Secure Coding Guidelines for Application Development

Guidelines for Secure Coding

Contents

Version History

Version	Date	Section	Change by
1.0	Nov 10, 2022	All	Abhishek Shroti
1.2	Jan 2023	Reworded few sections and minor changes.	Ambar Pande

1. Document Details

Name	Secure Coding Guidelines for Application Development
Version	1.0
Date	Nov 24, 2022
Author	Abhishek Shroti

2. Document Reviewers

Name	Title	Date Reviewed
Ambar Pande	Principal Consultant	Nov 28 2022

3. Purpose

Cyber threats are increasing day by day. In such a scenario, writing secure code becomes an important requisite part of the secure software development life cycle. Most of the time either the novice or experienced developers tend to write working code due to time constraints or do not have adequate security framework to write secure code.
Additionally, in common situation, most of the part of testing is focused on functional testing. Such practices can lead to security vulnerabilities/flaws and it could either lead to the crashing of applications or security attacks.

With above in context, making developers aware of secure coding practice, becomes of paramount importance.

Secure coding for Cyber security is a vast topic. This document intends to provide development teams about major points to be taken care of while developing code.

This document is intended for developers.

4. Scope

The scope of this secure coding guideline applies to all developers and testers which are involved in development/testing, who have or are responsible for the development of any Program/Application/API/Microservices. Anyone involved in development may follow these guidelines for improving application security and thus reducing vulnerabilities.

5. Definition

S.No	Term	Definition
1	ISMS	Information Security Management System
2	IT	Information Technology
3	MFA	Multi Factor Authentication

6. Guidelines

- All the developers / testers may follow Secure Coding standards while developing / designing any solution / application.
- Practicing secure software design and other phases of software engineering life cycle for a structured, small, and simple code. In addition, make use of a secure coding checklist.
- In order to track changes made to the code or document, use version/configuration control, this enables easy rollback to a previous version in case of a serious mistake.

- Good practice also recommends the use of the latest compilers, which often include defenses against coding errors.
- Security Design Patterns can be used to tackle similar security-related concerns and provide solutions to known problems (Example, Java).
- Use reputed language libraries to protect the web application from security vulnerabilities (Example, Cross-Site Scripting). Libraries Example, helmet and DOMPurify for Node.JS
- Developers / Application owners need to take the sign off from the Cybersecurity team before moving code to the production environments.
- Cybersecurity team shall be responsible for making the developers aware of the best practices at least once every quarter.
- All systems & applications shall be designed to adhere to strong password policy.

7. Procedure

7.1. Application Patching

Regular scanning of vendor-supplied security patches or mitigation of a reported vulnerability, in accordance with the Vulnerability Management Standard, are critical components in protecting the integrity of Network, Systems and Applications.

7.2. Use of Environment

- **Production Environment**

The production environment contains applications or services which are intended for the end-users. Before moving anything to the production environment or even making slight modifications, the permission of involved parties must be taken and tracked using Email.

- **Pre-Production/Staging/QA**

The applications need to be thoroughly verified using different mechanisms i.e automated tools, manual tools and QA Testing to remove the fail cases. If any security vulnerabilities still exists, prior permission shall be taken by the cybersecurity team before deploying it to production.

8. Security Vulnerabilities Checklist for Developers (Best Practices)

- **Input Validation:** User Input is one of the most important entry points for any application, this gives attackers a pathway to inject various attack vectors inside the application. Therefore, it is necessary to take care of how our applications process the user input which is received. Priority must be given to filter out and sanitize while rendering user-controllable values to any application. For the same, we recommend the following measures to follow while writing code.

- **Validation-check:** User Input validation must also be performed on the server-side rather than depending on checks on client-side as it can be bypassed.
- **Data-type:** Input type must be checked with required data type eg- string, integer and also reject unwanted input types, eg: a function needs only integer in such a case if not required reject boolean or string values
- **Add check on data type:** In case of integer inputs allow inputs under a specific range only as required by application which would prevent overflow issues.
- **Sanitization:** Input sanitization should always be carried in order to block harmful and unwanted characters
- **Encoding:** Encode CR & LF characters (\r, \n) so that even when they're supplied, they aren't recognized and processed by the server.
- **Central Validation:** There should be a centralized input validation method that can be called for validation each time.
- **Validate client data:** Validate all client provided data before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values).
- Validate data range and length and if failed it must be rejected.

