Veracode Summary Report

Summary Report
As of 18 Mar 2016

Prepared for:	ownCloud Inc.
Prepared on:	April 4, 2016
Application:	ownCloud Server

Industry:	Software
Business Criticality:	BC5 (Very High)
Required Analysis:	Static
Type(s) of Analysis Conducted:	Static
Scope of Static Scan:	2 of 3 Modules Analyzed

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 Summary Report

Summary Report for ownCloud Inc.

Mitigated Veracode Level: VL4

Original Veracode Level: VL2

Rated: Mar 18, 2016

Application: ownCloud Server

Adjusted/Published Rating: A*/D

Scans Included in Report

Static Scan	Dynamic Scan	Manual Scan
17 Mar 2016 Static Score: 100 Completed: 3/18/16	Not Included in Report	Not Included in Report

About this Analysis

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

Analyses Performed vs. Required

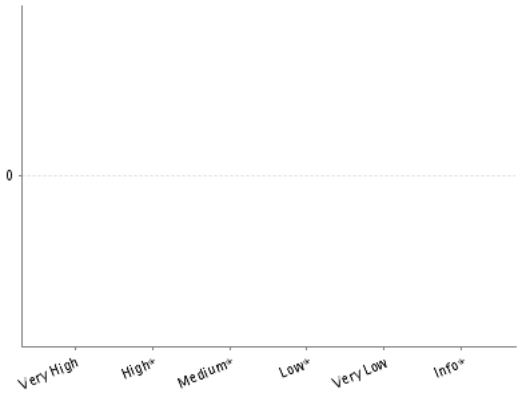
	Any	Static	Dynamic	Manual
Performed:		●	○	○
Required:	○	●	○	○

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.

Security Flaws by Severity



Top Risks

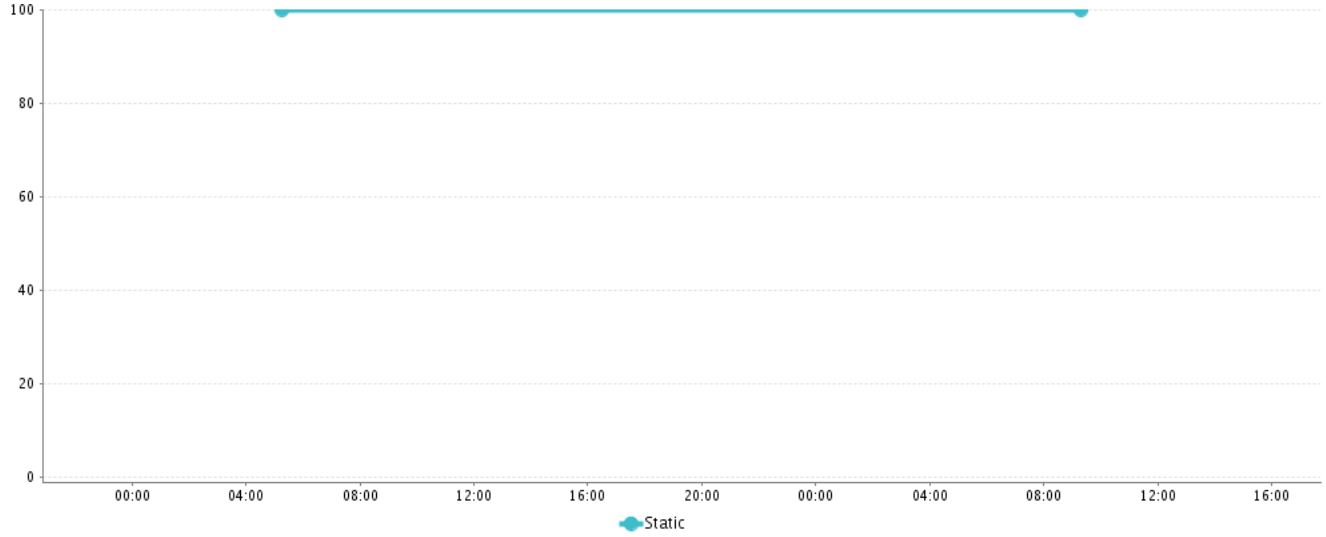
Top security flaws detected in the application, ordered by score impact, included:

Total Flaws detected in application: 0*

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

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

Engine Version: 90542

The following modules were included in the application scan:

Module Name	Compiler	Operating Environment	Engine Version
JS files within daily.zip	JAVASCRIPT_5_1	JavaScript	90542
PHP files within daily.zip	PHP_5	PHP	90542

The following modules were not selected for a full scan. Code paths in these modules that are not called from a scanned module are not included in this report.

