AppScan on Cloud
Scan Security Report

Name: audit-persistence_20200316_11_53_58

Created by:	HCL Application Security on Cloud, Version 8.0.1357
Scan name:	audit-persistence
Scan file name:	audit-persistence.irx
Scan started:	Monday, March 16, 2020 10:34:07 AM (UTC)
Operating system:	SAST

Summary of security issues

High severity issues:	3
Total security issues:	**3**

Summary

Total security issues: **3**

Fix Groups: **2**

	Number of Issues	
Fix Group #1: SslConfig.buildSslContextFromPem() (Validation.Required)	2	🟥🟥
Ungrouped File() (PathTraversal)	1	🟥

🟥 High 🟨 Medium ⬜ Low

Issue Types: **2**

	Number of Issues	
PathTraversal	1	🟥
Validation.Required	2	🟥🟥

🟪 Critical 🟥 High 🟨 Medium ⬜ Low ☐ Informational

Issues

Fix Group #1: SslConfig.buildSslContextFromPem()

This section groups 2 issues of type Validation.Required with significant commonality in the their traces.

These issues are grouped together to try to help you find a common fix that resolves them all.

Fix:

- com.bettercloud.vault.SslConfig.buildSslContextFromPem():SSLContext

This method is a part of the application code and appears in each of the grouped issue's traces. You should begin investigating a possible fix in the implementation of the method.

Alternate Fix Suggestions

- com.bettercloud.vault.SslConfig.buildSsl():void
- com.bettercloud.vault.SslConfig.inputStreamToUTF8(InputStream):String
- java.io.ByteArrayInputStream(byte[]):void
- java.lang.String.getBytes(String):byte[]
- java.security.cert.CertificateFactory.generateCertificate(InputStream):Certificate

These method calls are also common to the traces of the issues in this group. They represent other possible These method calls are also common to the traces of the issues in this group. They represent other possible locations to investigate a fix.

H Validation.Required

Issue 1 of 2

Issue ID:	27b6072d-4465-ea11-a94c-00155d550e89
Severity:	High
Status	New
Classification	Definitive
Fix Group ID:	5a73b037-ba58-ea11-a94c-00155d55408c
Location	java.security.cert.CertificateFactory.generateCertificate(InputStream):Certificate SslConfig:542
Source File	SslConfig
Line	542
Source File	SslConfig
Availability Impact	Partial
Confidentiality Impact	Partial
Integrity Impact	Partial
Date Created	Friday, March 13, 2020
Last Updated	Monday, March 16, 2020
CWE:	20
Source:	java.io.BufferedReader.readLine():String *via* SslConfig:619
Sink:	java.security.cert.CertificateFactory.generateCertificate(InputStream):Certificate *via* SslConfig:542

Issue 1 of 2 - Details

Trace

```
com.bettercloud.vault.SslConfig.build():SslConfig
    com.bettercloud.vault.SslConfig:457
    inputStreamToUTF8(input)
        com.bettercloud.vault.SslConfig:619
        str = in.readLine()
        com.bettercloud.vault.SslConfig:622
        utf8 = utf8.append(str)
        com.bettercloud.vault.SslConfig:625
        return utf8.toString()
    com.bettercloud.vault.SslConfig:462
    this.buildSsl()
        com.bettercloud.vault.SslConfig:483
        this.buildSslContextFromPem()
            com.bettercloud.vault.SslConfig:541
            this.pemUTF8.getBytes("UTF-8")
            com.bettercloud.vault.SslConfig:541
            pem = new ByteArrayInputStream(this.pemUTF8.getBytes("UTF-8"))
            com.bettercloud.vault.SslConfig:542
            certificate = certificateFactory.generateCertificate(pem)
```

Issue 2 of 2

Issue ID:	28b6072d-4465-ea11-a94c-00155d550e89
Severity:	High
Status	New
Classification	Definitive
Fix Group ID:	5a73b037-ba58-ea11-a94c-00155d55408c
Location	java.security.cert.CertificateFactory.generateCertificate(InputStream):Certificate SslConfig:542
Source File	SslConfig
Line	542
Source File	SslConfig
Availability Impact	Partial
Confidentiality Impact	Partial
Integrity Impact	Partial
Date Created	Friday, March 13, 2020
Last Updated	Monday, March 16, 2020
CWE:	20
Source:	java.nio.file.Files.readAllBytes(Path):byte[] *via* EnvironmentLoader:26
Sink:	java.security.cert.CertificateFactory.generateCertificate(InputStream):Certificate *via* SslConfig:542

