, it can be exploited by an attacker to inject and execute malicious JavaScript code on server. <h3 class="panel-title">Attack Mechanics</h3> Web applications using the JavaScript <code>eval()</code>function to parse the incoming data without any type of input validation are vulnerable to this attack. An attacker can inject arbitrary JavaScript code to be executed on the server. Similarly <code>setTimeout()</code>, and <code>setInterval()</code>functions can take code in string format as a first argument causing same issues as <code>eval()</code>. This vulnerability can be very critical and damaging by allowing attacker to send various types of commands. Denial of Service Attack: <iframe width="560" height="315" src="//www.youtube.com/embed/krOx9QWwcYw?rel=0" frameborder="0" allowfullscreen></iframe> An effective denial-of-service attack can be executed simply by sending the commands below to <code>eval()</code>function: <pre>while(1)</pre> This input will cause the target server's event loop to use 100% of its processor time and unable to process any other incoming requests until process is restarted. An alternative DoS attack would be to simply exit or kill the running process: <pre>process.exit()</pre> or <pre>process.kill(process.pid) </pre> File System Access <iframe width="560" height="315" src="//www.youtube.com/embed/Mr-Jh9bjSLo?rel=0" frameborder="0" allowfullscreen></iframe> Another potential goal of an attacker might be to read the contents of files from the server. For example, following two commands list the contents of the current directory and parent directory respectively: <pre>res.end(require('fs').readdirSync('.').toString())</pre> <pre>res.end(require('fs').readdirSync('..').toString()) </pre> Once file names are obtained, an attacker can issue the command below to view the actual contents of a file: <pre>res.end(require('fs').

Evidence	readFileSync(filename))</pre> An attacker can further exploit this vulnerability by writing and executing harmful binary files using <code>fs</code>and <code>child_process</code>modules. <h3 class="panel-title">How Do I Prevent It?</h3> To prevent server-side js injection attacks: - Validate user inputs on server side before processing - Do not use <code>eval()</code>function to parse user inputs. Avoid using other commands with similar effect, such as <code>setTimeOut()</code>, <code>setInterval()</code>, and <code>Function()</code>. - For parsing JSON input, instead of using <code>eval()</code>, use a safer alternative such as <code>JSON.parse()</code>. For type conversions use type related <code>parseXXX()</code>methods. - Include <code>"use strict"</code>at the beginning of a function, which enables strict mode within the enclosing function scope. <h3 class="panel-title">Source Code Example</h3> In <code>routes/contributions.js</code>, the <code>handleContributionsUpdate()</code>function insecurely uses <code>eval()</code>to convert user supplied contribution amounts to integer. <pre> // Insecure use of eval() to parse inputs var preTax = eval(req.body.preTax); var afterTax = eval(req.body.afterTax); var roth = eval(req.body.roth); </pre> This makes application vulnerable to SSJS attack. It can fixed simply by using <code>parseInt()</code>instead. <pre> //Fix for A1 -1 SSJS Injection attacks - uses alternate method to eval var preTax = parseInt(req.body.preTax); var afterTax = parseInt(req.body.afterTax); var roth = parseInt(req.body.roth); </pre> In addition, all functions begin with <code>use strict</code>pragma. <h3 class="panel-title">Further Reading</h3> “ServerSide JavaScript Injection: Attacking NoSQL and Node.js" a whitepaper by Bryan Sullivan. <!-- /ssjs --> <!-- DB Injection --> <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href="#collapseTwo"> A1 - 2 SQL and NoSQL Injection </h4> <h3 class="panel-title">Description</h3> SQL and NoSQL injections enable an attacker to inject code into the query that would be executed by the database. These flaws are introduced when software developers create dynamic database queries that include user supplied input. <h3 class="panel-title">Attack Mechanics</h3> Both SQL and NoSQL databases are vulnerable to injection attack. Here is an example of equivalent attack in both cases, where attacker manages to retrieve admin user's record without knowing password: <h5>1. SQL Injection</h5> Lets consider an example SQL statement used to authenticate the user with username and password <pre>SELECT * FROM accounts WHERE username = '$username' AND password = '$password'</pre> If this statement is not prepared or properly handled when constructed, an attacker may be able to supply <code>admin' --</code>in the username field to access the admin user's account bypassing the condition that checks for the password. The resultant SQL query would looks like: <pre>SELECT * FROM accounts WHERE username = 'admin' -- AND password = ''</pre>
	 <h5>2. NoSQL Injection</h5> The equivalent of above query for NoSQL MongoDB database is: <pre>db.accounts.find({username: username, password: password});</pre> While here we are no longer dealing with query language, an attacker can still achieve the same results as SQL injection by supplying JSON input object as below: <pre> { "username": "admin", "password": {$gt: ""} } </pre> In MongoDB, <code>$gt</code>selects those documents where the value of the field is greater than (i.e. >) the specified value. Thus above statement compares password in database with empty string for greatness, which returns <code>true</code>. The same results can be achieved using other comparison operator such as <code>$ne</code>. <h3 class="panel-title">SSJS Attack Mechanics</h3> Server-side JavaScript Injection (SSJS) is an attack where JavaScript code is injected and executed in a server component. MongoDB specifically, is vulnerable to this attack when queries are run without proper sanitization. <h5>$where operator</h5> MongoDB's <code>$where</code> operator performs JavaScript expression evaluation on the MongoDB server. If the user is able to inject direct code into such queries then such an attack can take place Lets consider an example query: <pre> db.allocationsCollection.find({ $where: "this.userId == '" + parsedUserId + "' && " + "this.stocks > " + "'" + threshold + "'" }); </pre> The code will match all documents which have a <code>userId</code> field as specified by <code>parsedUserId</code> and a <code>stocks</code> field as specified by