Module Name	Compiler	Operating Environment	Engine Version
hiddeninput.exe	MSVC9_X86	Win32	90542

Security Improvement Roadmap for ownCloud Server 17 Mar 2016 Static - Not Specified

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 periodic Static Scan. Your next analysis must be completed by 6/18/16. Please submit your application for Static Scan by the deadline and remediate the required detected flaws to conform to your assigned policy.

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

Policy Evaluation

Policy Name: Veracode Recommended High

Revision: 1

Policy Status: 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	VL4	VL4*	Passed
(VL4) Min Analysis Score	80	100*	Passed
(VL4) Max Severity	Medium	Flaws found: 0*	Passed

* - Reflects violated rules that have mitigated flaws

Scan Requirements

Scan Type	Frequency	Last performed	Status
Static	Quarterly	3/18/16	Passed

Remediation

Flaw Severity	Grace Period	Flaws Exceeding	Status
Very High	0 days	0	Passed
High	0 days	0	Passed
Medium	0 days	0	Passed
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	0	Passed

About Veracode's Methodology

The Veracode platform uses static and dynamic analysis (for web applications) to inspect executables and identify security flaws in your applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security flaws. The static binary analysis engine models the binary executable into an intermediate representation, which is then verified for security flaws using a set of automated security scans. 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.

65 Network Drive, Burlington, MA 01803

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

Appendix A: Accepted Mitigated Flaws (by ownCloud Inc.)

NOTE: Veracode does not review the mitigation strategy described below and is not responsible for its contents or the accuracy of any statements provided.

High (16 flaws)

Fix Required by Policy - Mitigation Accepted

Code Injection(16 flaws)

Associated Flaws by CWE ID:

Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (CWE ID 98)(16 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2461	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - This function is only called with trusted input and this is thus not exploitable. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2455	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - This function is only called with trusted input. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2486	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Input is taken from \OC::$server->getAppManager()->getInstalledApps() which is trusted input *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2453	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $file comes from \OC_App::getEnabledApps which is trusted input only accessible to administrators. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2405	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $vendorAutoLoad is from "OC::$THIRDPARTYROOT . '/3rdparty/autoload.php'" wheras "OC::$THIRDPARTYROOT" contains only trusted input from the config file. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2427	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - This file can only be invoked by an administrator

Flaw Id	Module	Exploitability	Mitigation Comment
			using the OCC tool. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2477	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $themePath contains the path to the currently used theme. It is generated using "OC::$SERVERROOT . '/themes/' . OC_Util::getTheme() . '/defaults.php'" whereas OC_Util::getTheme() is read from the configuration file. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2459	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - This is called by methods such as "\OC_Installer::installApp", those are only reachable for administrators which anyways could install a malicious application. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2417	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $app and $parts come from the database with trusted input. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2387	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $file comes from the database with trusted input. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2422	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $file contains only trusted input pointing to installed applications. *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2397	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $this->autoConfigFile contains only trusted data (\OC::$SERVERROOT.'/config/autoconfig.php') *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2410	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - \OC::$SERVERROOT contains no user-controlled input. *Mitigation Accepted (Lukas Reschke):* Promoted from

Flaw Id	Module	Exploitability	Mitigation Comment
			Sandbox - Accept all proposed mitigations
2475	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - Only called with trusted input from \OC_App::getEnabledApps *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2403	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - OC::$SERVERROOT is generated using "str_replace("\\", '/', substr(__DIR__, 0, -4))" in lib/base.php and contains no user controlled input. This is thus a not exploitable false positive *Mitigation Accepted (Lukas Reschke):* Promoted from Sandbox - Accept all proposed mitigations
2476	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - Data comes from \OCA\Files\App::getNavigationManager()->getAll() which is trusted input provided by other applications. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.

Medium (83 flaws)

Fix Required by Policy - Mitigation Accepted

Directory Traversal(35 flaws)

Associated Flaws by CWE ID:

External Control of File Name or Path (CWE ID 73)(35 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2487	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* \OC_App::getAppVersionByPath is only called by trusted input *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2406	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* self::$configDir is build as OC::$SERVERROOT . '/config/' and contains no user input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Flaw Id	Module	Exploitability	Mitigation Comment
2421	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Only called for copying applications which can only be invoked by administrators. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2464	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - Only called from \OC_Util::copySkeleton which is trusted input *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2456	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - Function is not invoked with any untrusted user input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2390	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* rmdirr is not called with user input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2448	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* $fileName contains always only trusted values. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2474	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* $fileName contains always only trusted values. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2388	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* installApp can only be executed by administrators. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2447	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - \OC_Installer::installApp can only be invoked by administrators *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2473	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* \OC_Installer::installApp can only be invoked by administrators. Furthermore $path references a temporary resource that has to be deleted after extraction.