- **Output Encoding**
  - Conduct all encoding on a trusted system
  - Encode all characters unless they are known to be safe for the intended interpreter
  - **Sanitization:** Input sanitization should always be carried out in order to block harmful and unwanted characters.

- **Authentication and Password Management**
  - All authentication controls should fail securely without exposing information.
  - Use a centralized implementation for all authentication controls, including libraries that call external authentication services.
  - All authentication controls must be enforced on a trusted system
  - Password hashing must be implemented on a trusted system.
  - Authenticate all the Web Pages/API/ Resources/ unless it is meant to be public.

    - Authentication error should not clearly state which start of the authentication system is wrong. Ex: “Username or Password is wrong”.
    - Use only HTTP POST requests to transmit authentication credentials.
    - Enforce password length requirements established by policy or regulation.
    - Notify users when a password reset occurs.
    - Re-authenticate users prior to performing critical operations.
    - Use MFA for highly sensitive or high-value transactional accounts

- **Session Management**
    - Session identifier creation must always be done on a trusted system.
    - Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site.
    - Generate a new session identifier on any re-authentication.
    - JSON Web Token must be implemented by following the standard guidelines.
    - The token secret must be stored in a key management service and every time the token must be verified to validate the status of the token.

- **Access Control**
    - Access controls should fail securely.
    - Restrict access to protected URLs to only authorized users.
    - Restrict access to protected functions to only authorized users.
    - Restrict access to services to only authorized users.
    - Restrict access to application data to only authorized users.
    - Enforce application logic flows to comply with business rules.
    - Restrict access to application data to only authorized users.
    - Create an Access Control Policy to document application business rules, types and authorization access criteria.

- **Cryptographic Practices**
    - All cryptographic functions used to protect secrets from the application user must be implemented on a trusted system.
    - Protect master secrets from unauthorized access.
    - Make use of the Key Management service to store the important key files.
    - Cryptographic modules should fail securely.
    - Establish and utilize a policy and process for how cryptographic keys will be managed.

- **Error Handling and Logging**
    - Do not disclose sensitive information in error responses, including system details, session identifiers or account information.
    - Use error handlers that do not display debugging or stack trace information
    - Implement generic error messages and use custom error pages.
    - The application should handle application errors and not rely on the server configuration.
    - Error handling logic associated with security controls should deny access by default.
    - Restrict access to logs to only authorized individuals.
    - Log all input validation failures.
    - Log all access control failures
    - Log all system exceptions
    - Log attempts to connect with invalid or expired session tokens
    - Log all apparent tampering events, including unexpected changes to state data.

- **Data Protection**
    - Implement least privilege, restrict users to only the functionality, data and system information that is required to perform their tasks.
    - Protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files as soon as they are no longer required.
    - Encrypt highly sensitive stored information, like authentication verification data, even on the server-side. Always use well-vetted algorithms, see "Cryptographic Practices".
    - Protect server-side source code from being downloaded by a user.
    - Do not store passwords, connection strings or other sensitive information in clear text or in any non cryptographically secure manner on the client-side.
    - Remove comments in user-accessible production code that may reveal backend systems or other sensitive information.
    - Do not include sensitive information in HTTP GET request parameters.
    - The application should support the removal of sensitive data when that data is no longer required.

- **Communication Security**
    - Implement encryption for the transmission of all sensitive information. This should include TLS for protecting the connection and may be supplemented by discrete encryption of sensitive files or non HTTP based connections.
    - TLS certificates should be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required.
    - Failed TLS connections should not fall back to an insecure connection.
    - Utilize TLS connections for all content requiring authenticated access and for all other sensitive information.
    - Specify character encodings for all connections.
    - Filter parameters containing sensitive information from the HTTP referer, when linking to external sites.

