Draft[1]

How Does CVE-2023-24998 Impact ESAPI?

Kevin W. Wall <kevin.w.wall@gmail.com>

Summary

Category:	Apache Commons FileUpload before 1.5 does not limit the number of request parts to be processed resulting in the possibility of an attacker triggering a Denial of Service (DoS) with a malicious upload or series of uploads. Note that, like all of the file upload limits, the new configuration option (FileUploadBase#setFileCountMax) is not enabled by default and must be explicitly configured.
Module:	Apache Commons FileUpload – a direct compile-time dependency used by ESAPI to support “file uploading” over HTTP.
Announced:	In this security bulletin.
Credits:	GitHub Dependabot, Snyk, Martin Bektchiev
Affects:	All versions of ESAPI 2.x prior to 2.5.2.0 and all versions of ESAPI 1.x.
Details:	**Exploitable as used by ESAPI, but see discussion below for important details.**
GitHub Issue #:	None.
Related:	CVE-2023-ABCDE (TBD); no other ESAPI security bulletins.
CWE:	CWE-770 (Allocation of Resources Without Limits or Throttling)
CVE Identifier:	CVE-2023-24998
CVSS Severity (version 3.1)	CVSS v3.1 Base Score: 7.5 (High) Impact Subscore: 3.6 Exploitability Subscore: 3.9 CVSS Vector CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

1 We will remove this DRAFT notice and finalize this once we get a new CVE ID issued for ESAPI related to CVE-2023-24998. That may not be available until after the ESAPI 2.5.2.0 release is made available, which is fine as links to the security bulletins are not release (i.e., tag or branch) specific.

Background

OWASP ESAPI (the OWASP Enterprise Security API) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications. The ESAPI for Java library is designed to make it easier for programmers to retrofit security into existing applications. ESAPI for Java also serves as a solid foundation for new development.

One of the security controls provided by ESAPI for Java is its provision for (relatively) "safe file uploads". ESAPI leverages the functionality of Apache Commons FileUpload to achieve this functionality.

Problem Description

According to the description in NIST's National Vulnerability Database (NVD), the current description for CVE-2023-24998 states:

> "Apache Commons FileUpload before 1.5 does not limit the number of request parts to be processed resulting in the possibility of an attacker triggering a DoS with a malicious upload ***or series of uploads***. Note that, like all of the file upload limits, the new configuration option (FileUploadBase#setFileCountMax) is not enabled by default and must be explicitly configured."

First, after analyzing the Apache Commons FileUpload code and its (attempted) remediation, it is my understanding that this will ***NOT*** protect you against a "series of uploads". Unless I misread something in their code (possible, as it was very late and I was very tired), the file size limit applies on a per HTTP request. More details about that are provided in the Impact section below.

For ESAPI though, the $64,000 question that everyone is asking is "will using ESAPI leave my application code exposed to CVE-2023-24998 in a manner that makes this CVE exploitable?" That is the question this analysis attempts to answer, but the TL;DR answer for those of you not interested in the details is, "**Yes**".

ESAPI only uses Apache Commons FileUpload in all the HTTPUtilities.getFileUploads methods and in the ESAPI WAF class (ESAPIWebApplicationFirewallFilter), specifically via an implementation class that the ESAPI WAF uses (InterceptingHTTPServletRequest). **Thus, if you are not using either of those 2 ESAPI features, you are not even interacting with Apache Commons FileUpload at all and thus *NOT* affected. If you are *only* using the ESAPI WAF, you are also not affected because the WAF component doesn't actually attempt to upload any files. However, if you *are* using any of the HTTPUtilities.getFileUploads methods, you potentially are affected.**

