Veracode Detailed Report

Application Security Report
As of 18 Jun 2020

Prepared for:	Exela Technologies
Prepared on:	June 19, 2020
Application:	Data Room
Industry:	Not Specified
Business Criticality:	BC5 (Very High)
Required Analysis:	Static, Manual Penetration Test
Type(s) of Analysis Conducted:	Static, Dynamic
Scope of Static Scan:	2 of 2 Modules Analyzed
Scope of Dynamic Scan:	https://dataroom.exela.global/#/login

Inside This Report

65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com

Veracode Detailed Report

Application Security Report As of 18 Jun 2020

Veracode Level: VL2
Rated: Jun 18, 2020

Application:	Data Room	Business Criticality:	Very High
Target Level:	VL5	Published Rating:	CA

Scans Included in Report

Static Scan	Dynamic Scan	Manual Penetration Test
18 Jun 2020 Static Score: 77 Completed: 6/18/20	Dataroom (1) Score: 99 Completed: 12/12/19	Not Included in Report

Executive Summary

This report contains a summary of the security flaws identified in the application using manual penetration testing, automated static and/or automated dynamic security analysis techniques. This is useful for understanding the overall security quality of an individual application or for comparisons between applications.

Application Business Criticality: BC5 (Very High)

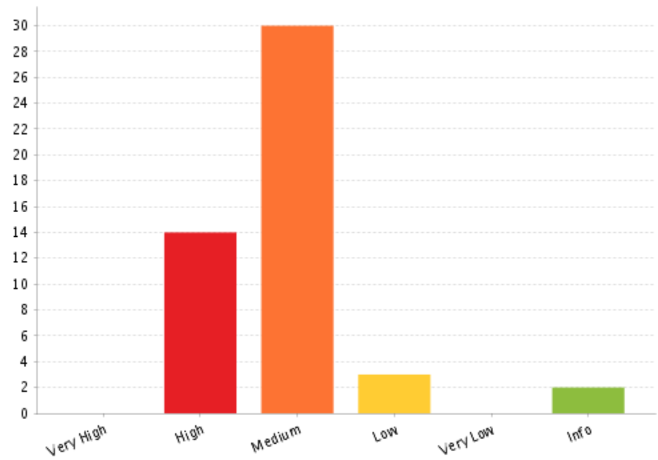
Impacts:Operational Risk (High), Financial Loss (High)

An application's business criticality is determined by business risk factors such as: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations. The Veracode Level and required assessment techniques are selected based on the policy assigned to the application.

Analyses Performed vs. Required

	Any	Static	Dynamic	Manual Penetration Test
Performed:		●	●	○
Required:	○	●	○	●

Summary of Flaws Found by Severity



Action Items:

Veracode recommends the following approaches ranging from the most basic to the strong security measures that a vendor can undertake to increase the overall security level of the application.

Required Analysis

→ Your policy requires Manual Penetration Test but it has not been performed. Please submit your application for Manual Penetration Test and remediate the required detected flaws to conform to your assigned policy.

→ Your policy requires periodic Static Scan. Your next analysis must be completed by 9/18/20. Please submit your application for Static Scan by the deadline and remediate the required detected flaws to conform to your assigned policy.

Flaws To Fix By Expires Date

A grace period is specified for any flaw that violates the rules contained in your policy. These include CWE, Rollup Category, Issue Severity, Industry Standards as well as any flaws that prevent an application from achieving a minimum Veracode Level and/or score. To maintain policy compliance you must fix these flaws and resubmit your application for scanning before the grace period expires. The detailed flaw listing will badge the flaws that must be fixed and show the fix by date as well.

- The grace period has expired [12/19/19] for 1 flaw that was found in your Static Scan.
- The grace period has expired [1/9/20] for 2 flaws that were found in your Static Scan.
- The grace period has expired [1/20/20] for 1 flaw that was found in your Static Scan.
- The grace period has expired [6/18/20] for 40 flaws that were found in your Static Scan.

Flaws To Fix For Minimum Score

- A rule in your policy requires a minimum score of 90. Fix 14 High flaws and 1 Medium flaw to reach a score of 90.
- Your Static Scan was due on 6/18/20 for follow-up analysis to satisfy the grace period on your minimum score rule and your application is no longer compliant with your policy. Submit application for follow-up Static Scan once flaws have been remediated in order to regain compliance with your policy.

Flaw Severities

- Medium severity flaws and above must be fixed for policy compliance.

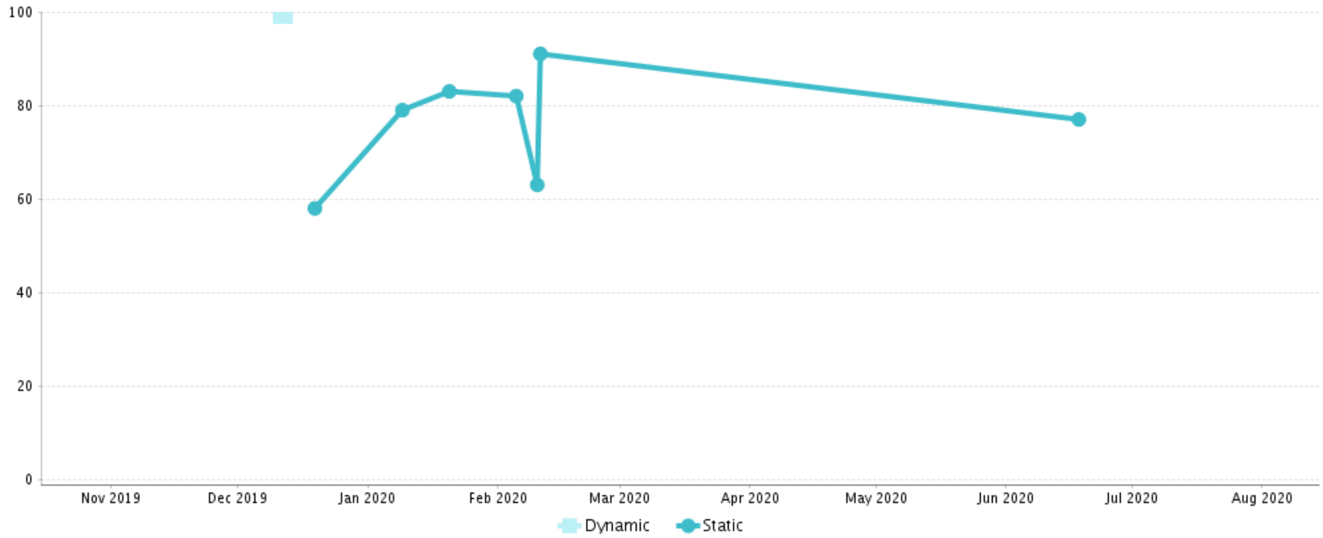
Longer Timeframe (6 - 12 months)

- Certify that software engineers have been trained on application security principles and practices.