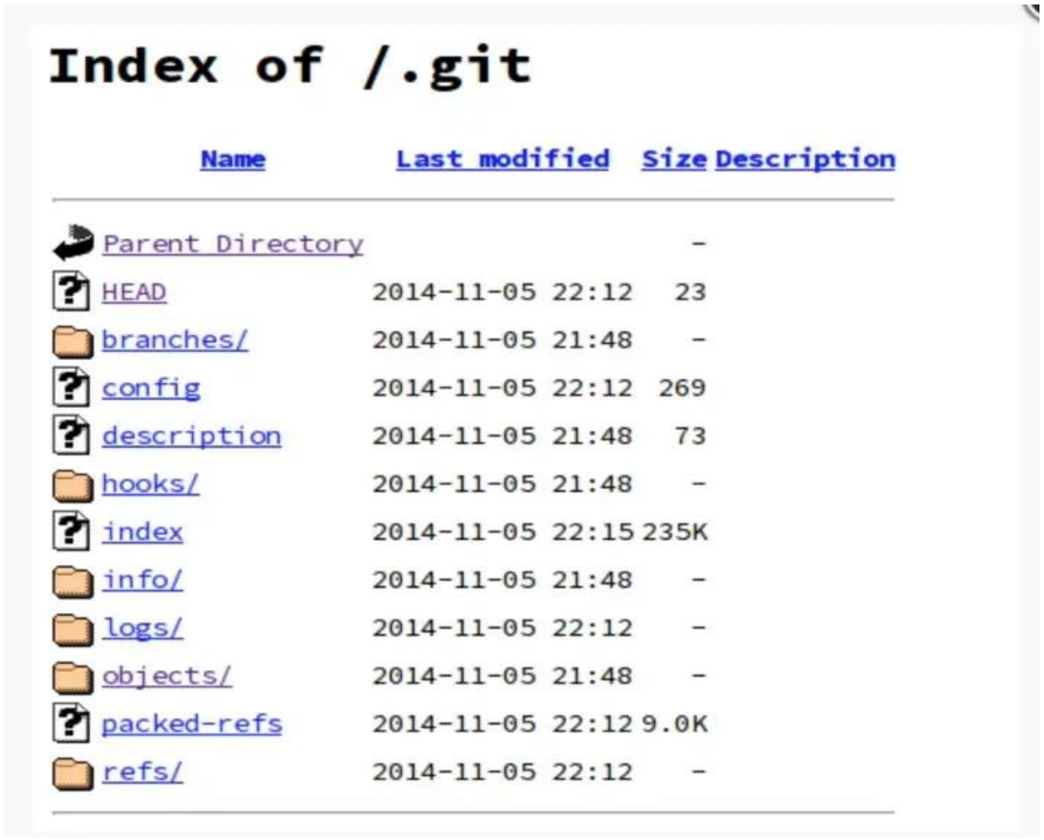
- **System Configuration**
    - Turn off directory listings
    - Ensure servers, frameworks and system components are running the latest approved version.
    - Ensure servers, frameworks and system components have all patches issued for the version in use
    - Remove all unnecessary functionality and files.
    - Restrict the web server, process and service accounts to the least privileges possible.
    - When exceptions occur, fail securely.

- **Database Security**
    - Use strongly typed parameterized queries.
    - Utilize input validation and output encoding and be sure to address meta characters. If these fail, do not run the database command.
    - Ensure that variables are strongly typed
    - Use secure credentials for database access
    - The application should use the lowest possible level of privilege when accessing the database.
    - Use stored procedures to abstract data access and allow for the removal of permissions to the base tables in the database.
    - Close the connection as soon as possible.
    - Remove or change all default database administrative passwords. Utilize strong passwords/phrases or implement multi-factor authentication.

    - Turn off all unnecessary database functionality
    - Disable any default accounts that are not required to support business requirements.