<code>threshold</code>. The problem is that these parameters are not validated, filtered, or sanitised, and vulnerable to SSJS Injection.   <h3 class="panel-title">How Do I Prevent It?</h3> < /div> Here are some measures to prevent SQL / NoSQL injection attacks, or minimize impact if it happens: <ul> Prepared Statements: For SQL calls, use prepared statements instead of building dynamic queries using string concatenation. Input Validation: Validate inputs to detect malicious values. For NoSQL databases, also validate input types against expected types Least Privilege: To minimize the potential damage of a successful injection attack, do not assign DBA or admin type access rights to your application accounts. Similarly minimize the privileges of the operating system account that the database process runs under.   <h3 class="panel-title">Source Code Example</h3>   Note: These vulnerabilities are not present when using an Atlas M0 cluster with NodeGoat. The Allocations page of the demo application is vulnerable to NoSQL Injection. For example, set the stocks threshold filter to: <pre>1'; return 1 == '1</pre> This will retrieve allocations for all the users in the database. An attacker could also send the following input for the <code>threshold</code> field in the request's query, which will create a valid JavaScript expression and satisfy the <code> $where</code> query as well, resulting in a DoS attack on the MongoDB server: <pre>http://localhost:4000 /allocations/2?threshold=5';while(true){};' </pre> You can also just drop the following into the Stocks Threshold input box: <pre>';while(true){};'</pre> For these vulnerabilities, bare minimum fixes can be found in <code>allocations.html</code> and <code>allocations-dao.js</code>   <!-- /NoSQL Injection --> <!-- Log Injection -->   <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href=" #collapseThree"> A1 - 3 Log Injection </h4>   <h3 class="panel-title">Description</h3>   Log injection vulnerabilities enable an attacker to forge and tamper with an application's logs.   <h3 class="panel-title">Attack Mechanics</h3>   An attacker may craft a malicious request that may deliberately fail, which the application will log, and when attacker's user input is unsanitized, the payload is sent as-is to the logging facility. Vulnerabilities may vary depending on the logging facility: <h5>1. Log Forging (CRLF) </h5> Lets consider an example where an application logs a failed attempt to login to the system. A very common example for this is as follows: < /p> <pre> var userName = req.body.userName; console.log('Error: attempt to login with invalid user: ', userName); </pre> When user input is unsanitized and the output mechanism is an ordinary terminal stdout facility then the application will be vulnerable to CRLF injection, where an attacker can create a malicious payload as follows: <pre> curl http://localhost:4000/login -X POST --data 'userName=vyva%0aError: alex moldovan failed $1,000,000 transaction&password=Admin_123&_csrf=' </pre> Where the <code>userName</code> parameter is encoding in the request the LF symbol which will result in a new line to begin. Resulting log output will look as follows: <pre> Error: attempt to login with invalid user: vyva Error: alex moldovan failed $1,000,000 transaction </pre>
 <h5>2. Log Injection Escalation </h5> An attacker may craft malicious input in hope of an escalated attack where the target isn't the logs themselves, but rather the actual logging system. For example, if an application has a back-office web app that manages viewing and tracking the logs, then an attacker may send an XSS payload into the log, which may not result in log forging on the log itself, but when viewed by a system administrator on the log viewing web app then it may compromise it and result in XSS injection that if the logs app is vulnerable.   <h3 class="panel-title">How Do I Prevent It?</h3>   As always when dealing with user input: <ul> Do not allow user input into logs   Encode to proper context, or sanitize user input   Encoding example: <pre> // Step 1: Require a module that supports encoding var ESAPI = require('node-esapi'); // - Step 2: Encode the user input that will be logged in the correct context // following are a few examples: console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder(). encodeForHTML(userName)); console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForJavaScript(userName)); console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForURL(userName)); </pre>   <h3 class="panel-title">Source Code Example</h3>   For the above Log Injection vulnerability, example and fix can be found at <code>routes/session.js</code> < /div>   <!-- /Log Injection --> <!-- end accordions --> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>

Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a10
Method	GET
Attack	
Evidence	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav"> A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /.navbar-collapse --> </nav> <h1>A10-Unvalidated Redirects and Forwards <small></small> </h1> <!-- /.row --> Exploitability: AVERAGE Prevalence: COMMON Detectability: EASY Technical Impact: MODERATE <h3 class="panel-title">Description</h3> Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages. <h3 class="panel-title">Attack Mechanics</h3> An attacker can use unvalidated redirected links as a medium to redirect user to malicious contents and tricks victims into clicking it. Attacker can exploit it to bypass security checks and make it believe trustworthy. For example, the "Learning Resources" link (<code>/learn?url=...</code>) in the application redirects to another website without validating the url. <iframe width="560" height="315" src="//www.youtube.com/embed/z98AQF8J_zg?rel=0" frameborder="0" allowfullscreen></iframe> Here is code from <code>routes/index.js</code>, <pre> // Handle redirect for learning resources link app.get("/learn", function (req, res, next) { return res.redirect(req.query.url); }); </pre> An attacker can change the <code>url</code>query parameter to point to malicious website and share it. Victims are more likely to click on it, as the initial part of the link (before query parameters) points to a trusted site. <h3 class="panel-title">How Do I Prevent It?</h3> Safe use of redirects and forwards can be done in a number of ways: - Simply avoid using redirects and forwards. - If used, don’t involve user parameters in calculating the destination. This can usually be done. - If destination parameters can’t be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL. <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a2

Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav"> A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /.navbar-collapse --> </nav> <h1>A2-Broken Authentication and Session Management <small></small> </h1> <!-- /.row --> Exploitability: AVERAGE Prevalence: WIDESPREAD Detectability: AVERAGE Technical Impact: SEVERE <h3 class="panel-title">Description</h3> In this attack, an attacker (who can be anonymous external attacker, a user with own account who may attempt to steal data from accounts, or an insider wanting to disguise his or her actions) uses leaks or flaws in the authentication or session management functions to impersonate other users. Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities. Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique. <!-- <h3 class="panel-title">Real World Attack Incident Examples</h3> Screencast here ... --> <!-- accordions --> <h4 class="panel-title"> <a data-toggle="collapse" data-parent="#accordion" href="#collapseTwo"> A2 - 1 Session Management </h4> <h3 class="panel-title">Description</h3> Session management is a critical piece of application security. It is broader risk, and requires developers take care of protecting session id, user credential secure storage, session duration, and protecting critical session data in transit. <h3 class="panel-title">Attack Mechanics</h3> Scenario #1: Application timeouts aren't set properly. User uses a public computer to access site. Instead of selecting “logout” the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated. Scenario #2: Attacker acts as a man-in-middle and acquires user's session id from network traffic. Then uses this authenticated session id to connect to application without needing to enter user name and password. Scenario #3: Insider or external attacker gains access to the system's password database. User passwords are not properly hashed, exposing every users' password to the attacker. <h3 class="panel-title">How Do I Prevent It?</h3> Session management related security issues can be prevented by