The (original[2]) description for CVE-2023-24998 implies that the "problem" was with the FileUploadBase class. Indeed, a cursory examination of this class between the 1.4 and 1.5 releases shows that the 1.5 version added a counter (fileCountMax) for the maximum allowed files per single HTTP request and if that counter is exceeded, a FileCountLimitExceededException is thrown. There were also setter / getter methods for that counter added and as noted in the CVE description one must explicitly call FileUploadBase.setFileCountMax(long) before doing the uploads, otherwise no upper limit is enforced. But what is easy to miss is that FileUploadBase is an *abstract* class which in turn implies that other classes must be impacted as well. In other words, one cannot simply recursively grep one's source code for FileUploadBase to see if you are using it and if not, conclude that your code is not affected. Instead, you must also look for all these other Apache Commons FileUpload classes that are subclasses of FileUploadBase and check to see if you are using them as well. As it turns out, the Apache Commons FileUpload classes that extend FileUploadBase are:

- org.apache.commons.fileupload.FileUpload
- org.apache.commons.fileupload.DiskFileUpload
- org.apache.commons.fileupload.portlet.PortletFileUpload
- org.apache.commons.fileupload.servlet.ServletFileUpload

So, minimally one needs to search for the use of any of these subclasses as well as the abstract FileUploadBase class.

As it turns out, ESAPI uses the ServletFileUpload class, which makes the DoS vulnerability described in CVE-2023-24998, exploitable in ESAPI.

Thus, our conclusion is, this CVE *is* exploitable via ESAPI. Because of this, the ESAPI team is working with GitHub Security team to file a CVE specific to ESAPI. That will allow us to make the CVE description specific to the affected classes in ESAPI.

Related Concerns

Please see CVE-2023-ABCDE (TBD) for additional details.

Impact

The underlying problem is that an attacker can keep uploading so many new uniquely named files that eventually your system resources (inodes for *nix file systems) become exhausted and no more files will be able to be able to be created on that file system. Hence the DoS aspect.

2 By the time you read this, hopefully the original CVE description will have been updated as per my request to note all the affected Apache Commons FileUpload classes rather than only the single abstract class.

If your application is *not* using one of the HTTPUtilities.getFileUploads methods, then your application is ***not*** impacted by the DoS vulnerability in CVE-2023-24998. If that is the case, Software Composition Analysis (SCA) tools and/or services like OWASP Dependency Check, BlackDuck, Snyk, Veracode's SourceClear, GitHub Dependabot, etc. will still continue to give you warnings that you may be required to explain to your management in order to justify continued use of ESAPI in your application. And while we would like you to update to 2.5.2.0 or whatever the latest, you can use the Workarounds section to quiet your SCA tools and point your management to this Security Bulletin as justification for not upgrading to 2.5.2.0 or later.

However, if your applications ***are*** using one of the HTTPUtilities.getFileUploads methods, then you might[3] be subject to a DoS attack. Specifically, a successful attack could result in an attacker creating files on your file system that you are using for uploading files until no more additional files can be created in that file system. (That is, on *nix systems, this is a DoS attack against inodes on a file system or possibly against disk quota restrictions.)

I believe that Apache Commons Files should have addressed this CVE so that it was secure-by-default so that it would have been sufficient to simply upgrade to version 1.5 or later. The library maintainers could have set the default value by picking up the value from a system property (e.g., “apache.fileupload.filecountmax”) and still allow explicit calls to FileUploadBase.setFileCountMax(long) to override the system property. If no system property or call to that new method were not made, only then would it default to not restricting the maximum number of files uploaded per HTTP request.

Because Apache did not choose a secure-by-default approach, I believe that the Apache Commons FileUpload patch for CVE-2023-24998.

If you wish to support file uploads at all and implement it so that it is mostly immune to DoS attacks caused by repeated uploading of numerous files, then it is not *sufficient* to simply add a mechanism that imposes a limit on the number of uploaded files *per HTTP request*. That upper bound must apply per *authenticated* user, not per *anonymous* HTTP request attempt so abuse of the file upload feature can more easily be tracked, as well as raising the complexity of the attack. If you want to address that, I suggest using a separate, independent WAF (the poor man’s ESAPI WAF does not attempt to address DoS type attacks via request throttling, etc) or perhaps a RASP solution similar to the OWASP AppSensor Project. A RASP approach, such as AppSensor uses, could be configured (or extended, if the behavior is already be supported within the existing framework) to impose limits on a maximum limits on both the number of files and the number of bytes on per authenticated user basis. Alternately, using an external WAF approach—even Apache mod_security and the OWASP Core Rules Set probably is sufficient—can provide an element of protection, even if all it is able to do is to be configured to provide rate limiting on per IP address basis.

