Third-Party Library Assessment – Aspose Words

April 9, 2020

ISE is a security consultancy based in Baltimore, Maryland. One of ISE's clients requested that ISE perform a vulnerability analysis on the Aspose.Words Java library version 20.3 before they integrated it into their application. ISE discovered a number of issues, some of which are directly exploitable and some of which are "strategic weaknesses", which we consider issues that lack direct exploitability but nonetheless reduce the security posture of a product (or in this case, the users of a product). ISE recommends that Aspose take action to remediate these issues and subsequently engage in coordinated disclosure with the wider security community.

Vulnerabilities

ISE considers security issues with direct exploitability to be "Vulnerabilities"; these issues are typically of higher severity than non-exploitable ones and are typically prioritized for remediation. The following section details vulnerabilities ISE discovered while evaluating the Aspose.Words Java library.

Credential Disclosure Via Linked Images

Severity: High

On Windows-based hosts, documents containing references to resources using UNC paths (e.g., `\\example.com\a\b`) will be processed by default. In a domain environment, this will cause the host to send its domain credentials in a hashed format to the specified server. If an attacker is able to convince a user or server to process a document with such a resource link pointing to a host they control, the attacker will receive the user or service account's credentials in NTLM hash format, which can then be reused in a classic pass-the-hash attack[1] to allow the attacker to gain access to any resource as the victim user or service account. If the account in question uses a weak or guessable password, the attacker could additionally perform a password cracking attack to recover the account password for further malicious use.

Attack: Credential Disclosure

In a scenario such as a web service where a Windows host processes documents with attacker-controlled content, an attacker could submit a document containing a reference to a UNC path (encoded as necessary to be a valid URI, such as `file://///example.com/a/b`) to be processed. The victim user or service would process the document, and the Aspose library would subsequently ask the OS to load the resource, initiating an SMB session to the target host and leaking the service account's username and NTLM password hash as a result. The attacker could then leverage this in a pass-the-hash attack and/or attempt to crack the hash.

Recommendation: Disable Remote Loading Functionality By Default

ISE recommends that Aspose disable loading of non-embedded images in all document types by default; users should have to explicitly specify that they wish to allow remote images to be loaded using e.g., an additional parameter in the `LoadOptions` class. This option should be documented and have warnings about the dangers of allowing remote or locally referenced images and contain information about using the `IResourceLoadingCallback` interface to more granularly restrict the acceptable reference locations if necessary.

In general, software (including software libraries) should use secure settings by default as the overwhelming majority of users will leave the software in its default state. Many if not most users of the Aspose product will not require remote

[1] https://en.wikipedia.org/wiki/Pass_the_hash

or referenced image support, and those users should be protected by default rather than needing to explicitly secure their application.

Additionally, the documentation for the `IResourceLoadingCallback` interface incorrectly implies that the interface will not be used when loading referenced resources from DOC(X) files. The documentation located at the following location:

- https://apireference.aspose.com/java/words/com.aspose.words/iresourceloadingcallback#resourceLoading(com.aspose.words.ResourceLoadingArgs)

As of this writing, it states the following:

Implement this interface if you want to control how Aspose.Words loads external resource when importing a document from HTML or MHTML.

This is incomplete; In ISE's experiments, it appears as though the interface will be used for external resources in any file type. Aspose should double-check the documentation and ensure it matches the actual behavior of their library.

Recommendation: Improve Documentation

ISE recommends that Aspose improve the documentation of the Aspose.Words library to include additional details about the risks of allowing remote image loading, including recommendations for disabling the functionality or filtering the URLs that the library will load.

Local Image Disclosure Via Linked Images

Severity: Medium

Similar to the previous finding, processing a document with a reference to a local image file will result in that file being included in the final document which can lead to sensitive information disclosure.

Attack: Local Image Disclosure

In a web service scenario, an attacker could upload a maliciously crafted document containing references to known or guessed images on the server's filesystem, which would be included in the resulting PDF. This could result in disclosure of sensitive or private data such as content uploaded by other users.

Recommendation: Disable Remote Loading Functionality By Default

ISE recommends that Aspose disable remote loading functionality by default as described in the previous issue 'Credential Disclosure Via Linked Images'.

Lack of Sanity Checking of Embedded Images

Severity: Medium

An attacker could upload a document that either referenced or included extremely large images ("decompression bombs"). When processing these images, the library will consume extremely large amounts of memory and CPU time. The library by default uses the Java image processing APIs, which have relatively low limits on image size; as a result, this attack is less impactful than it might otherwise be. The largest image size that the library accepts using default JVM settings is approximately 10 thousand pixels by 10 thousand pixels, resulting in approximately 500MB of memory used.