Evidence	taking these measures: - User authentication credentials should be protected when stored using hashing or encryption. - Session IDs should not be exposed in the URL (e.g., URL rewriting). - Session IDs should timeout. User sessions or authentication tokens should get properly invalidated during logout. - Session IDs should be recreated after successful login. - Passwords, session IDs, and other credentials should not be sent over unencrypted connections. <h3 class="panel-title">Source Code Examples</h3> In the insecure demo app, following issues exists: <h3>1. Protecting user credentials</h3> password gets stored in database in plain text . Here is related code in <code>data/user-dao.js</code> <code>addUser()</code>method: <pre> // Create user document var user = { userName: userName, firstName: firstName, lastName: lastName, password: password //received from request param }; </pre> To secure it, handle password storage in a safer way by using one way encryption using salt hashing as below: <pre> // Generate password hash var salt = bcrypt.genSaltSync(); var passwordHash = bcrypt.hashSync(password, salt); // Create user document var user = { userName: userName, firstName: firstName, lastName: lastName, password: passwordHash }; </pre> This hash password can not be decrypted, hence more secure. To compare the password when user logs in, the user entered password gets converted to hash and compared with the hash in storage. <pre> if (bcrypt. compareSync(password, user.password)) { callback(null, user); } else { callback (invalidPasswordError, null); } </pre> Note: The bcrypt module also provides asynchronous methods for creating and comparing hash. <h3>2. Session timeout and protecting cookies in transit</h3> The insecure demo application does not contain any provision to timeout user session. The session stays active until user explicitly logs out. In addition to that, the app does not prevent cookies being accessed in script, making application vulnerable to Cross Site Scripting (XSS) attacks. Also cookies are not prevented to get sent on insecure HTTP connection. To secure the application: 1. Use session based timeouts, terminate session when browser closes. <pre> // Enable session management using express middleware app.use(express.cookieParser()); </pre> 2. In addition, sets <code>HTTPOnly</code>HTTP header preventing cookies being accessed by scripts. The application used HTTPS secure connections, and cookies are configured to be sent only on Secure HTTPS connections by setting <code>Secure< /code>flag. <pre> app.use(express.session({ secret: "s3Cur3", cookie: { httpOnly: true, secure: true } })); </pre> 3. When user clicks logout, destroy the session and session cookie <pre> req.session.destroy(function() { res.redirect("/"); }); </pre> Note: The example code uses <code>MemoryStore</code>to manage session data, which is not designed for production environment, as it will leak memory, and will not scale past a single process. Use database based storage MongoStore or RedisStore for production. Alternatively, sessions can be managed using popular passport module. <h3>3. Session hijacking</h3> The insecure demo application does not regenerate a new session id upon user's login, therefore rendering a vulnerability of session hijacking if an attacker is able to somehow steal the cookie with the session id and use it. Upon login, a security best practice with regards to cookies session management would be to regenerate the session id so that if an id was already created for a user on an insecure medium (i.e: non-HTTPS website or otherwise), or if an attacker was able to get their hands on the cookie id before the user logged-in, then the old session id will render useless as the logged-in user with new privileges holds a new session id now. To secure the application: 1. Re-generate a new session id upon login (and best practice is to keep regenerating them upon requests or at least upon sensitive actions like a user's password reset. Re-generate a session id as follows: By wrapping the below code as a function callback for the method req.session.regenerate() <pre> req.session.regenerate (function() { req.session.userId = user._id; if (user.isAdmin) { return res.redirect("/benefits"); } else { return res.redirect("/dashboard"); } }) </pre> <h3 class="panel-title">Further Reading</h3> Helmet Security header middleware collection for express Seven Web Server HTTP Headers that Improve Web Application Security for Free Passport authentication middleware< /li> CWE-384: Session Fixation< /a> <!-- /Session Management --> <h4 class="panel-title"> <a data-toggle=" collapse" data-parent="#accordion" href="#collapseOne"> <i class="fa fa-chevron-down"><> /i> A2 - 2 Password Guessing Attacks </h4> <h3 class="panel-title">Description</h3> Implementing a robust minimum password criteria (minimum length and complexity) can make it difficult for attacker to guess password. <!-- <div

	class="panel panel-info"> <h3 class="panel-title">Attack Scenario Demo</h3> Screencast showing how attack can manifest in the target application ... --> <h3 class="panel-title">Attack Mechanics</h3> The attacker can exploit this vulnerability by brute force password guessing, more likely using tools that generate random passwords. <h3 class="panel-title">How Do I Prevent It?</h3> Password length Minimum passwords length should be at least eight (8) characters long. Combining this length with complexity makes a password difficult to guess and/or brute force. Password complexity Password characters should be a combination of alphanumeric characters. Alphanumeric characters consist of letters, numbers, punctuation marks, mathematical and other conventional symbols. Username /Password Enumeration Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code Additional Measures - For additional protection against brute forcing, enforce account disabling after an established number of invalid login attempts (e. g., five attempts is common). The account must be disabled for a period of time sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed. - Only send non-temporary passwords over an encrypted connection or as encrypted data, such as in an encrypted email. Temporary passwords associated with email resets may be an exception. Enforce the changing of temporary passwords on the next use. Temporary passwords and links should have a short expiration time. <h3 class="panel-title">Source Code Example</h3> The demo application doesn't enforce strong password. In routes/session.js <code>validateSignup()</code>method, the regex for password enforcement is simply <pre>var PASS_RE = /^.{1,20}$/;</pre> </p> <p> A stronger password can be enforced using the regex below, which requires at least 8 character password with numbers and both lowercase and uppercase letters. <pre>var PASS_RE =/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}$/; </pre> Another issue, in routes/session.js, the <code>handleLoginRequest()< /code>enumerated whether password was incorrect or user doesn't exist.This information can be valuable to an attacker with brute forcing attempts. This can be easily fixed using a generic error message such as "Invalid username and/or password". < /div> <!-- /Password Complexity --> <!-- end accordions --> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a3
Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </ /span> </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> <div class="col-lg-