Flaw Id	Module	Exploitability	Mitigation Comment
			Mitigation Accepted (Lukas Reschke): Accept proposed mitigations
2438	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* \OC_Installer::downloadApp is only invokable by administrators and here the downloaded application would be removed from it's temporary directory. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2391	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - $path is trusted input provided by ownCloud linking to a temporary directory. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2488	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* checkAppsIntegrity is only called by trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2468	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* $transFile comes from \OC\L10N\L10N::__construct which gets the values from \OC\L10N\Factory::get which only loads trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2412	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Function does what it is expected to do. ownCloud does also not use this functionality. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2430	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Function does what it is expected to do. ownCloud does also not use this functionality. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2466	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* \OC_Response::sendFile does what the function is expected to do. Send the file $filePath. This functionality is not used in ownCloud itself and third-party applications have to ensure that the passes path makes sense. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2409	PHP files within	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only

Flaw Id	Module	Exploitability	Mitigation Comment
	daily.zip		called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2399	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2457	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2423	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2414	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2450	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2411	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2437	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed

Flaw Id	Module	Exploitability	Mitigation Comment
			mitigations.
2449	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2415	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2424	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2463	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2426	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2454	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2481	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2472	PHP files within	Neutral	*Mitigated by Design (Lukas Reschke):* OC_Archive is only

Flaw Id	Module	Exploitability	Mitigation Comment
	daily.zip		called from \OC_Installer::downloadApp which contains only trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2432	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - OC::$SERVERROOT is trusted input. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.

Cross-Site Scripting(2 flaws)

Associated Flaws by CWE ID:

Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (CWE ID 80)(2 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2441	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Passed input is already sanitized before using "\OCP\Util::sanitizeHTML" *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2143	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* The used jQuery version is properly escaping input passed to ".attr(" *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Insufficient Input Validation(2 flaws)

Associated Flaws by CWE ID:

URL Redirection to Untrusted Site ('Open Redirect') (CWE ID 601)(2 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
1707	JS files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Redirection only happens to trusted endpoints. This is required for OAuth 2 integration with external storages.

Flaw Id	Module	Exploitability	Mitigation Comment
			Mitigation Accepted (Lukas Reschke): Accept proposed mitigations
2010	JS files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* Redirection only happens to trusted endpoints. This is required for OAuth 2 integration with external storages. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Credentials Management(15 flaws)

Associated Flaws by CWE ID:

Use of Hard-coded Credentials (CWE ID 798)(1 flaw)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2049	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Not an hard-coded password but simply the use of the string "userName" *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Use of Hard-coded Password (CWE ID 259)(14 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2483	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a placeholder. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2442	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a variable name containing the word "pass" *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2418	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Not an hard-coded password and

Flaw Id	Module	Exploitability	Mitigation Comment
			only the NTLM derivation function of SwiftMailer. This functionality is also not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2383	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Simple boolean check whether a password is set or not. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2384	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a placeholder. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2183	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a placeholder. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2094	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a placeholder. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2433	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a placeholder. *Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a placeholder. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2086	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Not an hard-coded password but simply a JavaScript integration test. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
1876	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Not an hard-coded password but simply a JavaScript integration test. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
1932	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a JavaScript integration test. *Mitigation Accepted (Lukas Reschke):* Accept proposed

Flaw Id	Module	Exploitability	Mitigation Comment
			mitigations.
1846	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a JavaScript integration test. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
1958	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a JavaScript integration test. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
1724	JS files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Promoted from Sandbox - Only a JavaScript integration test. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.

Cryptographic Issues(29 flaws)

Associated Flaws by CWE ID:

Use of a Broken or Risky Cryptographic Algorithm (CWE ID 327)(29 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2381	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only a caching key. No cryptographic action is performed. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2395	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used as a caching key. Not cryptographic action is performed. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2389	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 as required by Cram MD5 authentication. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2458	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 as required by CramMD5 authentication. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Flaw Id	Module	Exploitability	Mitigation Comment
2444	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 as required by CramMD5 authentication. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2480	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used to hash a key so that it has expected values. No cryptographic impact. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2392	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used as a caching key. Not cryptographic action is performed. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2419	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 is used for generating a caching key. No security impact. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2429	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 only used to verify integrity. This functionality is furthermore not used by ownCloud and will be removed with 9.1. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2436	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used as checksum over the file. No cryptographic context. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2439	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used as checksum over the file. No cryptographic context. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2440	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* CRC32 as required by GZIP. No cryptographic context. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2452	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* CRC32 as required by GZIP. No cryptographic context. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Flaw Id	Module	Exploitability	Mitigation Comment
2469	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used as caching key. No cryptographic context. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2416	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 is used for generating a caching key. No security impact. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2382	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 as required by NTLM authentication. Furthermore, this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2401	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 as required by NTLM. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2402	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD4 as required by NTLM. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2407	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used as a caching key. No cryptographic relevance. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2434	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 is used to generate the file ETag. This is not cryptographically relevant. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2482	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used to verify integrity of transer to S3. No cryptographic relevance. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2435	PHP files within	Likely	*Mitigated as Potential False Positive (Lukas Reschke):*