65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com

Application Trend Data



Scope of Static Scan

The following modules were included in the static scan because the scan submitter selected them as entry points, which are modules that accept external data.

Engine Version: 20200520200321

The following modules were included in the application scan:

Module Name	Compiler	Operating Environment	Engine Version
JS files within dataroomscan.zip	JAVASCRIPT_5_1	JavaScript	20200520200321
Python files within dataroomscan.zip	Python		20200505194254

File Differences Between Scans
The uploaded modules for this scan do not match the modules you uploaded for the previous scan. This disparity can affect the scan results even if Veracode did not find flaws in the files with differences. See appendix for more details.

Scope of Dynamic Scan

These are the parameters that were used to perform the application scan:

Setting	Value
Target URL	https://dataroom.exela.global/#/login
Max Links to Crawl	5000
Number of Links Visited	49
Successful Logins	4 of 14
Login User ID	test.account
User Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0
Scan Duration	26 Minutes
Scan Window	12/11/2019 @ 11:56 PM EST - 12/12/2019 @ 11:52 PM EST

It is important to note that this application may include additional directories or URLs which were not included in this analysis. We recommend that you contact the vendor to determine whether all relevant URLs have been included.

Allowed Hosts	Directory Restrictions
https://dataroom.exela.global/	Directory and Subdirectories

Flaw Types by Severity and Category

	Static Scan Security Quality Score = 77 (-14) from prior scan		Dynamic Scan Security Quality Score = 99 from prior scan		
Very High	**0**		**0**		
High	**14**	**(+12)**	**0**		
Authorization Issues	5	(+3)			
SQL Injection	9	(+9)			
Medium	**30**	**(+12)**	**0**	**(-1)**	
CRLF Injection	9	(+8)			
Credentials Management		(-10)			
Cross-Site Scripting (XSS)	7	(+2)			
Cryptographic Issues	4	(+4)			
Directory Traversal	3	(+1)			
Encapsulation				(-1)	
Insufficient Input Validation	7	(+7)			
Low	**1**	**(+1)**	**2**	**(+2)**	
Information Leakage	1	(+1)	2	(+2)	
Very Low	**0**		**0**		
Informational	**0**		**2**	**(+2)**	
Server Configuration			2	(+2)	
Total	**45**	**(+25)**	**4**	**(+3)**	

Policy Evaluation

Policy Name: Veracode Recommended Very High

Revision: 1

Policy Status: Did Not Pass

Description

Veracode provides default policies to make it easier for organizations to begin measuring their applications against policies. Veracode Recommended Policies are available for customers as an option when they are ready to move beyond the initial bar set by the Veracode Transitional Policies. The policies are based on the Veracode Level definitions.

Rules

Rule type	Requirement	Findings	Status
Minimum Veracode Level	VL5	VL2	Did not pass
(VL5) Min Analysis Score	90	77	Did not pass
(VL5) Max Severity	Medium	Flaws found: 44	Did not pass

Scan Requirements

Scan Type	Frequency	Last performed	Status
Static	Quarterly	6/18/20	Passed
Manual	Annually	Never	Need to complete (before 12/11/20)

Remediation

Flaw Severity	Grace Period	Flaws Exceeding	Status
Very High	0 days	0	Passed
High	0 days	14	Did not pass
Medium	0 days	30	Did not pass
Low	0 days	0	Passed
Very Low	0 days	0	Passed
Informational	0 days	0	Passed

Type	Grace Period	Exceeding	Status
Min Analysis Score	0 days	1	Did not pass

Findings & Recommendations

Detailed Flaws by Severity

Very High (0 flaws)

No flaws of this type were found

High (14 flaws)

Fix Required by Policy

Authorization Issues(5 flaws)

Description

Authorization is the process or method by which an application determines whether a user, service, or application has the necessary permissions to perform a requested action. Web applications often restrict access to specific content or functionality based on the user's role or privilege level. If authorization is not implemented properly, an attacker can manipulate a web site to gain access to data or functionality that should be protected.

Authorization should not be confused with authentication. Authentication is the process of verifying a user's identity, while authorization enforces what that user is permitted to do after they have successfully authenticated to the system.

Recommendations

Be sure that authorization is properly enforced at the server side for every action. Centralize authorization routines when possible. Follow the principle of least privilege when designing security controls.

Associated Flaws by CWE ID:

Authorization Bypass Through User-Controlled Key (CWE ID 639)(5 flaws)

Description

The system's access control functionality does not prevent one user from gaining access to another user's records by modifying the key value identifying the record. Retrieval of a user record occurs in the system based on some key value that is under user control. The key would typically identify a user related record stored in the system and would be used to lookup that record for presentation to the user. It is likely that an attacker would have to be an authenticated user in the system. However, the authorization process would not properly check the data access operation to ensure that the authenticated user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other authorization checks present in the system. One manifestation of this weakness would be if a system used sequential or otherwise easily guessable session ids that would allow one user to easily switch to another user's session and view/modify their data.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	692	2	-	JS files within dataroomscan.zip	.../dataroom-create-group.component.ts 187	6/18/20
NEW	700	3	-	JS files within dataroomscan.zip	.../dataroom-folders.component.ts 264	6/18/20
	432	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1628	12/19/19

		Flaw Id	Module #	Class #	Module	Location	Fix By
		537	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1755	1/20/20
NEW		696	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 3880	6/18/20

SQL Injection(9 flaws)

Description

SQL injection vulnerabilities occur when data enters an application from an untrusted source and is used to dynamically construct a SQL query. This allows an attacker to manipulate database queries in order to access, modify, or delete arbitrary data. Depending on the platform, database type, and configuration, it may also be possible to execute administrative operations on the database, access the filesystem, or execute arbitrary system commands. SQL injection attacks can also be used to subvert authentication and authorization schemes, which would enable an attacker to gain privileged access to restricted portions of the application.

Recommendations

Several techniques can be used to prevent SQL injection attacks. These techniques complement each other and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a SQL injection vulnerability.

* Use parameterized prepared statements rather than dynamically constructing SQL queries. This will prevent the database from interpreting the contents of bind variables as part of the query and is the most effective defense against SQL injection.
* Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible.
* Normalize all user-supplied data before applying filters or regular expressions, or submitting the data to a database. This means that all URL-encoded (%xx), HTML-encoded (&#xx;), or other encoding schemes should be reduced to the internal character representation expected by the application. This prevents attackers from using alternate encoding schemes to bypass filters.
* When using database abstraction libraries such as Hibernate, do not assume that all methods exposed by the API will automatically prevent SQL injection attacks. Most libraries contain methods that pass arbitrary queries to the database in an unsafe manner.