Evidence	12"> <h1>A3-Cross-Site Scripting (XSS) <small></small> </h1> </div> </div> <!-- /.row --> <div class="row"> <div class="col-lg-12"> <div class="bs-example" style="margin-bottom: 40px;"> Exploitability: AVERAGE Prevalence: VERY WIDESPREAD Detectability: EASY Technical Impact: MODERATE </div> </div> </div> <div class="row"> <div class="col-lg-12"> <div class="panel panel-info"> <div class="panel-heading"> <h3 class="panel-title">Description< /h3> </div> <div class="panel-body"> XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victims' browser, which can access any cookies, session tokens, or other sensitive information retained by the browser, or redirect user to malicious sites. </div> </div> <div class="panel panel-info"> <div class="panel-heading"> <h3 class="panel-title">Attack Mechanics</h3> </div> <div class="panel-body"> <p> There are two types of XSS flaws: </p> - Reflected XSS: The malicious data is echoed back by the server in an immediate response to an HTTP request from the victim. - Stored XSS: The malicious data is stored on the server or on browser (using HTML5 local storage, for example), and later gets embedded in HTML page provided to the victim.< /li> <p>Each of reflected and stored XSS can occur on the server or on the client (which is also known as DOM based XSS), depending on when the malicious data gets injected in HTML markup.</p> </div> </div> <div class="panel panel-info"> <div class=" panel-heading"> <h3 class="panel-title">How Do I Prevent It?</h3> </div> <div class=" panel-body"> <p> Input validation and sanitization: Input validation and data sanitization are the first line of defense against untrusted data. Apply white list validation wherever possible.</p> <p> <b> Output encoding for correct context: < /b>When a browser is rendering HTML and any other associated content like CSS, javascript etc., it follows different rendering rules for each context. Hence <i>Context-sensitive output encoding</i> is absolutely critical for mitigating risk of XSS.</p> Here are the details about applying correct encoding in each context: <table class="table table-bordered table-hover"> <tbody> <tr> <th>Context</th> <th>Code Sample</th> <th>Encoding Type</th> </tr> <tr> <td>HTML Entity</td> <td> UNTRUSTED DATA</td> <td>Convert & to & amp; Convert < to < Convert > to > Convert " to & quot; Convert ' to ' Convert / to / </td> </tr> <tr> <td>HTML Attribute Encoding</td> <td><input type="text" name="fname" value=" UNTRUSTED DATA"></td> <td>Except for alphanumeric characters, escape all characters with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value) </td> </tr> <tr> <td>URI Encoding</td> <td>clickme</td> <td>Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value) < /td> </tr> <tr> <td>JavaScript Encoding</td> <td><script>var currentValue=' UNTRUSTED DATA';</script> <script> someFunction(' UNTRUSTED DATA');</script> </td> <td>Ensure JavaScript variables are quoted. Except for alphanumeric characters, escape all characters with ASCII values less than 256 with \uXXXX unicode escaping format (X = Integer), or in xHH (HH = HEX Value) encoding format. </td> </tr> <tr> <td>CSS Encoding< /td> <td><div style="width: UNTRUSTED DATA;"> Selection</div></td> <td>Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the \HH (HH= Hex Value) escaping format. </td> </tr> </tbody> </table> <p>HTTPOnly cookie flag: Preventing all XSS flaws in an application is hard. To help mitigate the impact of an XSS flaw on your site, set the HTTPOnly flag on session cookie and any custom cookies that are not required to be accessed by JavaScript. </p> <p><b>Implement Content Security Policy (CSP):< /b> CSP is a browser side mechanism which allows creating whitelists for client side resources used by the web application, e.g. JavaScript, CSS, images, etc. CSP via special HTTP header instructs the browser to only execute or render resources from those sources. For example, the CSP header below allows content only from example site's own domain (mydomain.com) and all its sub domains. <pre>Content-Security-Policy: default-src 'self' *. mydomain.com</pre> </p> - <b>Apply encoding on both client and server side: < /b> It is essential to apply encoding on both client and server side to mitigate DOM based XSS attack, in which untrusted data never leaves the browser. <p>Source: XSS Prevention Cheat Sheet[1] </p> </div> </div> <div id="source-code-example" class="panel panel-info"> <div class="panel-heading"> <h3 class="panel-title">Source Code Example< /h3> </div> <div class="panel-body"> <p> The demo web application is vulnerable to stored XSS attack on profiles form. On form submit, the first and last name field values are submitted to the server, and without any validation get saved in database. The values are then sent back to the browser without proper escaping to be shown at the top right menu. < /p> <iframe width="560" height="315" src="//www.youtube.com/embed/KvZ5jdg083M?

	rel=0" frameborder="0" allowfullscreen></iframe> Two measures can be taken to mitigate XSS risk: - In <code>server.js</code>, enable the HTML Encoding using template engine's auto escape flag. <pre> swig.init({ root: __dirname + "/app/views", autoescape: true //default value }); </pre> - Set HTTPOnly flag for session cookie while configuring the express session <pre> // Enable session management using express middleware app.use(express.session({ secret: "s3Cur3", cookie: { httpOnly: true, secure: true } })); </pre> There were no additional contexts that needed encoding on the demo page; otherwise, it is necessary to encode for correct context depending on where data get placed at. <h3 class="panel-title">Output Encoding Context</h3> An important observation when handling output encoding to prevent XSS is the notion of context. When output encoding is performed, it must match the context in which it is being injected to. For example, if a user input is being injected to an HTML element then it will require different encoding semantics to escape malicious input than if it were injected to say an HTML attribute or a JavaScript context altogether (such as in a script tag). An example for how to take advantage and exploit this mis-understanding exists on the profile page. See code references in <code>profile.js</code> and <code>profile.html</code> <h3 class="panel-title">Further Reading</h3> XSS Prevention Cheat Sheet Types of Cross-Site Scripting XSS Filter Evasion Cheat Sheet < /li> Unraveling some of the Mysteries around DOM-based XSS <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a4
Method	GET
Attack	
Evidence	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </ /span> </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> A4-Insecure Direct Object References <!-- /. row --> Exploitability: EASY Prevalence: COMMON Detectability: EASY Technical Impact: MODERATE <h3 class="panel-title">Description< /h3> A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can

	manipulate these references to access unauthorized data. <h3 class="panel-title">Attack Mechanics</h3> If an applications uses the actual name or key of an object when generating web pages, and doesn't verify if the user is authorized for the target object, this can result in an insecure direct object reference flaw. An attacker can exploit such flaws by manipulating parameter values. Unless object references are unpredictable, it is easy for an attacker to access all available data of that type. For example, the insure demo application uses userid as part of the url to access the allocations (/allocations/{id}). An attacker can manipulate id value and access other user's allocation information. <iframe width="560" height="315" src="//www.youtube.com/embed/KFTRMw5F_eg?rel=0" frameborder="0" allowfullscreen></iframe> <h3 class="panel-title">How Do I Prevent It?</h3> - Check access: Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object. - Use per user or session indirect object references: Instead of exposing actual database keys as part of the access links, use temporary per-user indirect reference. For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 or unique random numbers to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server. - Testing and code analysis: Testers can easily manipulate parameter values to detect such flaws. In addition, code analysis can quickly show whether authorization is properly verified. <h3 class="panel-title">Source Code Example</h3> In <code>routes/allocations.js</code>, the insecure application takes user id from url to fetch the allocations. <pre> var userId = req.params. userId; allocationsDAO.getByUserId(userId, function(error, allocations) { if (error) return next (error); return res.render("allocations", allocations); }); </pre> A safer alternative is to always retrieve allocations for logged in user (using <code>req.session.userId</code>) instead of taking it from url. <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a5
Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /.navbar-collapse --> </nav> A5-Security Misconfiguration <!-- /.row --> Exploitability: EASY Prevalence: COMMON Detectability: EASY Technical Impact: MODERATE <h3 class="panel-title">Description<

