ZAP Scanning Report

Summary of Alerts

Generated on Sat, 14 May 2022 13:59:31

Risk Level	Number of Alerts
High	1
Medium	3
Low	3
Informational	0

Alerts

Name	Risk Level	Number of Instances
Remote OS Command Injection	High	1
Cross-Domain Misconfiguration	Medium	5
CSP: Wildcard Directive	Medium	2
X-Frame-Options Header Not Set	Medium	2
Absence of Anti-CSRF Tokens	Low	2
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	5
X-Content-Type-Options Header Missing	Low	2

Alert Detail

High (Medium)	**Remote OS Command Injection**
Description	Attack technique used for unauthorized execution of operating system commands. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs.
URL	http://10.10.10.55/api/config
Method	POST
Parameter	interval
Attack	1;&sleep 15&
Instances	1
Solution	If at all possible, use library calls rather than external processes to recreate the desired functionality. Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise. For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the command locally in the session's state instead of sending it out to the client in a hidden form field. Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using the ESAPI Encoding control or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. If you need to use dynamically-generated query strings or commands in spite of the risk, properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allow list (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection. If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line. 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. Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments. 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." When constructing OS command strings, use stringent allow lists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping. Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components. Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.
Reference	http://cwe.mitre.org/data/definitions/78.html https://owasp.org/www-community/attacks/Command_Injection
CWE Id	78
WASC Id	31
Source ID	1

Medium (Medium)	**Cross-Domain Misconfiguration**
Description	Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server
URL	http://10.10.10.55
Method	GET
Evidence	Access-Control-Allow-Origin: *
URL	http://10.10.10.55/sitemap.xml
Method	GET
Evidence	Access-Control-Allow-Origin: *
URL	http://10.10.10.55/
Method	GET
Evidence	Access-Control-Allow-Origin: *
URL	http://10.10.10.55/api/config
Method	POST
Evidence	Access-Control-Allow-Origin: *
URL	http://10.10.10.55/robots.txt
Method	GET
Evidence	Access-Control-Allow-Origin: *
Instances	5
Solution	Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.
Other information	The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.
Reference	http://www.hpenterprisesecurity.com/vulncat/en/vulncat/vb/html5_overly_permissive_cors_policy.html
CWE Id	264
WASC Id	14
Source ID	3

Medium (Medium)	**CSP: Wildcard Directive**
Description	The following directives either allow wildcard sources (or ancestors), are not defined, or are overly broadly defined: frame-ancestors, form-action The directive(s): frame-ancestors, form-action are among the directives that do not fallback to default-src, missing/excluding them is the same as allowing anything.
URL	http://10.10.10.55/robots.txt
Method	GET
Evidence	default-src 'none'
URL	http://10.10.10.55/sitemap.xml
Method	GET
Evidence	default-src 'none'
Instances	2
Solution	Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.
Reference	http://www.w3.org/TR/CSP2/ http://www.w3.org/TR/CSP/ http://caniuse.com/#search=content+security+policy http://content-security-policy.com/ https://github.com/shapesecurity/salvation https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources
CWE Id	693
WASC Id	15
Source ID	3

Medium (Medium)	**X-Frame-Options Header Not Set**
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://10.10.10.55
Method	GET
Parameter	X-Frame-Options
URL	http://10.10.10.55/
Method	GET
Parameter	X-Frame-Options
Instances	2
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (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
Source ID	3

Low (Medium)	**Absence of Anti-CSRF Tokens**
Description	No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf. CSRF attacks are effective in a number of situations, including: * The victim has an active session on the target site. * The victim is authenticated via HTTP auth on the target site. * The victim is on the same local network as the target site. CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.
URL	http://10.10.10.55
Method	GET
Evidence	<form action="/api/config" method="POST">
URL	http://10.10.10.55/
Method	GET
Evidence	<form action="/api/config" method="POST">
Instances	2
Solution	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. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Other information	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF] was found in the following HTML form: [Form 1: "update-interval"].
Reference	http://projects.webappsec.org/Cross-Site-Request-Forgery http://cwe.mitre.org/data/definitions/352.html
CWE Id	352
WASC Id	9
Source ID	3

Low (Medium)	**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://10.10.10.55/sitemap.xml
Method	GET
Evidence	X-Powered-By: Express
URL	http://10.10.10.55/robots.txt
Method	GET
Evidence	X-Powered-By: Express
URL	http://10.10.10.55
Method	GET
Evidence	X-Powered-By: Express
URL	http://10.10.10.55/
Method	GET
Evidence	X-Powered-By: Express
URL	http://10.10.10.55/api/config
Method	POST
Evidence	X-Powered-By: Express
Instances	5
Solution	Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.
Reference	http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html
CWE Id	200
WASC Id	13
Source ID	3

Low (Medium)	**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://10.10.10.55/
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.10.10.55
Method	GET
Parameter	X-Content-Type-Options
Instances	2
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.
Other information	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.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://owasp.org/www-community/Security_Headers
CWE Id	693
WASC Id	15
Source ID	3