Associated Flaws by CWE ID:

Improper Neutralization of Special Elements in Data Query Logic (CWE ID 943)(9 flaws)

Description

The application generates a query intended to access or manipulate data in a data store such as a database, but it does not neutralize or incorrectly neutralizes special elements that can modify the intended logic of the query.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Instances found via Static Scan

		Flaw Id	Module #	Class #	Module	Location	Fix By
NEW		706	1	-	JS files within dataroomscan.zip	/dataroomscan/.../carousel.js 1298	6/18/20

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	702	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 632	6/18/20
NEW	720	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1167	6/18/20
NEW	685	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1572	6/18/20
NEW	711	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 2199	6/18/20
NEW	689	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 2946	6/18/20
NEW	708	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 3187	6/18/20
NEW	697	21	-	JS files within dataroomscan.zip	.../templateProfileController.js 79	6/18/20
NEW	705	21	-	JS files within dataroomscan.zip	.../templateProfileController.js 159	6/18/20

Medium (30 flaws)

Fix Required by Policy

CRLF Injection(9 flaws)

Description

The acronym CRLF stands for "Carriage Return, Line Feed" and refers to the sequence of characters used to denote the end of a line of text. CRLF injection vulnerabilities occur when data enters an application from an untrusted source and is not properly validated before being used. For example, if an attacker is able to inject a CRLF into a log file, he could append falsified log entries, thereby misleading administrators or cover traces of the attack. If an attacker is able to inject CRLFs into an HTTP response header, he can use this ability to carry out other attacks such as cache poisoning. CRLF vulnerabilities primarily affect data integrity.

Recommendations

Apply robust input filtering for all user-supplied data, using centralized data validation routines when possible. Use output filters to sanitize all output derived from user-supplied input, replacing non-alphanumeric characters with their HTML entity equivalents.

Associated Flaws by CWE ID:

Improper Output Neutralization for Logs (CWE ID 117)(9 flaws)

Description

A function call could result in a log forging attack. Writing untrusted data into a log file allows an attacker to forge log entries or inject malicious content into log files. Corrupted log files can be used to cover an attacker's tracks or as a delivery mechanism for an attack on a log viewing or processing utility. For example, if a web administrator uses a browser-based utility to review logs, a cross-site scripting attack might be possible.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Avoid directly embedding user input in log files when possible. Sanitize untrusted data used to construct log entries by using a safe logging mechanism such as the OWASP ESAPI Logger, which will automatically remove unexpected carriage returns and line feeds and can be configured to use HTML entity encoding for non-alphanumeric data. Only write custom blacklisting code when absolutely necessary. Always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	715	2	-	JS files within dataroomscan.zip	.../dataroom-create-group.component.ts 364	6/18/20
NEW	719	3	-	JS files within dataroomscan.zip	.../dataroom-folders.component.ts 131	6/18/20
NEW	712	4	-	JS files within dataroomscan.zip	.../dataroom-team-members.component.ts 90	6/18/20
NEW	698	6	-	JS files within dataroomscan.zip	.../folder-management-list.component.ts 1042	6/18/20
NEW	714	9	-	JS files within dataroomscan.zip	.../group-permission.component.ts 512	6/18/20
NEW	717	10	-	JS files within dataroomscan.zip	/.../header/header.component.ts 119	6/18/20
NEW	687	18	-	JS files within dataroomscan.zip	/.../ng-pagination/pagination.js 393	6/18/20
NEW	691	23	-	JS files within dataroomscan.zip	.../upload-template.component.ts 73	6/18/20
NEW	701	23	-	JS files within dataroomscan.zip	.../upload-template.component.ts 78	6/18/20

→ Cross-Site Scripting (XSS)(7 flaws)

Description

Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed occur whenever a web application uses untrusted data in the output it generates without validating or encoding it. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise sensitive information, with new attack vectors being discovered on a regular basis. XSS is also commonly referred to as HTML injection.

XSS vulnerabilities can be either persistent or transient (often referred to as stored and reflected, respectively). In a persistent XSS vulnerability, the injected code is stored by the application, for example within a blog comment or message board. The attack occurs whenever a victim views the page containing the malicious script. In a transient XSS vulnerability, the injected code is included directly in the HTTP request. These attacks are often carried out via malicious URLs sent via email or another website and requires the victim to browse to that link. The consequence of an XSS attack to a victim is the same regardless of whether it is persistent or transient; however, persistent XSS vulnerabilities are likely to affect a greater number of victims due to its delivery mechanism.

Recommendations

Several techniques can be used to prevent XSS attacks. These techniques complement each other and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a XSS vulnerability.

* Use output filtering to sanitize all output generated from user-supplied input, selecting the appropriate method of encoding based on the use case of the untrusted data. For example, if the data is being written to the body of an HTML page, use HTML entity encoding. However, if the data is being used to construct generated Javascript or if it is consumed by client-side methods that may interpret it as code (a common technique in Web 2.0 applications), additional restrictions may be necessary beyond simple HTML encoding.
* Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible.
* Do not permit users to include HTML content in posts, notes, or other data that will be displayed by the application. If users are permitted to include HTML tags, then carefully limit access to specific elements or attributes, and use strict validation filters to prevent abuse.

Associated Flaws by CWE ID:

→ Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (CWE ID 80)(7 flaws)

Description

This call contains a cross-site scripting (XSS) flaw. The application populates the HTTP response with untrusted input, allowing an attacker to embed malicious content, such as Javascript code, which will be executed in the context of the victim's browser. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise confidential information, with new attack vectors being discovered on a regular basis.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Use contextual escaping on all untrusted data before using it to construct any portion of an HTTP response. The escaping method should be chosen based on the specific use case of the untrusted data, otherwise it may not protect fully against the attack. For example, if the data is being written to the body of an HTML page, use HTML entity escaping; if the data is being written to an attribute, use attribute escaping; etc. When a web framework provides built-in support for automatic XSS escaping, do not disable it. Both the OWASP Java Encoder library for Java and the Microsoft AntiXSS library provide contextual escaping methods. For more details on contextual escaping, see https://www.owasp.org/index.php/XSS_%%28Cross_Site_Scripting%%29_Prevention_Cheat_Sheet. In addition, as a best practice, always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	716	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 771	6/18/20
NEW	710	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 3274	6/18/20
NEW	707	11	-	JS files within dataroomscan.zip	/.../providers/http.service.ts 162	6/18/20
	368	17	-	JS files within dataroomscan.zip	.../ng-pagination.component.ts 55	6/18/20
	306	19	-	JS files within dataroomscan.zip	/.../pdf-editor.component.ts 281	6/18/20
	152	20	-	JS files within dataroomscan.zip	/.../snackbar.component.ts 23	6/18/20
NEW	721	22	-	JS files within dataroomscan.zip	.../text-view.component.html 4	6/18/20

Cryptographic Issues(4 flaws)

Description

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Common cryptographic mistakes include, but are not limited to, selecting weak keys or weak cipher modes, unintentionally exposing sensitive cryptographic data, using predictable entropy sources, and mismanaging or hard-coding keys.