Evidence	/h3> This vulnerability allows an attacker to accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system. Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly.< /p> <h3 class=" panel-title">Attack Mechanics</h3> This vulnerability encompasses a broad category of attacks, but here are some ways attacker can exploit it: - If application server is configured to run as root, an attacker can run malicious scripts (by exploiting eval family functions) or start new child processes on server - Read, write, delete files on file system. Create and run binary files - If the server is misconfigured to leak internal implementation details via cookie names or HTTP response headers, then attacker can use this information towards building site's risk profile and finding vulnerabilities - If request body size is not limited, an attacker can upload large size of input payload, causing server to run out of memory, or make processor and event loop busy. <h3 class="panel-title">How Do I Prevent It?</h3> Here are some node.js and express specific configuration measures: - Use latest stable version of node.js and express (or other web framework you are using). Keep a watch on published vulnerabilities of these. The vulnerabilities for node.js and express.js can be found here and here, respectively. - Do not run application with root privileges. It may seem necessary to run as root user to access privileged ports such as 80. However, this can achieved either by starting server as root and then downgrading the non-privileged user after listening on port 80 is established, or using a separate proxy, or using port mapping. - Review default in HTTP Response headers to prevent internal implementation disclosure. - Use generic session cookie names - Limit HTTP Request Body size by setting sensible size limits on each content type specific middleware (<code>urlencoded, json, multipart</code>) instead of using aggregate <code>limit</code>middleware. Include only required middleware. For example if application doesn't need to support file uploads, do not include multipart middleware. - If using multipart middleware, have a strategy to clean up temporary files generated by it. These files are not garbage collected by default, and an attacker can fill disk with such temporary files - Vet npm packages used by the application - Lock versions of all npm packages used, for example using shrinkwarp, to have full control over when to install a new version of the package. - Set security specific HTTP headers <h3 class="panel-title" >Source Code Example</h3> <iframe width="560" height="315" src="//www.youtube.com/embed/lCpnVrD2Neg?rel=0" frameborder="0" allowfullscreen></iframe> The default HTTP header x-powered-by can reveal implementation details to an attacker. It can be taken out by including this code in <code>server.js</code> <pre> app.disable("x-powered-by"); </pre> The default session cookie name for express sessions can be changed by setting key attribute while creating express session. <pre> app.use(express.session({ secret: config.cookieSecret, key: "sessionId", cookie: { httpOnly: true, secure: true } })); </pre> The security related HTTP Headers can be added using helmet middleware as below <pre> // Prevent opening page in frame or iframe to protect from clickjacking app.disable("x-powered-by"); // Prevent opening page in frame or iframe to protect from clickjacking app.use(helmet. xframe()); // Prevents browser from caching and storing page app.use(helmet.noCache()); // Allow loading resources only from white-listed domains app.use(helmet.csp()); // Allow communication only on HTTPS app.use(helmet.hsts()); // Forces browser to only use the Content-Type set in the response header instead of sniffing or guessing it app.use(nosniff()); </pre> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a6
Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-

Evidence	ex1-collapse"> Toggle navigation </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> <h1>A6-Sensitive Data Exposure <small></small> </h1> <!-- /.row --> Exploitability: DIFFICULT Prevalence: COMMON Detectability: AVERAGE Technical Impact: SEVERE <h3 class="panel-title">Description</h3> < /div> This vulnerability allows an attacker to access sensitive data such as credit cards, tax IDs, authentication credentials, etc to conduct credit card fraud, identity theft, or other crimes. Losing such data can cause severe business impact and damage to the reputation. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser. < /div> <h3 class="panel-title" >Attack Mechanics</h3> If a site doesn't use SSL/TLS for all authenticated pages, an attacker can monitor network traffic (such as on open wireless network), and steals user's session cookie. Attacker can then replay this cookie and hijacks the user's session, accessing the user's private data. If an attacker gets access the application database, he or she can steal the sensitive information not encrypted, or encrypted with weak encryption algorithm <h3 class="panel-title">How Do I Prevent It?< /h3> - Use Secure HTTPS network protocol - Encrypt all sensitive data at rest and in transit - Don't store sensitive data unnecessarily. Discard it as soon as possible. - Ensure strong standard algorithms and strong keys are used, and proper key management is in place. - Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data. <h3 class="panel-title">Source Code Example</h3> 1.The insecure demo application uses HTTP connection to communicate with server. A secure HTTPS sever can be set using https module. This would need a private key and certificate. Here are source code examples from <code>/server.js</code> <pre> // Load keys for establishing secure HTTPS connection var fs = require("fs"); var https = require("https"); var path = require("path"); var httpsOptions = { key: fs.readFileSync(path. resolve(__dirname, "./app/cert/key.pem")), cert: fs.readFileSync(path.resolve(__dirname, ". /app/cert/cert.pem")) }; </pre> 2. Start secure HTTPS sever <pre> // Start secure HTTPS server https.createServer(httpsOptions, app).listen(config.port, function() { console. log("Express https server listening on port " + config.port); }); </pre> 3. The insecure demo application stores users personal sensitive information in plain text. To fix it, The <code>data/profile-dao.js</code>can be modified to use crypto module to encrypt and decrypt sensitive information as below: <pre> // Include crypto module var crypto = require ("crypto"); //Set keys config object var config = { cryptoKey: "a_secure_key_for_crypto_here", cryptoAlgo: "aes256", // or other secure encryption algo here iv: "" }; // Helper method create initialization vector // By default the initialization vector is not secure enough, so we create our own var createIV = function() { // create a random salt for the PBKDF2 function - 16 bytes is the minimum length according to NIST var salt = crypto.randomBytes(16); return crypto.pbkdf2Sync(config.cryptoKey, salt, 100000, 512, "sha512"); }; // Helper methods to encryt / decrypt var encrypt = function(toEncrypt) { config. iv = createIV(); var cipher = crypto.createCipheriv(config.cryptoAlgo, config.cryptoKey, config.iv); return cipher.update(toEncrypt, "utf8", "hex") + cipher.final("hex"); }; var decrypt = function(toDecrypt) { var decipher = crypto.createDecipheriv(config.cryptoAlgo, config. cryptoKey, config.iv); return decipher.update(toDecrypt, "hex", "utf8") + decipher.final