Issue 2 of 2 - Details

Trace

```
com.bettercloud.vault.SslConfig.build():SslConfig
    com.bettercloud.vault.SslConfig:455
    this.environmentLoader.loadVariable("VAULT_SSL_CERT")
        com.bettercloud.vault.EnvironmentLoader:26
        bytes = readAllBytes(get(getProperty("user.home"), new String()[][0]).resolve(".vault-token"))
        com.bettercloud.vault.EnvironmentLoader:28
        new String(bytes, "UTF-8")
        com.bettercloud.vault.EnvironmentLoader:28
        value = new String().trim()
    com.bettercloud.vault.SslConfig:455
    pemFile = new File(this.environmentLoader.loadVariable("VAULT_SSL_CERT"))
    com.bettercloud.vault.SslConfig:456
    input = new FileInputStream(pemFile)
    com.bettercloud.vault.SslConfig:457
    inputStreamToUTF8(input)
    com.bettercloud.vault.SslConfig:462
    this.buildSsl()
        com.bettercloud.vault.SslConfig:483
        this.buildSslContextFromPem()
            com.bettercloud.vault.SslConfig:541
            this.pemUTF8.getBytes("UTF-8")
            com.bettercloud.vault.SslConfig:541
            pem = new ByteArrayInputStream(this.pemUTF8.getBytes("UTF-8"))
            com.bettercloud.vault.SslConfig:542
            certificate = certificateFactory.generateCertificate(pem)
```

Ungrouped File()

This section lists the remaining 1 issues that could not be included in any other fix groups.

H PathTraversal

Issue 1 of 1

Issue ID:	26b6072d-4465-ea11-a94c-00155d550e89
Severity:	High
Status	New
Classification	Definitive
Fix Group ID:	5b73b037-ba58-ea11-a94c-00155d55408c
Location	java.io.File(String):void SslConfig:455
Source File	SslConfig
Line	455
Source File	SslConfig
Availability Impact	Partial
Confidentiality Impact	Partial
Integrity Impact	Partial
Date Created	Friday, March 13, 2020
Last Updated	Monday, March 16, 2020
CWE:	73
Source:	java.nio.file.Files.readAllBytes(Path):byte[] *via* EnvironmentLoader:26
Sink:	java.io.File(String):void *via* SslConfig:455

Issue 1 of 1 - Details

Trace

```
com.bettercloud.vault.SslConfig.build():SslConfig
    com.bettercloud.vault.SslConfig:455
    this.environmentLoader.loadVariable("VAULT_SSL_CERT")
        com.bettercloud.vault.EnvironmentLoader:26
        bytes = readAllBytes(get(getProperty("user.home"), new String()[][0]).resolve(".vault-token"))
        com.bettercloud.vault.EnvironmentLoader:28
        new String(bytes, "UTF-8")
        com.bettercloud.vault.EnvironmentLoader:28
        value = new String().trim()
    com.bettercloud.vault.SslConfig:455
    pemFile = new File(this.environmentLoader.loadVariable("VAULT_SSL_CERT"))
```

Advisories

H SAST: PathTraversal

Description

This API accepts a directory, a filename, or both. If user supplied data is used to create the file path, the path can be manipulated to point to directories and files which should not be allowed access or which may contain malicious data or code.

Path traversal attacks usually involve modifying the path to a file and/or the file name itself. Consider the following PHP code:

```
<?php
    $file_data = file("/Users/{$_POST['UserName']}.php");
?>
```

This code attempts to open a file based on a path created from user supplied data. This data could contain text such as "../" to manipulate the final path to point to a totally different path higher up in the directory tree. In addition, on some platforms, you can add a final string terminating null character (%00) to the end of the user supplied string to truncate the file extension (in this example, the extension is .data). If an attacker entered the following for the UserName POST parameter:

```
http://www.example.com/MyPage.php?UserName=../../../../../../../../../../etc/passwd%00
```

They could possibly access the password file on this computer.

This is especially dangerous when file contents are written to the browser in the HTTP response message. In this case an attacker could attempt to inject a Cross-Site Scripting (XSS) attack into the file whose contents will be written to the browser. When a legitimate user views this file's data in their browser, the injected script will be executed on their computer.

If a user supplied path is passed as an argument to an API that deletes a file, an attacker may try to remove critical files used by the operating system to function correctly. This effectively causes a Denial-of-Service (DoS) attack. Alternately, an attacker may try to delete files that are critical for other users to be able to login and use this web application. This effectively causes a localized DoS attack against one or more users of this web application. If an attacker wants to cover their tracks, this attack vector can also be used to delete log files containing information that could be used to track down an attacker.

If a user supplied path is passed as an argument to an API that writes data to a file, an attacker may try to corrupt a file to cause a DoS attack or add information to a file. By adding massive amounts of information to a file, such as a log file, the attacker may be able to hide their tracks within the sheer amount of information in the log file or by having their tracks removed after the log file is automatically truncated when it grows beyond a specified size.