- **File Management**
    - Do not pass user supplied data directly to any dynamic include function.
    - Require authentication before allowing a file to be uploaded
    - Limit the type of files that can be uploaded to only those types that are needed for business purposes
    - Validate uploaded files are the expected type by checking file headers. Checking for file type by extension alone is not sufficient
    - Do not save files in the same web context as the application. Files should either go to the content server or in the database.
    - Prevent or restrict the uploading of any file that may be interpreted by the web server.
    - Turn off execution privileges on file upload directories
    - Do not pass user supplied data into a dynamic redirect. If this must be allowed, then the redirect should accept only validated, relative path URLs
    - Do not pass directory or file paths, use index values mapped to pre-defined list of paths
    - Never send the absolute file path to the client
    - Ensure application files and resources are read-only
    - Scan user uploaded files for viruses and malware

- **Using Github / code sharing / collaboration tools**
    - Dont mark your code public / don’t make critical repositories public.
    - Do not put config details in the source code.
    - Don’t commit dependencies into source control.
    - Don’t commit local configuration into source control.
    - Always create a meaningful git ignore file.



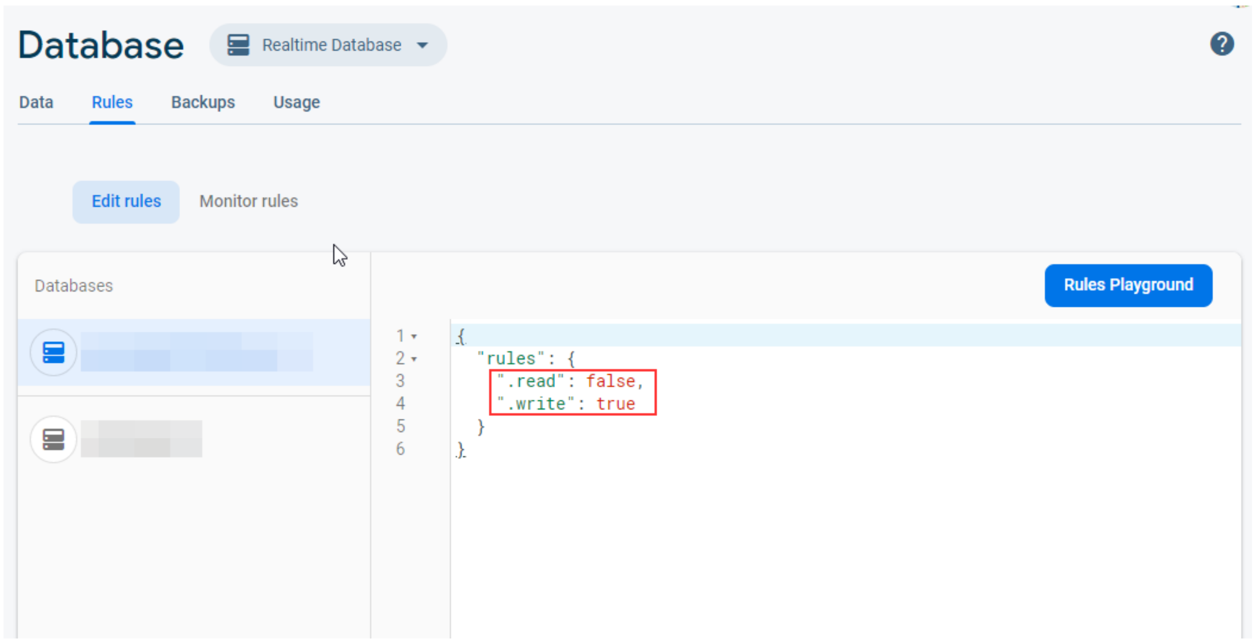
```
Index of /.git

      Name            Last modified     Size Description

  Parent Directory                        -
  HEAD               2014-11-05 22:12    23
  branches/          2014-11-05 21:48     -
  config             2014-11-05 22:12   269
  description        2014-11-05 21:48    73
  hooks/             2014-11-05 21:48     -
  index              2014-11-05 22:15 235K
  info/              2014-11-05 21:48     -
  logs/              2014-11-05 22:12     -
  objects/           2014-11-05 21:48     -
  packed-refs        2014-11-05 22:12 9.0K
  refs/              2014-11-05 22:12     -
```