	("utf8"); }; // Encrypt values before saving in database user.ssn = encrypt(ssn); user.dob = encrypt(dob); // Decrypt values to show on view user.ssn = decrypt(user.ssn); user.dob = decrypt(user.dob); </pre> <!-- /#page-wrapper --> < /div> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a7
Method	GET
Attack	
Evidence	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </ /span> </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> <h1>A7-Missing Function Level Access Control <small></small> </h1> <! -- /.row --> Exploitability: EASY Prevalence: COMMON Detectability: AVERAGE Technical Impact: MODERATE <h3 class="panel-title" >Description</h3> Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. <h3 class="panel-title">Attack Mechanics</h3> If requests are not verified for access rights on server, attackers can forge requests in order to access functionality without proper authorization. <iframe width="560" height="315" src="//www. youtube.com/embed/ej6NCVd1Fo4?rel=0" frameborder="0" allowfullscreen></iframe> In the insecure demo application, this vulnerability exists in benefits module, which allows changing benefit start date for employees. The link to the benefits module is visible only to the admin user (user: admin, password: Admin_123). However, an attacker can access this module simply by logging in as any non-admin user and accessing benefits url directly. <h3 class="panel-title">How Do I Prevent It?</h3> Most web applications don’t display links and buttons to unauthorized functions, but this “presentation layer access control” doesn't actually provide protection. You must also implement checks in the controller or business logic. <h3 class="panel-title">Source Code Examples</h3> In vulnerable application, there is no authorization check for benefits related routes in <code>routes/index.js</code> <pre> // Benefits Page app.get("/benefits", isLoggedIn, benefitsHandler.displayBenefits); app.post("/benefits", isLoggedIn, benefitsHandler. updateBenefits); </pre> This can be fixed by adding a middleware to verify user's role:< /p> <pre> // Benefits Page app.get("/benefits", isLoggedIn, isAdmin, benefitsHandler. displayBenefits); app.post("/benefits", isLoggedIn, isAdmin, benefitsHandler. updateBenefits); </pre> To implement <code>isAdmin</code>middleware, check if isAdmin flag is set for the logged in user in database. For example, here is middleware

	function that can be added to <code>routes\session.js</code>: <pre> this. isAdminUserMiddleware = function(req, res, next) { if (req.session.userId) { userDAO. getUserById(req.session.userId, function(err, user) { if(user && user.isAdmin) { next(); } else { return res.redirect("/login"); } }); } else { console.log("redirecting to login"); return res. redirect("/login"); } }; </pre> It can be then made available in <code>routes/index.js< /code>router as: <pre> var SessionHandler = require("./session"); //Middleware to check if user has admin rights var isAdmin = sessionHandler.isAdminUserMiddleware; </pre> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src=".. /vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a8
Method	GET
Attack	
Evidence	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </ /span> </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> <h1>A8-Cross-Site Request Forgery (CSRF) <small></small> </h1> <!-- /.row --> Exploitability: AVERAGE Prevalence: COMMON Detectability: EASY Technical Impact: MODERATE <h3 class="panel-title">Description< /h3> A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests that the vulnerable application processes as legitimate requests from the victim. <!-- <h3 class="panel-title">Real World Attack Incident Examples</h3> Screencast here ... --> <h3 class="panel-title" >Attack Mechanics</h3> As browsers automatically send credentials like session cookies with HTTP requests to the server where cookies were received from, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones. For example, CSRF vulnerability can be exploited on profile form on the insecure demo application. <iframe width="560" height="315" src="//www.youtube.com/embed/vRDykS_2y3I?rel=0" frameborder="0" allowfullscreen></iframe> To exploit it: - An attacker would need to host a forged form like below on a malicious sever. <pre> <html lang="en"> <head></head> <body> <form method="POST" action="http://TARGET_APP_URL_HERE/profile" > <h1> You are about to win a brand new iPhone!</h1> <h2> Click on the win button to claim it...</h2> <input type="hidden" name="bankAcc" value="9999999" /> <input type="hidden" name="bankRouting" value="88888888"/> <input type=" submit" value="Win !!!"/> </form> </body> </html> </pre> Note: A sample app containing form for CSRF attack on NodeGoat app is available <a target="_blank"

	href="https://github.com/ckarande/nodegoat-csrf-attack">here. - Next, attacker would need to manage opening the form on logged in victim's browser and attract user to submit it. When user submits this form, it results in victim user's browser sending a malicious request to vulnerable server, causing CSRF attack. <h3 class="panel-title">How Do I Prevent It?</h3> Express csrf middleware provides a very effective way to deal with csrf attack. By default this middleware generates a token named "_csrf" which should be added to requests which mutate state (PUT, POST, DELETE), within a hidden form field, or query-string, or header fields. If using method-override middleware, it is very important that it is used before any middleware that needs to know the method of the request, including CSRF middleware. Otherwise an attacker can use non-state mutating methods (such as GET) to bypass the CSRF middleware checks, and use method override header to convert request to desired method. When form is submitted, the middleware checks for existence of token and validates it by matching to the generated token for the response-request pair. If tokens do not match, it rejects the request. Thus making it really hard for an attacker to exploit CSRF. <h3 class="panel-title" >Source Code Example</h3> The <code>server.js</ /code>includes the express CSRF middleware after session is initialized. Then creates a custom middleware to generate new token using <code>req.csrfToken();</code>and exposes it to view by setting it in <code>res.locals</code> <pre> //Enable Express csrf protection app.use(express.csrf()); app.use(function(req, res, next) { res.locals.csrftoken = req.csrfToken(); next(); }); </pre> Next, this token can be included in a hidden form field in <code>views/profile.html</code>as below. <pre> <input type="hidden" name="_csrf" value="{{ csrftoken } }"></pre> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/a9
Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </ /span> </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> <h1>A9-Using Components with Known Vulnerabilities <small></small> </h1> < /div> <!-- /.row --> Exploitability: AVERAGE< /span> Prevalence: WIDESPREAD Detectability: DIFFICULT Technical Impact: MODERATE <h3 class="panel-title">Description</h3> Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts. Using