Allowing users to create files on the server based on their user input can be exploited by an attacker. All the attacker has to do is simply cause files (large or small) to be created on the server in significant number as to use up all the free disk space. This will not only slow the server to a crawl, but also prevent other legitimate users from being able to use the website.

Mitigation:

To prevent this type of attack you should first sanitize all user supplied data that will be used to build the path. The best way to do this is to use a whitelist. A whitelist is an accepted list of values that the application will accept. For example, the whitelist for user supplied data to be inserted into a path string would be only alpha-numeric characters with possibly the underscore (_) and dash (-) characters allowed.

In addition to using the whitelisting technique, the server should disallow access to all areas of the file system except those used specifically for the web application. This will prevent users from attempting to break out of the web application directory and access files in other areas of the file system.

Go to Table of Contents

H SAST: Validation.Required

Description

Input validation is necessary to ensure the integrity of the dynamic data of the application. Validation is useful to protect against cross-site scripting, SQL and command injection, and corrupt application data fields. Even if there are no directly vulnerable uses of a piece of data inside one application, data that is being passed to other applications should be validated to ensure that those applications are not given bad data. Validation, especially for size and metacharacters that might cause string expansion, is even more important when dealing with fixed size, overflowable buffers.

You should validate input from untrusted sources before using it. The untrusted data sources can be HTTP requests or other network traffic, 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 transfer information from the browser to the server side application.

Attackers use unvalidated parameters to target the application's security mechanisms such as authentication and authorization or business logic, and as the primary vector for exercising many other kinds of error, including buffer overflows. If the unvalidated parameters are stored in log files, used in dynamically generated database queries or shell commands, and/or stored in database tables, attackers may also target the server operating system, a database, back-end processing systems, or even log viewing tools.

For example, if the application looks up products from the database using an unvalidated productID from HTTP request. This productID can be manipulated using readily available tools to submit SQL injection attacks to the backend database.

Example *Java*

```
final String productID = request.getParameter( "productID" );
final String sql = "Select * from Product where productID = '" + productID + "'";
final Statement statement = connection.createStatement();
final boolean rsReturned = statement.execute(sql);
```

Example *C++*

```
char productID[28];
fscanf(fd, "%28s", productID);
char sql[71] = "Select * from Product where productID = '";
strncat(sql, productID, 28);
strncat(sql, "'", 1);
SQLPrepare(handle, sql, 71);
SQLRETURN ret = SQLExecute(handle);
```

Note that using dynamically generated SQL queries is another bad practice. Refer to vulnerability type for more detail.

Example *Java*

```
// This class would simply associate parameter names with a data type, plus bounds for
// numeric data or a regular expression for text.
Validator validator = Validator.getInstance( this.getServletContext );
Boolean valid = false;
try
{
    validator.validate( request );
    valid = true;
}
catch ( ValidationException e )
{
    request.getSession().invalidate();
    out.println("Invalid HTTP request");
    out.close();
}
if ( valid )
{
    final String productId = request.getParameter( "productID" );
    final String sql = "Select * from Product where productID= ?");
    final PreparedStatement ps = con.prepareStatement(sql);
    ps.setString(1,productID);
    ps.execute();
}
```

Example *C++*

```
char productID[28];
fscanf(fd, "%.28s", productID);
regex_t * productIDValidator;
regcomp(productIDValidator, "[^a-z]*", REG_EXTENDED);
int matchCount;
regmatch_t * matches;
regexec(productIDValidator, productID, matches, matchCount, 0);
if(0 == matches)
{
    char sql[70] = "Select * from Product Where productID = '?'";
    int sqlLen = 70;
    SQLPrepare(handle, sql, 70);
    SQLBindParameter(handle, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 28, 0, productID, *sqlLe
n)
    SQLRETURN ret = SQLExecute(handle);
}
else
{
    HandleBadProductID();
}
```

In this instance, a regular expression constrained input to a known set of characters, through a whitelist (rejecting inputs containing anything not in that set), and of a known and limited length. Using a whitelist instead of a blacklist is important, because it can be difficult to anticipate which characters (especially when Unicode is involved) may cause problems, while it is normally easy to determine which characters are legal in a given input field.

Mitigation:

The primary recommendation is to validate each input value against a detailed specification of the expectation for that value. This specification should detail characteristics like the character set allowed, length, whether to allow null, minimum value, maximum value, enumerated values, or a specific regular expression. For example, make sure all email fields have the same format. Also, limit name fields and other text fields to an appropriate character set, no special characters, and with expected minimum and maximum sizes. A input pattern violation can be evidence of an attack and should be logged and responded to appropriately.

There are several possible approaches to input validation in an application. The recommendation is to implement the features in a single component that is invoked in a central location. If this is not possible, then enforce a strong policy for the use of a common set of input validation functions.

Go to Table of Contents