Flaw Id	Module	Exploitability	Mitigation Comment
	daily.zip		CRC32 to check whether DynamoDB request needs to be retried. Not cryptographically relevant. Furthermore this functionality is not used by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2398	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Checksum as required by Amazon. No cryptographic relevance. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2465	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 over the content as required by S3 for integrity. No cryptographic context. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2378	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* No cryptographic action performed. This is there to check the integrity of the transferred information to AWS. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2413	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* MD5 is used for generating a caching key. No security impact. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2478	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Used to hash an unique identifier so the values are valid. No cryptographic context. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2396	PHP files within daily.zip	Likely	*Mitigated as Potential False Positive (Lukas Reschke):* Only used to hash a random identifier. No cryptographic relevance. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations
2462	PHP files within daily.zip	Likely	*Mitigated by Design (Lukas Reschke):* File is not used by ownCloud and cannot be invoked. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations

Low (4 flaws)

Information Leakage(4 flaws)

Associated Flaws by CWE ID:

Information Exposure Through an Error Message (CWE ID 209)(4 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2385	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Promoted from Sandbox - RuntimeException is only thrown in case of fatal errors like "3rdparty directory not found" with manually defined exception messages. This is expected behaviour and no information is leaked here. *Mitigation Accepted (Lukas Reschke):* Accept proposed mitigations.
2445	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Code path is not reached. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2467	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Code path is not reached. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2470	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Functionality is not used by ownCloud and thus code path is not reached. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations

Info (18 flaws)

Untrusted Initialization(18 flaws)

Associated Flaws by CWE ID:

External Initialization of Trusted Variables or Data Stores (CWE ID 454)(18 flaws)

Instances found via Static Scan

Flaw Id	Module	Exploitability	Mitigation Comment
2425	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Code path is disabled in ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations

Flaw Id	Module	Exploitability	Mitigation Comment
2451	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed arguments are escaped using "escapeshellarg" *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2394	PHP files within daily.zip	Neutral	*Mitigated by OS (Lukas Reschke):* Only called on Microsoft Windows operating systems. ownCloud does not support Microsoft Windows as server system. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2404	PHP files within daily.zip	Neutral	*Mitigated by OS (Lukas Reschke):* Only called on Microsoft Windows operating systems. ownCloud does not support Microsoft Windows as server system. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2428	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed values are properly escaped. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2460	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Code is not executed by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2379	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed values are hard-coded. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2471	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed values are escaped using "escapeshellarg" *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2386	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed value is trusted as it comes from the system. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2408	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Only called with trusted input. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2380	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed value is hard-coded.

Flaw Id	Module	Exploitability	Mitigation Comment
			Mitigation Accepted (Lukas Reschke): Accept all proposed mitigations
2393	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed value is hard-coded. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2443	PHP files within daily.zip	Neutral	*Mitigated by OS (Lukas Reschke):* Only called on Microsoft Windows operating systems. ownCloud does not support Microsoft Windows as server system. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2479	PHP files within daily.zip	Neutral	*Mitigated as Potential False Positive (Lukas Reschke):* Passed values are escaped using "escapeshellarg" *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2420	PHP files within daily.zip	Neutral	*Mitigated by OS (Lukas Reschke):* Only called on Microsoft Windows operating systems. ownCloud does not support Microsoft Windows as server system. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2400	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* Function is not used by ownCloud. Code path is thus never reached. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2446	PHP files within daily.zip	Neutral	*Mitigated by OS (Lukas Reschke):* Only called on Microsoft Windows operating systems. ownCloud does not support Microsoft Windows as server system. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations
2484	PHP files within daily.zip	Neutral	*Mitigated by Design (Lukas Reschke):* \getid3_write_vorbiscomment::WriteVorbisComment is not called by ownCloud. *Mitigation Accepted (Lukas Reschke):* Accept all proposed mitigations