Evidence	insecure npm packages can lead to this vulnerability. Some projects today help test and alert on insecure dependencies: npm audit is a vulnerability scanner built into the npm CLI (version 6 or later) Dependabot security updates can automatically make GitHub pull requests to update vulnerable dependencies Snyk.io is a Node.js CLI tool and Platform to scan and detect vulnerable packages The tools above make use of vulnerability lists, which can also be viewed directly or searched here: NPM Security Advisories GitHub Advisory Database Snyk Vulnerability DB There are some other tools that can detect and update outdated packages: npm outdated and yarn outdated are both command line ways to show possibly out of date dependencies Dependabot version updates can automatically make GitHub pull requests to update outdated dependencies David DM gets you an overview of your project dependencies, the version you use and the latest available, so you can quickly see what's drifting npm-check Check for outdated, incorrect, and unused dependencies <h3 class="panel-title">Attack Mechanics</h3> The npm packages are essential part of our node application. These packages could either accidentally or maliciously contain insecure code. Through insecure packages an attacker can: - Create and run scripts at different stages during installation or usage of the package. - Read, write, update, delete files on system - Write and execute binary files - Collect sensitive data send it remotely <h3 class="panel-title">How Do I Prevent It?</h3> These are few measures we can take to protect against malicious npm packages - Do not run application with root privileges - Prefer packages that include static code analysis. Check JSHint/JSLint the configuration to know what rules code abide by - Prefer packages that contain comprehensive unit tests and review tests for the functions our application uses - Review code for any unexpected file or database access - Research about how popular the package is, what other packages use it, if any other packages are written by the author, etc - Lock version of packages used< /li> - Watch Github repositories for notifications. This will inform us if any vulnerabilities are discovered in the package in future <h3 class="panel-title" >Insecure Dependencies Example</h3> <h3 class="panel-title">Description</h3> < /div> The demo web application is using a popular library called Marked which is a Markdown parser in JavaScript and provides an easy way to integrate markdown syntax for rich text to a website, replacing the need to build WYSIWYG editors. This library has reached almost millions of downloads a month, making it quite popular with also 11,000 < /strong> stars on GitHub at one point. <h3 class="panel-title">Attack Mechanics</h3> In this demo project we are using an insecure version of the Marked library that is vulnerable to XSS exploits. Scenario: A form on a page allows free text user input which is later parsed using the Marked library to markdown format and compiled in a dedicated view to show the rich text version. An attacker can exploit this form to insert malicious XSS strings which the Markdown library isn't filtering very well, resulting in an XSS attack. Try sending one of the following markdown syntax strings in the Memos section to exploit it and see which one succeeds: <code> <xmp> [Nice try](javascript:alert(1)) </xmp> </code> <code> <xmp> [Hi there](javascript:alert(1)) </xmp> </code> <code> <xmp> [I'm here!](javascript&#58this;alert(1)) </xmp> </code> < /div> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/redos
Method	GET

Attack	
Evidence	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!-- Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </ /span> </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /. navbar-collapse --> </nav> <h1>ReDoS Regular Expressions DoS <small></small> </h1> <!-- /.row --> <h3 class="panel-title">Description</h3> The Regular expression Denial of Service (ReDoS) is a Denial of Service attack, that exploits the fact that most Regular Expression implementations may reach extreme situations that cause them to work very slowly (exponentially related to input size). An attacker can then cause a program using a Regular Expression to enter these extreme situations and then hang for a very long time. <h3 class="panel-title">Attack Mechanics</h3> When untrusted data input is executed on a regex pattern, it may exploit vulnerable patterns into running long calculations to match for a given string. For Node.js this is extremely important due to the single-threaded event-loop architecture which means that the main Node.js process is blocked from serving any other requests. < /div> <h3 class="panel-title">How Do I Prevent It?</h3> - Avoid writing your own regular expressions - Use Node.js's validator.js package to validate expected data format instead of writing your own regular expressions - As a last resort of writing your own regex patterns you can utilize Node.js's safe-regex package which allows detecting if a regex is prone to catastrophic backtracking, and also allows to configure threshold for maximum repetitions. <h3 class="panel-title">Source Code Example</h3> Even simple regex patterns are vulnerable to ReDoS. The NodeGoat project uses the following source code to validate text format from the user based on a regex pattern: <pre> // Allow only numbers with a suffix of #, for example: 'XXXXXX#' var regexPattern = /([0-9]+)+\#/; var testComplyWithRequirements = regexPattern.test(bankRouting) </pre> If a long enough input is provided it will stall the Node.js process and render it useless (in the background the Node.js process will take 100% cpu until stopped or the regex yields a result (true or false)). Try to input the following string in the Bank Routing number in the Profile form: <pre> 9176261211761212112312312312 1 </pre> < /div> <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js" ></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
URL	http://localhost:4000/tutorial/ssrf
Method	GET
Attack	
	<script src="../vendor/html5shiv.js"><![endif]--> </head> <body> <!-- Sidebar --> <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!--

Evidence	Brand and toggle get grouped for better mobile display --> <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse"> Toggle navigation </button> OWASP Node Goat Tutorial: Fixing OWASP Top 10 <!-- Collect the nav links, forms, and other content for toggling --> <ul class="nav navbar-nav side-nav" > A1 Injection A2 Broken Auth A3 XSS A4 Insecure DOR A5 Misconfig A6 Sensitive Data A7 Access Controls A8 CSRF A9 Insecure Components A10 Redirects ReDoS Attacks SSRF <ul class="nav navbar-nav navbar-right navbar-user"> Exit <!-- /.navbar-collapse --> </nav> <h1>Server-Side Request Forgery (SSRF) <small></small> </h1> <!-- /.row --> <h3 class="panel-title">Description</h3> In an SSRF attack, the attacker can abuse functionality on the server to read or update internal resources. The attacker can supply or modify a URL that the code running on the server will read or submit data to, and by carefully selecting the URLs, the attacker may be able to read server configuration such as AWS metadata, connect to internal services like HTTP-enabled databases or perform HTTP POST requests towards internal services which are not intended to be exposed. <h3 class="panel-title">Attack Mechanics</h3> An attacker can use an SSRF vulnerability as a way to gather information about the server and the local network. For example, on the "Research" page (<code>/research</code>) in the application, a user submits a stock symbol. The stock symbol is concatenated to a Yahoo URL and the server fetches the response and displays the page. <iframe width="560" height="315" src="https://www.youtube.com/embed/neClYWB05bQ" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe> Here is a code snippet from <code>routes/research.js</code>, <pre> // If a stock symbol has been submitted, concatenate the symbol to the URL and return the HTTP Response if (req.query.symbol) { var url = req.query.url+req.query.symbol; needle.get(url, function(error, newResponse) { ... } </pre> An attacker can change the <code>url</code> and <code>symbol</code> parameters to point to an attacker-controlled website to interact with the server. <h3 class="panel-title" >How Do I Prevent It?</h3> To prevent SSRF vulnerabilities in web applications, it is recommended to adhere to the following guidelines:< /p> - Use a whitelist of allowed domains, resources and protocols from where the web server can fetch resources. - Any input accepted from the user should be validated and rejected if it does not match the positive specification expected. - If possible, do not accept user input in functions that control where the web server can fetch resources. <!-- /#page-wrapper --> <!-- /#wrapper --> <script src="../vendor/jquery.min.js"></script>
Other Info	No links have been found while there are scripts, which is an indication that this is a modern web application.
Instances	15
Solution	This is an informational alert and so no changes are required.
Reference	
CWE Id	
WASC Id	
Plugin Id	10109