Developers often make the dangerous assumption that they can improve security by designing their own cryptographic algorithm; however, one of the basic tenets of cryptography is that any cipher whose effectiveness is reliant on the secrecy of the algorithm is fundamentally flawed.

Recommendations

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

Associated Flaws by CWE ID:

Cleartext Storage of Sensitive Information (CWE ID 312)(4 flaws)

Description

The application stores sensitive information in plaintext within a resource that might be accessible to another control sphere, when the information should be encrypted or otherwise protected.

Effort to Fix: 4 - Simple design error. Requires redesign and up to 5 days to fix.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	709	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1578	6/18/20
NEW	690	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1876	6/18/20
NEW	694	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 2205	6/18/20
NEW	703	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 2544	6/18/20

Directory Traversal(3 flaws)

Description

Allowing user input to control paths used in filesystem operations may enable an attacker to access or modify otherwise protected system resources that would normally be inaccessible to end users. In some cases, the user-provided input may be passed directly to the filesystem operation, or it may be concatenated to one or more fixed strings to construct a fully-qualified path.

When an application improperly cleanses special character sequences in user-supplied filenames, a path traversal (or directory traversal) vulnerability may occur. For example, an attacker could specify a filename such as "../../etc/passwd", which resolves to a file outside of the intended directory that the attacker would not normally be authorized to view.

Recommendations

Assume all user-supplied input is malicious. Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters and ensure that the end result is not dangerous.

Associated Flaws by CWE ID:

External Control of File Name or Path (CWE ID 73)(3 flaws)

Description

This call contains a path manipulation flaw. The argument to the function is a filename constructed using untrusted input. If an attacker is allowed to specify all or part of the filename, it may be possible to gain unauthorized access to files on the server, including those outside the webroot, that would be normally be inaccessible to end users. The level of exposure depends on the effectiveness of input validation routines, if any.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Validate all untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters.

Instances found via Static Scan

		Flaw Id	Module #	Class #	Module	Location	Fix By
		505	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1401	1/9/20
NEW		693	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 1437	6/18/20
		503	15	-	JS files within dataroomscan.zip	/.../listViewController.js 199	1/9/20

Insufficient Input Validation(7 flaws)

Description

Weaknesses in this category are related to an absent or incorrect protection mechanism that fails to properly validate input that can affect the control flow or data flow of a program.

Recommendations

Validate input from untrusted sources before it is used. The untrusted data sources may include HTTP requests, file systems, databases, and any external systems that provide data to the application. In the case of HTTP requests, validate all parts of the request, including headers, form fields, cookies, and URL components that are used to transfer information from the browser to the server side application.

Duplicate any client-side checks on the server side. This should be simple to implement in terms of time and difficulty, and will greatly reduce the likelihood of insecure parameter values being used in the application.

Associated Flaws by CWE ID:

URL Redirection to Untrusted Site ('Open Redirect') (CWE ID 601)(7 flaws)

Description

A web application accepts a untrusted input that specifies a link to an external site, and uses that link to generate a redirect. This enables phishing attacks.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible. Check the supplied URL against a whitelist of approved URLs or domains before redirecting.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	704	3	-	JS files within dataroomscan.zip	.../dataroom-folders.component.ts 439	6/18/20
NEW	713	4	-	JS files within dataroomscan.zip	.../dataroom-team-members.component.ts 88	6/18/20
NEW	695	4	-	JS files within dataroomscan.zip	.../dataroom-team-members.component.ts 115	6/18/20
NEW	688	4	-	JS files within dataroomscan.zip	.../dataroom-team-members.component.ts 143	6/18/20
	433	5	-	JS files within dataroomscan.zip	.../file-comments.component.ts 43	6/18/20
NEW	686	6	-	JS files within dataroomscan.zip	.../folder-management-list.component.ts 178	6/18/20
NEW	718	16	-	JS files within dataroomscan.zip	.../manage-permission.component.ts 130	6/18/20

Low (3 flaws)

Information Leakage(3 flaws)

Description

An information leak is the intentional or unintentional disclosure of information that is either regarded as sensitive within the product's own functionality or provides information about the product or its environment that could be useful in an attack. Information leakage issues are commonly overlooked because they cannot be used to directly exploit the application. However, information leaks should be viewed as building blocks that an attacker uses to carry out other, more complicated attacks.

There are many different types of problems that involve information leaks, with severities that can range widely depending on the type of information leaked and the context of the information with respect to the application. Common sources of information leakage include, but are not limited to:

* Source code disclosure
* Browsable directories
* Log files or backup files in web-accessible directories
* Unfiltered backend error messages
* Exception stack traces
* Server version information
* Transmission of uninitialized memory containing sensitive data

Recommendations

Configure applications and servers to return generic error messages and to suppress stack traces from being displayed to end users. Ensure that errors generated by the application do not provide insight into specific backend issues.

Remove all backup files, binary archives, alternate versions of files, and test files from web-accessible directories of production servers. The only files that should be present in the application's web document root are files required by the application. Ensure that deployment procedures include the removal of these file types by an administrator. Keep web and application servers fully patched to minimize exposure to publicly-disclosed information leakage vulnerabilities.

Associated Flaws by CWE ID:

Exposure of Sensitive Information Through Environmental Variables (CWE ID 526)(2 flaws)

Description

This web server appears to be in a default configuration. Default configurations of web servers often provide too much information about their platform and version in HTTP headers and on error pages. This data is not itself dangerous, but it can help an attacker focus on vulnerabilities associated with your specific web server platform/version.

Effort to Fix: 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Recommendations

Configure your web server to avoid having it announce its own details. For example in Apache, these two configuration directives should be added to the configuration file: "ServerSignature Off" and "ServerTokens Prod". Utilize URLScan and IISLockdown for Microsoft's IIS web server.