3 “Might” because your application code or your environment may have other mitigating controls that prevent it.

For a determined attacker who wishes to cause a DoS attack by making your server run out of inodes or run out of file system space, placing restrictions only on a per HTTP request basis rather than taking a more holistic approach similar to the description in the previous paragraph is not going to do anything but slow your attackers down a bit. However, placing the defense at the level of per HTTP request might even make the situation worse. If you have (say) only 20k inodes available on your file system and you allow only 20 files to be uploaded per HTTP request, a determined DoS attacker cares little if they can use all those inodes with a single HTTP request or if they need to make 1000 HTTP requests to do so. In fact, if instead of a single attack of uploading 20k files in an single HTTP request, if instead, they hit you with 20k separate HTTP requests each uploading a single file as fast as they can send them, that may be worse because it may spike your server's CPU resources thus causing other temporary outages in your application.

So, let's not pretend this is a real solution. It is but a Band-Aid on a potentially gushing artery. Any real solution to prevent DoS attacks almost certainly is more complicated than something that can be placed in some general library. In theory, a specialized library could prevent this, but it would have to ensure more restrictions, such as ensuring that a user attempting file uploads was authenticated and perhaps a stateful check against that user's prior (or at least recent) file upload history for a given application. But this is seldom done in general libraries such as Apache Commons FileUpload or OWASP ESAPI as that would make that provided file upload functionality much less useful to the others who had a legitimate use case for anonymous file uploads or who don't wish to consider having the libraries prevent DoS attacks against because that is already covered in their threat model by some external device such as a commercial WAF.

Workaround

If you must use an ESAPI release prior to 2.5.2.0 and you are not using any of the HTTPUtilities.getFileUploads methods, then you can use the following (or similar) workaround when building your project to keep your SCA tools from complaining about this CVE.

If you are using Maven, in your application's pom.xml, reference your dependency on the ESAPI jar in this manner, to exclude any of Apache Commons FileUpload:

```
<dependency>
 <groupId>org.owasp.esapi</groupId>
 <artifactId>esapi</artifactId>
 <version>2.5.1.0</version>  <!-- Or whatever ESAPI version you are using. -->
 <exclusions>
     <exclusion>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
     </exclusion>
 </exclusions>
```

```
    </dependency>
```

(Note: To be completely safe, you would have to exclude this as a transitive dependency from any of your direct dependencies that may pull it in as well.)

Then, when you then build your project, this should exclude the Apache Commons FileUpload jar from your application's classpath. (Note that you do not have to specify a 'version' where you are *excluding* a jar.)

You can also exclude specific transitive dependencies using Gradle. If you use Gradle, follow these general instructions.

Solution

This section describes the high level changes made to ESAPI in version 2.5.2.0 to address CVE-2023-24998. Note that more perhaps could be done but it may impact backward compatibility with previous versions of ESAPI and we do not believe this issue is actually serious enough (despite its CVSSv3 score) to warrant breaking client code.

Some New ESAPI Properties

Several new lines were added to ESAPI's "configuration/esapi/ESAPI.properties" file to reference 2 new properties. That updated ESAPI.properties file with the new property setting will be in the separately downloadable **esapi-2.5.2.0-configuration.jar** file in our GitHub repository under the Releases section for release 2.5.2.0. (It will also be available in subsequent ESAPI releases.)