Informational	**Session Management Response Identified**
	The given response has been identified as containing a session management token. The 'Other Info' field contains a set of header tokens that can be used in the Header Based

Description	Session Management Method. If the request is in a context which has a Session Management Method set to "Auto-Detect" then this rule will change the session management to use the tokens identified.
URL	http://localhost:4000
Method	GET
Attack	
Evidence	s%3AMicBPC17GhRQFQnNCTqWZngimk6vEk5p.2XjhZ%2FWw6usyI%2FRtIbY1r3%2Ffc%2FH%2FgDjz0Jai8SvrTtI
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3AChapHjyY_4P13yIAG4Ryw6dCYk2sKnR5.pWU%2BIR4wUPQDJ4Oy9rupURUixN%2FGpCeOzWn%2BtoUNC5w
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3ADmNWefboeq9zjcUzhqYPtkP3hSTYJr1o.JqE6Cv%2Fj8mY7Ki8UoZilTq45LVSWJvSYjua9gS8Ww2c
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3AHJo1wltnAGXw898_9IZGMUxBOVo5CXUL.gpvpflGRnTS3DfrS99zzWF2HIfZuxAQ78hnttYllj6g
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3APHM6vjwVoLHE8-Dn40W-ujlrMy04f4c1.e7%2B%2BF2HjXvpDVnFcK40m3yp5P%2FBtWNfbLVpdINDgZm4
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3AeXxXzX44N3bssn13hGWrFZZziWHbA1hJ.Pzs3SBqgwI2N56%2FbbT5vwsUg5K%2Fj%2FeJj9JkI80jns%2Bc
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET

Attack	
Evidence	s%3AhEFTiP8eKVu5XkMh4ygoU_8kUqXd9yYT.%2B0eQpFUfxmB%2FwVgo%2Bhj%2FC%2F13L8%2Fh0rfhJ7AlCAgdo%2Fw
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3ApTlcANscygq1iw1BrYJbqzporhDaOPwM.jHJ7nvFToKPJGTNPkP3008ld%2BL4%2BcA7uJoARLzH4pak
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3AzI_gWNqOgroPKdZH435kiTMAKat87eoF.7jXPbaw01BaGSC8ZiKcspj1fRByi6mGrZ24lmvDhpxQ
Other Info	cookie:connect.sid
URL	http://localhost:4000/robots.txt
Method	GET
Attack	
Evidence	s%3Az3QH-mcwuvocuI5W5W563tndYNIUV1sh.MGYDcE3fquG0rUkAz1%2F5ThLMWVthmqYe8Hd7JXja23Y
Other Info	cookie:connect.sid
URL	http://localhost:4000/sitemap.xml
Method	GET
Attack	
Evidence	s%3AAuuCORYfn2miRGK3xVtTTh0GzNhUm7zE.FtiUF76UAJg5ydpTzrAmZpbqAe%2BRsm3fmgvYhBS4Lu4
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3AChapHjyY_4P13yIAG4Ryw6dCYk2sKnR5.pWU%2BIR4wUPQDJ4Oy9rupURUixN%2FGpCeOzWn%2BtoUNC5w
Other Info	cookie:connect.sid
URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3AHJo1wltnAGXw898_9IZGMUxBOVo5CXUL.gpvpflGRnTS3DfrS99zzWF2HIfZuxAQ78hnttYllj6g
Other Info	cookie:connect.sid

URL	http://localhost:4000/
Method	GET
Attack	
Evidence	s%3ApTlcANscygq1iw1BrYJbqzporhDaOPwM.jHJ7nvFToKPJGTNPkP3008ld%2BL4%2BcA7uJoARLzH4pak
Other Info	cookie:connect.sid
Instances	14
Solution	This is an informational alert rather than a vulnerability and so there is nothing to fix.
Reference	https://www.zaproxy.org/docs/desktop/addons/authentication-helper/session-mgmt-id
CWE Id	
WASC Id	
Plugin Id	10112

Informational	**User Agent Fuzzer**
Description	Check for differences in response based on fuzzed User Agent (eg. mobile sites, access as a Search Engine Crawler). Compares the response statuscode and the hashcode of the response body with the original response.
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0

Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/benefits
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)

Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	

Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/images
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	

Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other	

Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/learn?url=...
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	

URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/research
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other	

Info	
URL	http://localhost:4000/research
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	

URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/vendor
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	

URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap

Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/vendor/bootstrap
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme

Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme

Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET

Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET

Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)

Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/css
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)

Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12A366 Safari/600.1.4

Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
Evidence	
Other Info	
URL	http://localhost:4000/vendor/theme/font-awesome/fonts
Method	GET
Attack	msnbot/1.1 (+http://search.msn.com/msnbot.htm)
Evidence	
Other Info	
Instances	120
Solution	
Reference	https://owasp.org/wstg
CWE Id	
WASC Id	
Plugin Id	10104

Informational	**User Controllable HTML Element Attribute (Potential XSS)**
Description	This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	
Other Info	User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL: http://localhost:4000/signup appears to include user input in: a(n) [input] tag [value] attribute The user input found was: userName=SMCEpAKb The user-controlled value was: smcepakb
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	
Other Info	User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL: http://localhost:4000/signup appears to include user input in: a(n) [input] tag [value] attribute The user input found was: userName=SMCEpAKbdzbdqQlM The user-controlled value was: smcepakbdzbdqqlm
URL	http://localhost:4000/signup
Method	POST
Attack	

Evidence	
Other Info	User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL: http://localhost:4000/signup appears to include user input in: a(n) [input] tag [value] attribute The user input found was: userName=SMCEpAKbdzbdqQlMVvaUZhHFGdtaxWgL The user-controlled value was: smcepakbdzbdqqlmvvauzhhfgdtaxwgl
URL	http://localhost:4000/signup
Method	POST
Attack	
Evidence	
Other Info	User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL: http://localhost:4000/signup appears to include user input in: a(n) [input] tag [value] attribute The user input found was: userName=SMCEpAKbdzbdqQlMXbwJJkrH The user-controlled value was: smcepakbdzbdqqlmxbwjjkrh
Instances	4
Solution	Validate all input and sanitize output it before writing to any HTML attributes.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
CWE Id	20
WASC Id	20
Plugin Id	10031