Flaws found via Dynamic Scan

	Flaw Id	URL	Parameter	Exploitability
NEW	16	https://dataroom.exela.global:443/favicon.ico	Server: ECAcc (cha/81B4)	-
NEW	19	https://dataroom.exela.global:443/scripts.bundle.js	Server: Windows-	-

	Flaw Id	URL	Parameter	Exploitability
NEW			Azure-Web/1.0 Microsoft-HTTPAPI/2.0	

Exposure of Sensitive Information Through Sent Data (CWE ID 201)(1 flaw)

Description

Sensitive information may be exposed as a result of outbound network connections made by the application.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Ensure that the transfer of sensitive data is intended and that it does not violate application security policy or user expectations.

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
NEW	699	8	-	JS files within dataroomscan.zip	.../folderManagementController.js 3191	

Very Low (0 flaws)

No flaws of this type were found

Info (2 flaws)

Server Configuration(2 flaws)

Description

The application's supporting infrastructure, including web servers and application servers, can impact the security of the deployed application. Failing to lock down a server, for example, can result in information leaks via error pages, stack traces, or unnecessary files left in a web-accessible directory. Even though these servers are not part of the application codebase, they create insecurities in the environment which contribute to overall risk.

Recommendations

Remove all extraneous files, including demonstration applications and sample code, from production systems. Configure production servers with the minimum set of services required for the application to function, and ensure that information leaks do not occur via server-generated error pages.

Audit any third party dependencies or services that are deployed by default to ensure that they do not compromise the security of the application being supported.

Associated Flaws by CWE ID:

Configuration (CWE ID 16)(2 flaws)

Description

Weaknesses in this category are typically introduced during the configuration of the web server.

Effort to Fix: 2 - Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.

Recommendations

Follow specific recommendations attached to each flaw regarding how to configure the server in a more secure fashion.

Flaws found via Dynamic Scan

	Flaw Id	URL	Parameter	Exploitability
NEW	18	https://dataroom.exela.global:443/scripts.bundle.js	STS header	-
NEW	17	https://dataroom.exela.global:443/scripts.bundle.js	x-content-type-options header	-

About Veracode's Methodology

The Veracode platform uses static and dynamic analysis (for web applications) to identify software security flaws in your applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security flaws. Veracode static analysis models the application into an intermediate representation, which is then analyzed for security flaws using a set of automated security tests. Dynamic analysis uses an automated penetration testing technique to detect security flaws at runtime. Once the automated process is complete, a security technician verifies the output to ensure the lowest false positive rates in the industry. The end result is an accurate list of security flaws for the classes of automated scans applied to the application.

Veracode Rating System Using Multiple Analysis Techniques

Higher assurance applications require more comprehensive analysis to accurately score their security quality. Because each analysis technique (automated static, automated dynamic, manual penetration testing or manual review) has differing false negative (FN) rates for different types of security flaws, any single analysis technique or even combination of techniques is bound to produce a certain level of false negatives. Some false negatives are acceptable for lower business critical applications, so a less expensive analysis using only one or two analysis techniques is acceptable. At higher business criticality the FN rate should be close to zero, so multiple analysis techniques are recommended.

Application Security Policies

The Veracode platform allows an organization to define and enforce a uniform application security policy across all applications in its portfolio. The elements of an application security policy include the target Veracode Level for the application; types of flaws that should not be in the application (which may be defined by flaw severity, flaw category, CWE, or a common standard including OWASP, CWE/SANS Top 25, or PCI); minimum Veracode security score; required scan types and frequencies; and grace period within which any policy-relevant flaws should be fixed.

Policy constraints

Policies have three main constraints that can be applied: rules, required scans, and remediation grace periods.

Evaluating applications against a policy

When an application is evaluated against a policy, it can receive one of four assessments:

Not assessed The application has not yet had a scan published
Passed The application has passed all the aspects of the policy, including rules, required scans, and grace period.
Did not pass The application has not completed all required scans; has not achieved the target Veracode Level; or has one or more policy relevant flaws that have exceeded the grace period to fix.
Conditional pass The application has one or more policy relevant flaws that have not yet exceeded the grace period to fix.

Understand Veracode Levels

The Veracode Level (VL) achieved by an application is determined by type of testing performed on the application, and the severity and types of flaws detected. A minimum security score (defined below) is also required for each level.

There are five Veracode Levels denoted as VL1, VL2, VL3, VL4, and VL5. VL1 is the lowest level and is achieved by demonstrating that security testing, automated static or dynamic, is utilized during the SDLC. VL5 is the highest level and is achieved by performing automated and manual testing and removing all significant flaws. The Veracode Levels VL2, VL3, and VL4 form a continuum of increasing software assurance between VL1 and VL5.

For IT staff operating applications, Veracode Levels can be used to set application security policies. For deployment scenarios of different business criticality, differing VLs should be made requirements. For example, the policy for applications that handle credit card transactions, and therefore have PCI compliance requirements, should be VL5. A medium business criticality internal application could have a policy requiring VL3.

Software developers can decide which VL they want to achieve based on the requirements of their customers. Developers of software that is mission critical to most of their customers will want to achieve VL5. Developers of general purpose business software may want to achieve VL3 or VL4. Once the software has achieved a Veracode Level it can be communicated to customers through a Veracode Report or through the Veracode Directory on the Veracode web site.

Criteria for achieving Veracode Levels

The following table defines the details to achieve each Veracode Level. The criteria for all columns: Flaw Severities Not Allowed, Flaw Categories not Allowed, Testing Required, and Minimum Score.

*Dynamic is only an option for web applications.

Veracode Level	Flaw Severities Not Allowed	Testing Required*	Minimum Score
VL5	V.High, High, Medium	Static AND Manual	90
VL4	V.High, High, Medium	Static	80
VL3	V.High, High	Static	70
VL2	V.High	Static OR Dynamic OR Manual	60
VL1		Static OR Dynamic OR Manual	

When multiple testing techniques are used it is likely that not all testing will be performed on the exact same build. If that is the case the latest test results from a particular technique will be used to calculate the current Veracode Level. After 6 months test results will be deemed out of date and will no longer be used to calculate the current Veracode Level.

Business Criticality

The foundation of the Veracode rating system is the concept that more critical applications require higher security quality scores to be acceptable risks. Less business critical applications can tolerate lower security quality. The business criticality is dictated by the typical deployed environment and the value of data used by the application. Factors that determine business criticality are: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations.