The 3 lines that were added for the first new property to the ESAPI.properties file are:

```
# Maximum # of files that can be uploaded per HTTP request.
# Set to -1 for no maximum. Related to CVE-2023-24998.
HttpUtilities.MaxUploadFileCount=20
```

This new property is used as the default value to call ServletUpload.setFileCountMax(long) in the DefaultHTTPUtilities.getFileUploads(HttpServletRequest, File, List) method, which is the specific method that does the actual file upload. If this property is not found in your ESAPI.properties file, then a hard-coded default of 20 is used. Why 20? Because it seemed like a reasonable upper bound that we might expect human users to upload on any single HTTP request. If you don't like the value we chose, then tweak the property and set it to whatever positive integer that you would like. Setting it to '-1' (or any other negative integer for that matter) should cause it to act as though no upper bound is present and a value of '0' ought to prevent any file uploads at all (which would be a rather curious choice).

In addition, this property has been added to allow you to deny file upload access to anonymous users:

```
# Allowing anonymous users to do file uploads via HTTPUtilities.getFileUploads
# can make it easier for DoS attacks via uploading files easier. (See Security Bulletin #11,
# https://github.com/ESAPI/esapi-java-legacy/blob/develop/documentation/ESAPI-security-
bulletin11.pdf
# for details).
#
# By default, we allow anonymous users to upload files because we can only rely on
# ESAPI.authenticator().getCurrentUser() to determine if a user associated
# with the current HTTP session is authenticated and almost no one uses the
# ESAPI Authenticator because the reference implementation is just a toy
# implementation and is not enterprise scalable.
#
# If you are using the ESAPI Authenticator (the ESAPI reference implementation
# or you've implemented your own custom one), then you can set this property value
# to 'false' to disallow anonymous (i.e., unauthenticated) users to upload
# files. However, if you are not using the ESAPI Authenticator, then you should
# probably leave this set to 'false', otherwise you will completely prevent the
# use of HTTPUtilities.getFileUploads methods.
#
HttpUtilities.FileUploadAllowAnonymousUser=true
```

The comments accompanying this new property are intended to be self-explanatory.

Relavent ESAPI Code Changes

A static initializer block was added to the DefaultHTTPUtilities implementation class. That static initializer block is responsible for setting values associated with the new properties mentioned in the previous section. If those properties cannot be found, the same value as show above is used.

Those static initializers could fail if ESAPI is unable to locate your ESAPI.properties file, but it that can't be found, you are likely to have troubles elsewhere. If the ESAPI.properties file is found and has a valid integer value for the property HttpUtilities.MaxUploadFileCount, then that value is used to set the file count maximum for Apache Commons FileUpload.

If you have the property HttpUtilities.FileUploadAllowAnonymousUser to 'false' and ESAPI.authenticator().getCurrentUser() returns 'null', then the unchecked exception, java.security.AccessControlException is thrown when any of the HTTPUtilties.getFileUploads methods are called. (Note: Do not confuse that with ESAPI's org.owasp.esapi.errors.AccessControlException, which I would have used but it is a checked exception.[4] Using a checked exception here would force any ESAPI client using any of the HTTPUtilties.getFileUploads methods to change their code, making them either catch the exception or change the signature of their calling function's declaration thus breaking backward compatibility. Thus we had to go with an unchecked exception type.)

4 In this case, I think the JDK got it right and ESAPI got it wrong. Access control violations should be an unchecked RuntimeException since it rarely (if ever) can be verified at compile-time and in some cases, It can be remedied by forcing anonymous users to got back and authenticate and try again or in other cases, perhaps change the current effective role that they currently are using. But I digress.

As for the ESAPI WAF's InterceptingHTTPServletRequest class, no code change was required there because it does not call the method ServletFileUpload.parseRequest. It is in the method FileUploadBase.parseRequest where the limits on the file size and file count are imposed since that is the method that handles the actual file uploads. Thus if the parseRequest method is not called, as it isn't for the ESAPI WAF, no code changes are needed.

Acknowledgments

CVE notification: GitHub Dependabot, Snyk, Martin Bektchiev
Reviewers: Matt Seil, Jeremiah Stacey, Arshan Dabirsiaghi, Dave Wichers, Martin Bektchiev

References

https://nvd.nist.gov/vuln/detail/CVE-2023-24998

https://github.com/apache/commons-fileupload/tree/commons-fileupload-1.5