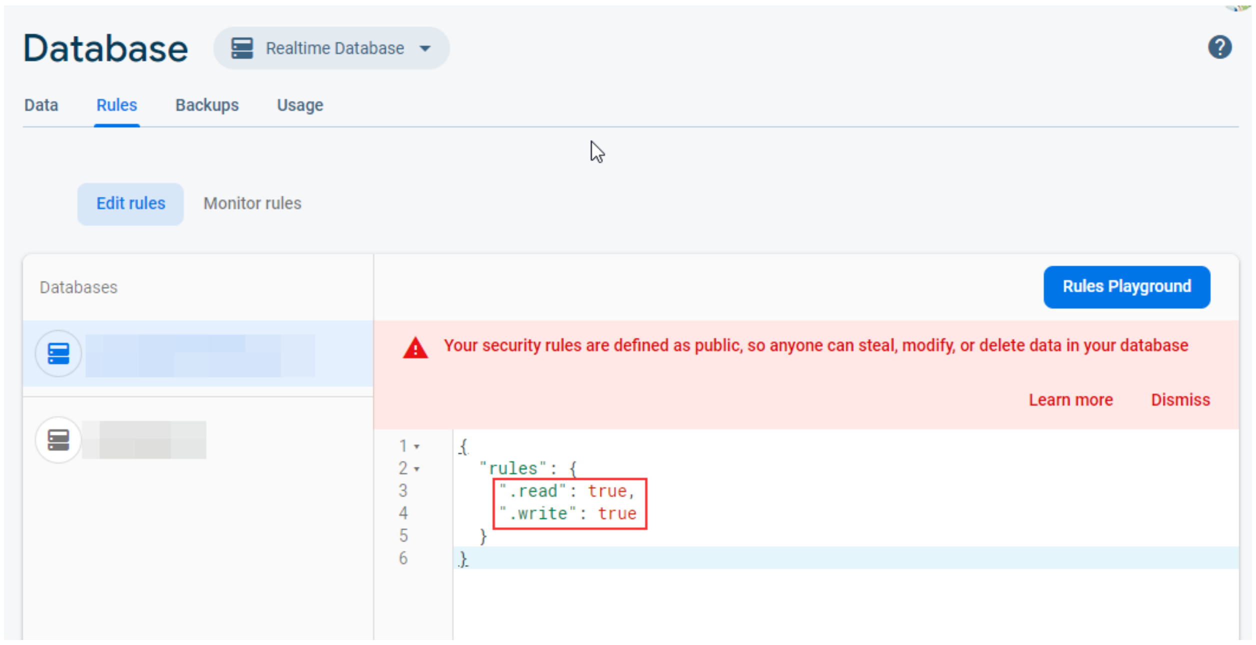
Note : Such misconfiguration results in complete source code disclosure of the application, and makes it quite easy for malicious actors to get a hold of your source code.

- **Firebase Misconfigurations**
  - **Properly Configured Firebase:** By default, the database settings for a database are read and write protected, meaning the read and write access is not allowed. To store real-time data, write access to the database must be enabled by setting the "write" flag to "true." Remember, that read access is still disabled, and the "read" flag is set to "false," as shown in the following screenshot.



```
{
  "rules": {
    ".read": false,
    ".write": true
  }
}
```