US. Govt. OMB Memorandum M-04-04; NIST FIPS Pub. 199

Business Criticality	Description
Very High	Mission critical for business/safety of life and limb on the line
High	Exploitation causes serious brand damage and financial loss with long term business impact
Medium	Applications connected to the internet that process financial or private customer information
Low	Typically internal applications with non-critical business impact
Very Low	Applications with no material business impact

Business Criticality Definitions

Very High (BC5) This is typically an application where the safety of life or limb is dependent on the system; it is mission critical the application maintain 100% availability for the long term viability of the project or business. Examples are control software for industrial, transportation or medical equipment or critical business systems such as financial trading systems.
High (BC4) This is typically an important multi-user business application reachable from the internet and is critical that the application maintain high availability to accomplish its mission. Exploitation of high criticality applications cause serious brand damage and business/financial loss and could lead to long term business impact.
Medium (BC3) This is typically a multi-user application connected to the internet or any system that processes financial or private customer information. Exploitation of medium criticality applications typically result in material business impact resulting

in some financial loss, brand damage or business liability. An example is a financial services company's internal 401K management system.

Low (BC2) This is typically an internal only application that requires low levels of application security such as authentication to protect access to non-critical business information and prevent IT disruptions. Exploitation of low criticality applications may lead to minor levels of inconvenience, distress or IT disruption. An example internal system is a conference room reservation or business card order system.

Very Low (BC1) Applications that have no material business impact should its confidentiality, data integrity and availability be affected. Code security analysis is not required for applications at this business criticality, and security spending should be directed to other higher criticality applications.

Scoring Methodology

The Veracode scoring system, Security Quality Score, is built on the foundation of two industry standards, the Common Weakness Enumeration (CWE) and Common Vulnerability Scoring System (CVSS). CWE provides the dictionary of security flaws and CVSS provides the foundation for computing severity, based on the potential Confidentiality, Integrity and Availability impact of a flaw if exploited.

The Security Quality Score is a single score from 0 to 100, where 0 is the most insecure application and 100 is an application with no detectable security flaws. The score calculation includes non-linear factors so that, for instance, a single Severity 5 flaw is weighted more heavily than five Severity 1 flaws, and so that each additional flaw at a given severity contributes progressively less to the score.

Veracode assigns a severity level to each flaw type based on three foundational application security requirements — Confidentiality, Integrity and Availability. Each of the severity levels reflects the potential business impact if a security breach occurs across one or more of these security dimensions.

Confidentiality Impact

According to CVSS, this metric measures the impact on confidentiality if a exploit should occur using the vulnerability on the target system. At the weakness level, the scope of the Confidentiality in this model is within an application and is measured at three levels of impact -None, Partial and Complete.

Integrity Impact

This metric measures the potential impact on integrity of the application being analyzed. Integrity refers to the trustworthiness and guaranteed veracity of information within the application. Integrity measures are meant to protect data from unauthorized modification. When the integrity of a system is sound, it is fully proof from unauthorized modification of its contents.

Availability Impact

This metric measures the potential impact on availability if a successful exploit of the vulnerability is carried out on a target application. Availability refers to the accessibility of information resources. Almost exclusive to this domain are denial-of-service vulnerabilities. Attacks that compromise authentication and authorization for application access, application memory, and administrative privileges are examples of impact on the availability of an application.

Security Quality Score Calculation

The overall Security Quality Score is computed by aggregating impact levels of all weaknesses within an application and representing the score on a 100 point scale. This score does not predict vulnerability potential as much as it enumerates the security weaknesses and their impact levels within the application code.

The Raw Score formula puts weights on each flaw based on its impact level. These weights are exponential and determined by empirical analysis by Veracode's application security experts with validation from industry experts. The score is normalized to a scale of 0 to 100, where a score of 100 is an application with 0 detected flaws using the analysis technique for the application's business criticality.

Understand Severity, Exploitability, and Remediation Effort

Severity and exploitability are two different measures of the seriousness of a flaw. Severity is defined in terms of the potential impact to confidentiality, integrity, and availability of the application as defined in the CVSS, and exploitability is defined in terms of the likelihood or ease with which a flaw can be exploited. A high severity flaw with a high likelihood of being exploited by an attacker is potentially more dangerous than a high severity flaw with a low likelihood of being exploited.

Remediation effort, also called Complexity of Fix, is a measure of the likely effort required to fix a flaw. Together with severity, the remediation effort is used to give Fix First guidance to the developer.

Veracode Flaw Severities

Veracode flaw severities are defined as follows:

Severity	Description
Very High	The offending line or lines of code is a very serious weakness and is an easy target for an attacker. The code should be modified immediately to avoid potential attacks.
High	The offending line or lines of code have significant weakness, and the code should be modified immediately to avoid potential attacks.
Medium	A weakness of average severity. These should be fixed in high assurance software. A fix for this weakness should be considered after fixing the very high and high for medium assurance software.
Low	This is a low priority weakness that will have a small impact on the security of the software. Fixing should be consideration for high assurance software. Medium and low assurance software can ignore these flaws.
Very Low	Minor problems that some high assurance software may want to be aware of. These flaws can be safely ignored in medium and low assurance software.
Informational	Issues that have no impact on the security quality of the application but which may be of interest to the reviewer.

Informational findings

Informational severity findings are items observed in the analysis of the application that have no impact on the security quality of the application but may be interesting to the reviewer for other reasons. These findings may include code quality issues, API usage, and other factors.

Informational severity findings have no impact on the security quality score of the application and are not included in the summary tables of flaws for the application.

Exploitability

Each flaw instance in a static scan may receive an exploitability rating. The rating is an indication of the intrinsic likelihood that the flaw may be exploited by an attacker. Veracode recommends that the exploitability rating be used to prioritize flaw remediation within a particular group of flaws with the same severity and difficulty of fix classification.

The possible exploitability ratings include:

Exploitability	Description
V. Unlikely	Very unlikely to be exploited
Unlikely	Unlikely to be exploited

Exploitability	Description
Neutral	Neither likely nor unlikely to be exploited.
Likely	Likely to be exploited
V. Likely	Very likely to be exploited

Note: All reported flaws found via dynamic scans are assumed to be exploitable, because the dynamic scan actually executes the attack in question and verifies that it is valid.

Effort/Complexity of Fix

Each flaw instance receives an effort/complexity of fix rating based on the classification of the flaw. The effort/complexity of fix rating is given on a scale of 1 to 5, as follows:

Effort/Complexity of Fix	Description
5	Complex design error. Requires significant redesign.
4	Simple design error. Requires redesign and up to 5 days to fix.
3	Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.
2	Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.
1	Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Flaw Types by Severity Level

The flaw types by severity level table provides a summary of flaws found in the application by Severity and Category. The table puts the Security Quality Score into context by showing the specific breakout of flaws by severity, used to compute the score as described above. If multiple analysis techniques are used, the table includes a breakout of all flaws by category and severity for each analysis type performed.

Flaws by Severity

The flaws by severity chart shows the distribution of flaws by severity. An application can get a mediocre security rating by having a few high risk flaws or many medium risk flaws.

Flaws in Common Modules

The flaws in common modules listing shows a summary of flaws in shared dependency modules in this application. A shared dependency is a dependency that is used by more than one analyzed module. Each module is listed with the number of executables that consume it as a dependency and a summary of the impact on the application's security score of the flaws found in the dependency.

The score impact represents the amount that the application score would increase if all the flaws in the shared dependency module were fixed. This information can be used to focus remediation efforts on common modules with a higher impact on the application security score.

Only common modules that were uploaded with debug information are included in the Flaws in Common Modules listing.

Action Items

The Action Items section of the report provides guidance on the steps required to bring the application to a state where it passes its assigned policy. These steps may include fixing or mitigating flaws or performing additional scans. The section also includes best practice recommendations to improve the security quality of the application.

Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is an industry standard classification of types of software weaknesses, or flaws, that can lead to security problems. CWE is widely used to provide a standard taxonomy of software errors. Every flaw in a Veracode report is classified according to a standard CWE identifier.

More guidance and background about the CWE is available at http://cwe.mitre.org/data/index.html.

About Manual Assessments

The Veracode platform can include the results from a manual assessment (usually a penetration test or code review) as part of a report. These results differ from the results of automated scans in several important ways, including objectives, attack vectors, and common attack patterns.

A manual penetration assessment is conducted to observe the application code in a run-time environment and to simulate real-world attack scenarios. Manual testing is able to identify design flaws, evaluate environmental conditions, compound multiple lower risk flaws into higher risk vulnerabilities, and determine if identified flaws affect the confidentiality, integrity, or availability of the application.

Objectives

The stated objectives of a manual penetration assessment are:

- Perform testing, using proprietary and/or public tools, to determine whether it is possible for an attacker to:
- Circumvent authentication and authorization mechanisms
- Escalate application user privileges
- Hijack accounts belonging to other users
- Violate access controls placed by the site administrator
- Alter data or data presentation
- Corrupt application and data integrity, functionality and performance
- Circumvent application business logic
- Circumvent application session management
- Break or analyze use of cryptography within user accessible components
- Determine possible extent access or impact to the system by attempting to exploit vulnerabilities
- Score vulnerabilities using the Common Vulnerability Scoring System (CVSS)
- Provide tactical recommendations to address security issues of immediate consequence

Provide strategic recommendations to enhance security by leveraging industry best practices

Attack vectors

In order to achieve the stated objectives, the following tests are performed as part of the manual penetration assessment, when applicable to the platforms and technologies in use:

- Cross Site Scripting (XSS)
- SQL Injection
- Command Injection
- Cross Site Request Forgery (CSRF)
- Authentication/Authorization Bypass
- Session Management testing, e.g. token analysis, session expiration, and logout effectiveness
- Account Management testing, e.g. password strength, password reset, account lockout, etc.
- Directory Traversal
- Response Splitting
- Stack/Heap Overflows
- Format String Attacks

- Cookie Analysis
- Server Side Includes Injection
- Remote File Inclusion
- LDAP Injection
- XPATH Injection
- Internationalization attacks
- Denial of Service testing at the application layer only
- AJAX Endpoint Analysis
- Web Services Endpoint Analysis
- HTTP Method Analysis
- SSL Certificate and Cipher Strength Analysis
- Forced Browsing

CAPEC Attack Pattern Classification

The following attack pattern classifications are used to group similar application flaws discovered during manual penetration testing. Attack patterns describe the general methods employed to access and exploit the specific weaknesses that exist within an application. CAPEC (Common Attack Pattern Enumeration and Classification) is an effort led by Cigital, Inc. and is sponsored by the United States Department of Homeland Security's National Cyber Security Division.

Abuse of Functionality

Exploitation of business logic errors or misappropriation of programmatic resources. Application functions are developed to specifications with particular intentions, and these types of attacks serve to undermine those intentions.

Examples:

- Exploiting password recovery mechanisms
- Accessing unpublished or test APIs
- Cache poisoning

Spoofing

Impersonation of entities or trusted resources. A successful attack will present itself to a verifying entity with an acceptable level of authenticity.

Examples:

- Man in the middle attacks
- Checksum spoofing
- Phishing attacks

Probabilistic Techniques

Using predictive capabilities or exhaustive search techniques in order to derive or manipulate sensitive information. Attacks capitalize on the availability of computing resources or the lack of entropy within targeted components.

Examples:

- Password brute forcing
- Cryptanalysis
- Manipulation of authentication tokens

Exploitation of Authentication

Circumventing authentication requirements to access protected resources. Design or implementation flaws may allow authentication checks to be ignored, delegated, or bypassed.

Examples:

- Cross-site request forgery
- Reuse of session identifiers
- Flawed authentication protocol

Resource Depletion

Affecting the availability of application components or resources through symmetric or asymmetric consumption. Unrestricted access to computationally expensive functions or implementation flaws that affect the stability of the application can be targeted by an attacker in order to cause denial of service conditions.

Examples:

- Flooding attacks
- Unlimited file upload size
- Memory leaks

Exploitation of Privilege/Trust

Undermining the application's trust model in order to gain access to protected resources or gain additional levels of access as defined by the application. Applications that implicitly extend trust to resources or entities outside of their direct control are susceptible to attack.

Examples:

- Insufficient access control lists
- Circumvention of client side protections
- Manipulation of role identification information

Injection

Inserting unexpected inputs to manipulate control flow or alter normal business processing. Applications must contain sufficient data validation checks in order to sanitize tainted data and prevent malicious, external control over internal processing.

Examples:

- SQL Injection
- Cross-site scripting
- XML Injection

Data Structure Attacks

Supplying unexpected or excessive data that results in more data being written to a buffer than it is capable of holding. Successful attacks of this class can result in arbitrary command execution or denial of service conditions.

Examples:

- Buffer overflow
- Integer overflow
- Format string overflow

Data Leakage Attacks

Recovering information exposed by the application that may itself be confidential or may be useful to an attacker in discovering or exploiting other weaknesses. A successful attack may be conducted passive observation or active interception methods. This attack pattern often manifests itself in the form of applications that expose sensitive information within error messages.

Examples:

- Sniffing clear-text communication protocols
- Stack traces returned to end users
- Sensitive information in HTML comments

Resource Manipulation

Manipulating application dependencies or accessed resources in order to undermine security controls and gain unauthorized access to protected resources. Applications may use tainted data when constructing paths to local resources or when constructing processing environments.

Examples:

- Carriage Return Line Feed log file injection
- File retrieval via path manipulation
- User specification of configuration files

Time and State Attacks

Undermining state condition assumptions made by the application or capitalizing on time delays between security checks and performed operations. An application that does not enforce a required processing sequence or does not handle concurrency adequately will be susceptible to these attack patterns.