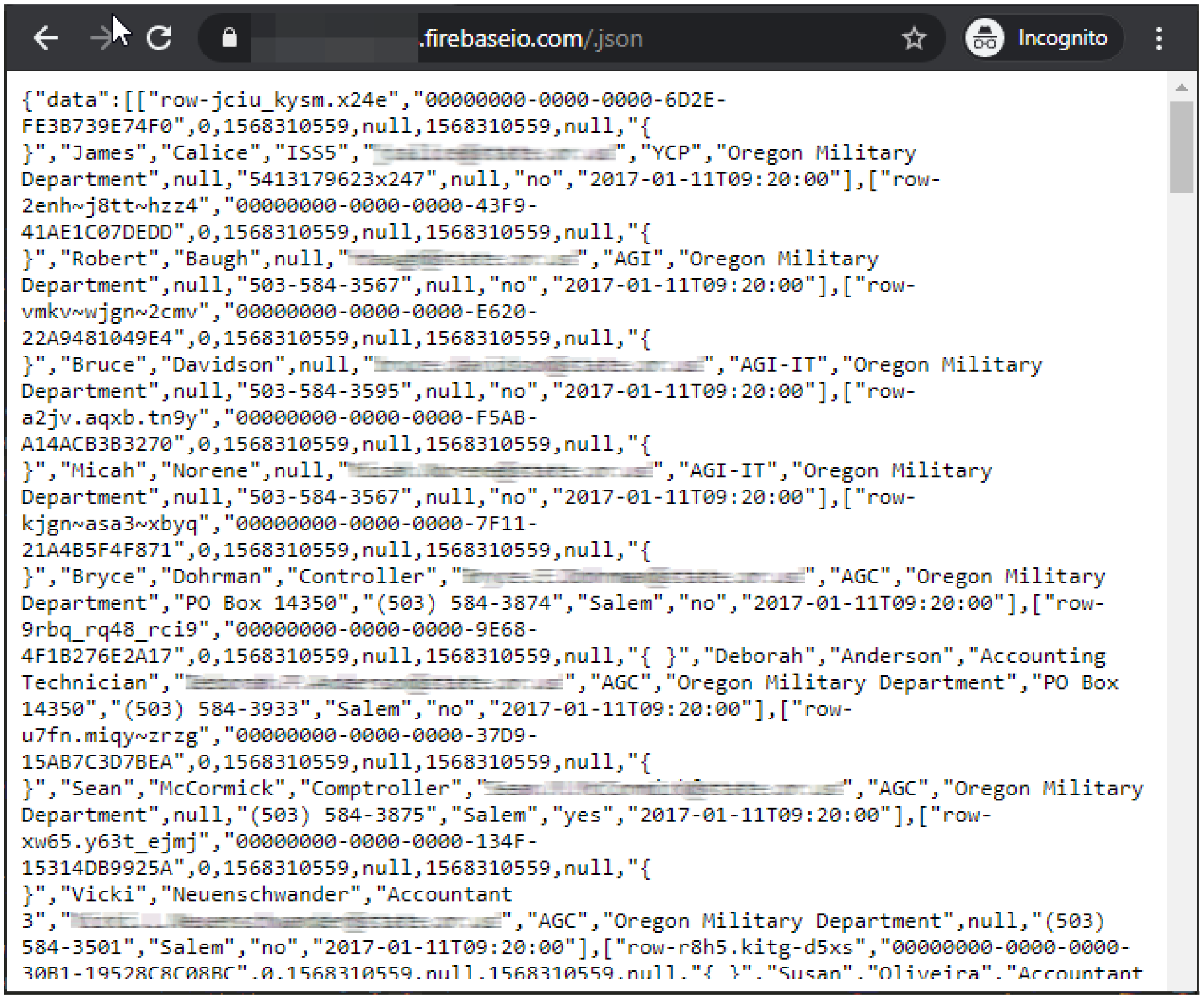
- **Misconfigured Firebase:** While testing the functioning of the database and checking the flow of data, developers often make the database public. They set the value of both "read" and "write" flag to "true." If the "read" flag is set to "true," it means that an adversary can read the content of the database without any authentication.