Examples:

- Bypassing intermediate form processing steps
- Time-of-check and time-of-use race conditions
- Deadlock triggering to cause a denial of service

Terms of Use

Use and distribution of this report are governed by the agreement between Veracode and its customer. In particular, this report and the results in the report cannot be used publicly in connection with Veracode's name without written permission.

Appendix A: Changes from Last Scan

Latest Scan		Prior Scan	
Static Scan			
Scan Name:	18 Jun 2020 Static	Scan Name:	11 Feb 2020 Static
Completed:	6/18/20	Completed:	2/11/20
Score:	77	Score:	91
Dynamic Scan			
Scan Name:	Dataroom (1)	Scan Name:	Dataroom
Completed:	12/12/19	Completed:	12/11/19
Score:	99	Score:	99

Changes in scope of scan (static)

New Modules

Module Name	Compiler	Operating Environment	Engine Version
JS files within dataroomscan.zip	JAVASCRIPT_5_1	JavaScript	20200520200321
Python files within dataroomscan.zip	Python		20200505194254

Removed modules

Module Name	Compiler	Operating Environment	Engine Version
JS files within BE_DATAROOM.zip	JAVASCRIPT_5_1	JavaScript	20200124175904
JS files within FE_DATAROOM.zip	JAVASCRIPT_5_1	JavaScript	20200124175904
Python files within FE_DATAROOM.zip	Python		20200124175904

Changes in scope of scan (dynamic)

The following dynamic scan parameters have changed since the last dynamic scan was run:

Setting	18 Jun 2020 Static	Dataroom
Number of Links Visited	49	61
Successful Logins	4 of 14	4 of 16
Login User ID	test.account	HQ.Client.Audits
Scan Window	12/11/2019 @ 11:56 PM EST - 12/12/2019 @ 11:52 PM EST	12/10/2019 @ 10:24 AM EST - 12/11/2019 @ 10:21 AM EST
Scan Duration	26 Minutes	1 Hour 3 Minutes

Flaws not detected in current scan

The following is a list of all flaws found in the prior scan of this application that were not detected in the current scan.

Medium (16 flaws)

Fix Required by Policy

CRLF Injection(1 flaw)

Associated Flaws by CWE ID:

Improper Output Neutralization for Logs (CWE ID 117)(1 flaw)

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	684	7	-	JS files within FE_DATAROOM.zip	.../folder-management-navigator.component.ts 1522	

Credentials Management(10 flaws)

Associated Flaws by CWE ID:

Use of Hard-coded Credentials (CWE ID 798)(6 flaws)

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	441	12	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-en.ts 8	
	263	12	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-en.ts 10	
	154	13	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-es.ts 9	
	307	13	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-es.ts 10	
	460	14	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-fr.ts 8	
	430	14	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-fr.ts 9	

Use of Hard-coded Password (CWE ID 259)(4 flaws)

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	137	12	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-en.ts 7	

	Flaw Id	Module #	Class #	Module	Location	Fix By
	149	12	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-en.ts 49	
	158	13	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-es.ts 8	
	410	13	-	JS files within FE_DATAROOM.zip	FE_DATAROOM/.../lang-es.ts 49	

Cross-Site Scripting (XSS)(5 flaws)

Associated Flaws by CWE ID:

Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (CWE ID 80)(5 flaws)

Instances found via Static Scan

	Flaw Id	Module #	Class #	Module	Location	Fix By
	671	8	-	JS files within BE_DATAROOM.zip	.../folderManagementController.js 661	
	631	8	-	JS files within BE_DATAROOM.zip	.../folderManagementController.js 798	
	585	8	-	JS files within BE_DATAROOM.zip	.../folderManagementController.js 911	
	541	8	-	JS files within BE_DATAROOM.zip	.../folderManagementController.js 1710	
	586	8	-	JS files within BE_DATAROOM.zip	.../folderManagementController.js 2616	

Appendix B: Referenced Source Files

Id	Filename	Path
1	carousel.js	/dataroomscan/FE_UI/app/modules/shared/directives/carousel/
2	dataroom-create-group.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/dataroom-group/create-group/dataroom-create-group-tab/
3	dataroom-folders.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/dataroom-group/create-group/dataroom-folders-tab/
4	dataroom-team-members.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/dataroom-group/create-group/dataroom-team-members-tab/
5	file-comments.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/folder-management/folder-management-list/file-comments/
6	folder-management-list.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/folder-management/folder-management-list/
7	folder-management-navigator.component.ts	FE_DATAROOM/src/app/modules/Boxoffice/folder-management/folder-management-navigator/
8	folderManagementController.js	BE_DATAROOM/boxoffice/controller/
9	group-permission.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/folder-management/folder-management-navigator/group-permission/
10	header.component.ts	/dataroomscan/FE_UI/app/modules/shared/components/header/
11	http.service.ts	/dataroomscan/FE_UI/app/modules/shared/providers/
12	lang-en.ts	FE_DATAROOM/src/app/translate/
13	lang-es.ts	FE_DATAROOM/src/app/translate/
14	lang-fr.ts	FE_DATAROOM/src/app/translate/
15	listViewController.js	/dataroomscan/BE_CORE/controller/
16	manage-permission.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/folder-management/folder-management-navigator/manage-permission/
17	ng-pagination.component.ts	/dataroomscan/FE_UI/app/modules/shared/modules/ng-data-tables/ng-pagination/
18	pagination.js	/dataroomscan/FE_UI/app/modules/shared/modules/ng-data-tables/ng-pagination/
19	pdf-editor.component.ts	/dataroomscan/FE_UI/app/modules/shared/components/pdf-editor/
20	snackbar.component.ts	/dataroomscan/FE_UI/app/modules/shared/components/snackbar/
21	templateProfileController.js	/dataroomscan/BE_CORE/controller/
22	text-view.component.html	/dataroomscan/FE_UI/app/modules/Boxoffice/folder-management/folder-management-list/text-view/
23	upload-template.component.ts	/dataroomscan/FE_UI/app/modules/Boxoffice/folder-management/folder-management-list/uploadtemplate/

Appendix C: Dynamic Flaw Inventory

Rescan Status	Number of Flaws
All	5
New	4
Open and Reopened	0
Cannot Reproduce	1
Fixed	0

Non-Reproducible Flaws

Veracode rescanned the application and the new results did not include the flaws listed in this table. Veracode could not determine if the flaws are fixed, and will remove them from this list after ten scans of the application.

Flaw ID	CWE ID and Name	Severity	Date Found	Path
2	693 & Protection Mechanism Failure	3	11 Dec 2019	dataroom.exela.global