Your security rules are defined as public, so anyone can steal, modify, or delete data in your database

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Misconfigured Firebase exposing sensitive real time data

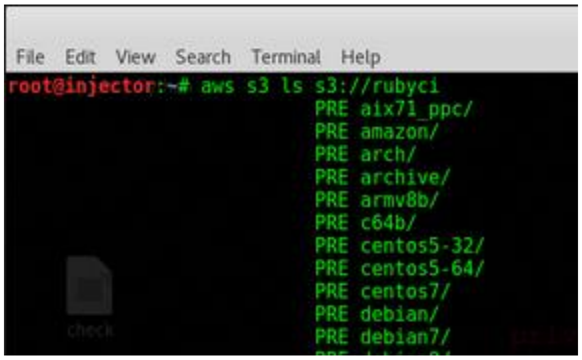

[redacted].firebaseio.com/.json

Incognito

```
{"data":[["row-jciu_kysm.x24e","00000000-0000-0000-6D2E-
FE3B739E74F0",0,1568310559,null,1568310559,null,"{
}","James","Calice","ISS5","[redacted]","YCP","Oregon Military
Department",null,"5413179623x247",null,"no","2017-01-11T09:20:00"],["row-
2enh~j8tt~hzz4","00000000-0000-0000-43F9-
41AE1C07DEDD",0,1568310559,null,1568310559,null,"{
}","Robert","Baugh",null,"[redacted]","AGI","Oregon Military
Department",null,"503-584-3567",null,"no","2017-01-11T09:20:00"],["row-
vmkv~wjgn~2cmv","00000000-0000-0000-E620-
22A9481049E4",0,1568310559,null,1568310559,null,"{
}","Bruce","Davidson",null,"[redacted]","AGI-IT","Oregon Military
Department",null,"503-584-3595",null,"no","2017-01-11T09:20:00"],["row-
a2jv.aqxb.tn9y","00000000-0000-0000-F5AB-
A14ACB3B3270",0,1568310559,null,1568310559,null,"{
}","Micah","Norene",null,"[redacted]","AGI-IT","Oregon Military
Department",null,"503-584-3567",null,"no","2017-01-11T09:20:00"],["row-
kjgn~asa3~xbyq","00000000-0000-0000-7F11-
21A4B5F4F871",0,1568310559,null,1568310559,null,"{
}","Bryce","Dohrman","Controller","[redacted]","AGC","Oregon Military
Department","PO Box 14350","(503) 584-3874","Salem","no","2017-01-11T09:20:00"],["row-
9rbq_rq48_rci9","00000000-0000-0000-9E68-
4F1B276E2A17",0,1568310559,null,1568310559,null,"{ }","Deborah","Anderson","Accounting
Technician","[redacted]","AGC","Oregon Military Department","PO Box
14350","(503) 584-3933","Salem","no","2017-01-11T09:20:00"],["row-
u7fn.miqy~zrzg","00000000-0000-0000-37D9-
15AB7C3D7BEA",0,1568310559,null,1568310559,null,"{
}","Sean","McCormick","Comptroller","[redacted]","AGC","Oregon Military
Department",null,"(503) 584-3875","Salem","yes","2017-01-11T09:20:00"],["row-
xw65.y63t_ejmj","00000000-0000-0000-134F-
15314DB9925A",0,1568310559,null,1568310559,null,"{
}","Vicki","Neuenschwander","Accountant
3","[redacted]","AGC","Oregon Military Department",null,"(503)
584-3501","Salem","no","2017-01-11T09:20:00"],["row-r8h5.kitg-d5xs","00000000-0000-0000-
30B1-19528C8C08BC",0,1568310559,null,1568310559,null,"{ }","Susan","Oliveira","Accountant
```

- **AWS Bucket Misconfiguration**
  - Block Public Access setting should be enabled

- S3 buckets should prohibit public read access
- S3 buckets should prohibit public write access
- S3 buckets should have server-side encryption enabled
- S3 buckets should require requests to use Secure Socket layer
- Amazon S3 permissions granted to other AWS accounts in bucket policies should be restricted
- S3 Block Public Access setting should be enabled at the bucket level



```
root@injector:~# aws s3 ls s3://rubyci
                           PRE aix71_ppc/
                           PRE amazon/
                           PRE arch/
                           PRE archive/
                           PRE armv8b/
                           PRE c64b/
                           PRE centos5-32/
                           PRE centos5-64/
                           PRE centos7/
                           PRE debian/
                           PRE debian7/
```