Attack: Denial of Service / Resource Exhaustion

An attacker could upload a malicious document for processing containing one or more decompression bombs, such as the ones available for testing purposes at https://bomb.codes/bombs. The service would consume large amounts of resources in doing so, potentially crashing or impacting the availability of the service for other users or resulting in increased compute costs for the host. In ISE's experiments, processing a document containing the 30,000 pixel by 30,000 pixel decompression bomb consumed a peak of 500MB of memory and about 30 seconds of CPU time on a modern, high speed CPU. An attacker could upload a document containing a large number of such images many times at once to a server in order to consume all of its available memory and CPU time in a denial-of-service attack.

Recommendation: Set Sane Image Size Limits

ISE recommends that Aspose by default set sane image size limits (perhaps 10,000 by 10,000), above which the library will not process an image. Ideally, Aspose would build their sanity checking so that images that fail the check would be blocked from loading entirely (as opposed to being loaded then checked, consuming large amounts of memory in the process); Aspose should *also* check after loading, as image files in some formats can be crafted to spoof their real dimensions.

Users could be allowed to explicitly raise the limit if necessary; the functionality for doing so should contain warnings about resource consumption in its documentation.

Recommendation: Allow Users to Set Resource Limits

In addition to the image size sanity checking described above, Aspose should also allow users to set overall memory, CPU, and time limits on the entire Aspose.Words library in order to more effectively manage resources and load on their services and applications.

Server-Side Request Forgery Via Linked Content

Severity: Medium

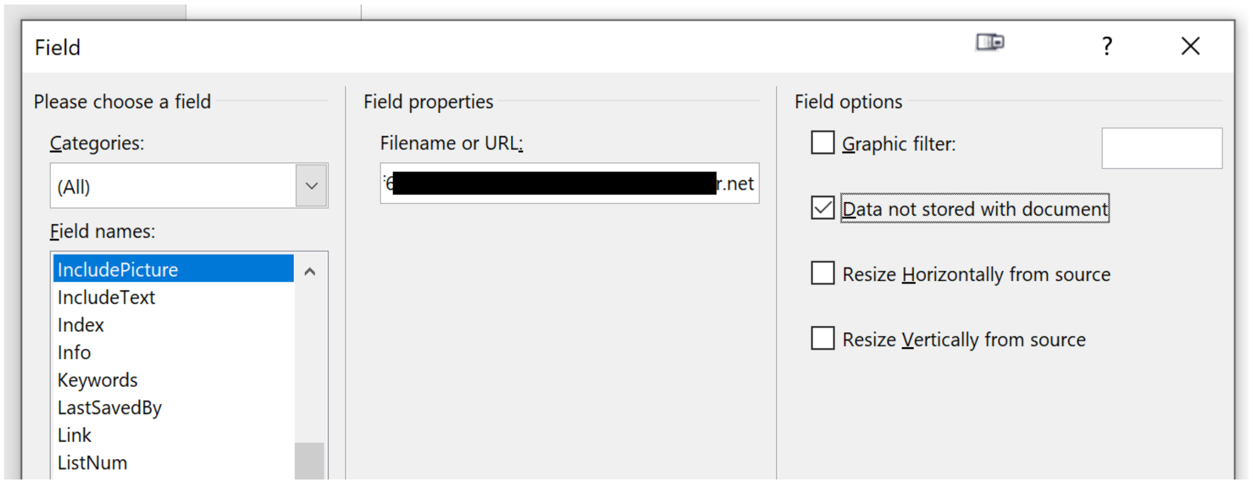
Creating a specially crafted document that includes linked (not embedded) content with HTTP paths will cause the server to request those images over the network when rendering a document, resulting in server-side request forgery against arbitrary URLs. This can be used to exfiltrate data stored on internally accessible services, perform port scans on internal networks using timing information, and de-cloak services that exist behind load balancers or DoS protection services.

This can be achieved in the following ways (list likely not exhaustive):

1. Linked images in Word doc(x) files (e.g., using the `IncludePicture` field)
2. Linked images in (X)HTML files (`<img>` tags)
3. Linked style sheets in (X)HTML files (`<link rel="stylesheet">`)

Attack: Internal Port Scan

An attacker could create a series of documents that contain embedded links to common internal IP address and port combinations, then submit them to a web service using the Aspose Words library to process the documents. Based on the length of time the service uses to process the document, the attacker could determine whether or not a given IP/Port combination is filtered by a firewall; longer processing times indicate the TCP SYN packet sent by the server was dropped by a firewall, while a quick processing time indicate a successful connection was made. Such a document could be constructed using the `IncludePicture` field in Word, among other ways:



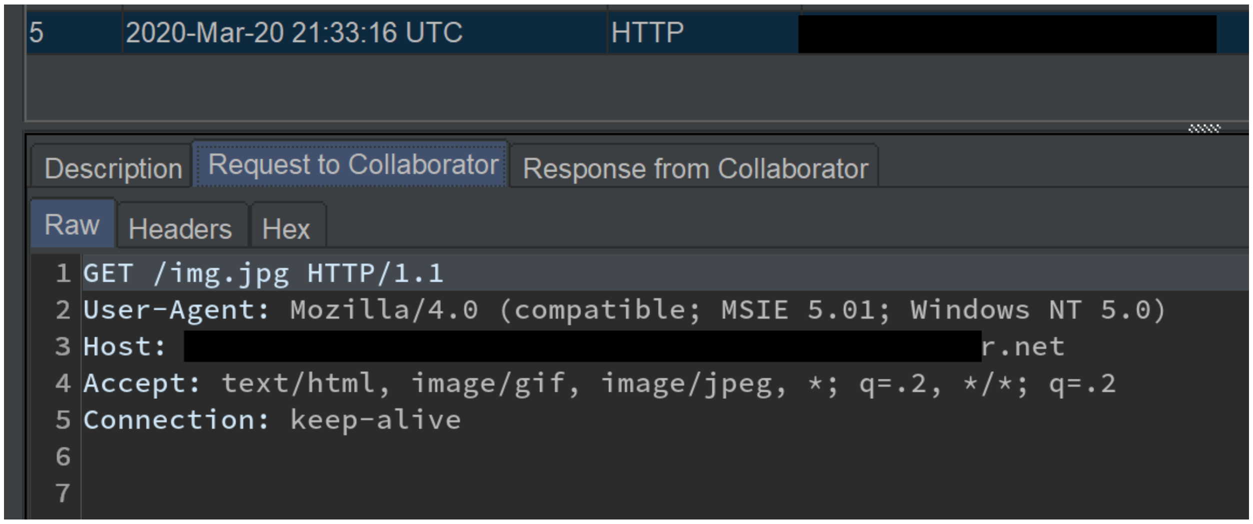
Categories: (All)

Field names:
- IncludePicture
- IncludeText
- Index
- Info
- Keywords
- LastSavedBy
- Link
- ListNum

Filename or URL: [redacted]r.net

- [ ] Graphic filter:
- [✓] Data not stored with document
- [ ] Resize Horizontally from source
- [ ] Resize Vertically from source

When processing the document, the server would make an HTTP request, as shown below:



```
5    2020-Mar-20 21:33:16 UTC    HTTP    [redacted]
```

```
GET /img.jpg HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: [redacted]r.net
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive


```

This functionality is enabled by default; however, it can be disabled by implementing the `IResourceLoadingCallback` interface.

Recommendation: Disable Remote Loading Functionality By Default

ISE recommends that Aspose disable remote loading functionality by default as described earlier in this document.

Strategic Weaknesses

ISE considers issues that lack direct exploitability but nonetheless reduce the security posture of a product (or in this case, the users of a product) to be “strategic weaknesses”. While they are typically of a lower severity and priority level than directly exploitable vulnerabilities, we nonetheless recommend they be addressed as part of the normal software development lifecycle.

File Type Auto Detection Cannot Be Disabled

Severity: Low

By default, the Aspose.Words library automatically detects the type of files as it loads them, ignoring any file extension or MIME type. Users can specify an expected file type using the `LoadOptions` class[2], but the library will fall back on automatic type detection if the specified loader fails. This dramatically increases the attack surface of any application that processes attacker-controlled documents using this library, as the attacker will be able to exploit a weakness in any of the library’s file formats against any application, even if that application has no reason to expect or allow that file format. For instance, an application that expects DOC files and configures Aspose.Words to expect them will nonetheless be vulnerable to any hypothetical weakness in the library’s HTML parsing routines. This can also result in attackers having access to more document features than the application developers expect; for instance, an application that attempts converts text files into PDFs could receive a DOC file with embedded images and the Aspose.Words library would still process such a file, images included, even though the original application developer did not intend to allow such documents to be produced.

In general, file type auto detection features are useful and convenient, but in cases where a specific file type is expected, they should be disabled to reduce attack surface. Libraries and applications that do not allow such features to be disabled present increased attack surface to attackers without providing benefit to legitimate users.

Recommendation: Allow Users to Disable File Type Auto Detection

ISE recommends that Aspose allow users to disable file type auto detection. This could be done using an addition to the existing `LoadOptions` class. It is acceptable for this feature to be on by default, but the documentation should remind users of the risks of doing so and instruct them on how to disable it if they wish.

Undisclosed Native Dependencies

Severity: Informational

The Aspose.Words library includes the HarfBuzz[3] embedded native dependency. No version information is included in the binary itself, and the documentation of the Aspose.Words library does not explicitly mention this dependency except in changelogs. Users who carefully examine the recommended Maven dependency entries for the library will see that a “HarfBuzz Shaping Plugin” is included, but users who are unfamiliar with the name will not understand that HarfBuzz is a native library and not a Java one. As a result, the Aspose.Words library may be affected by security vulnerabilities in HarfBuzz, without users of the library being aware of it; since the HarfBuzz libraries lack versioning information, users who do understand the risks involved in using native dependencies will not be able to determine if the version included in the Aspose.Words library is affected by any given security issue.

Additionally, the Aspose.Words library also depends on a number of additional Java libraries for advanced image processing, including the im4java[4] ImageMagick[5] Java bindings. ISE could find no mention of these additional

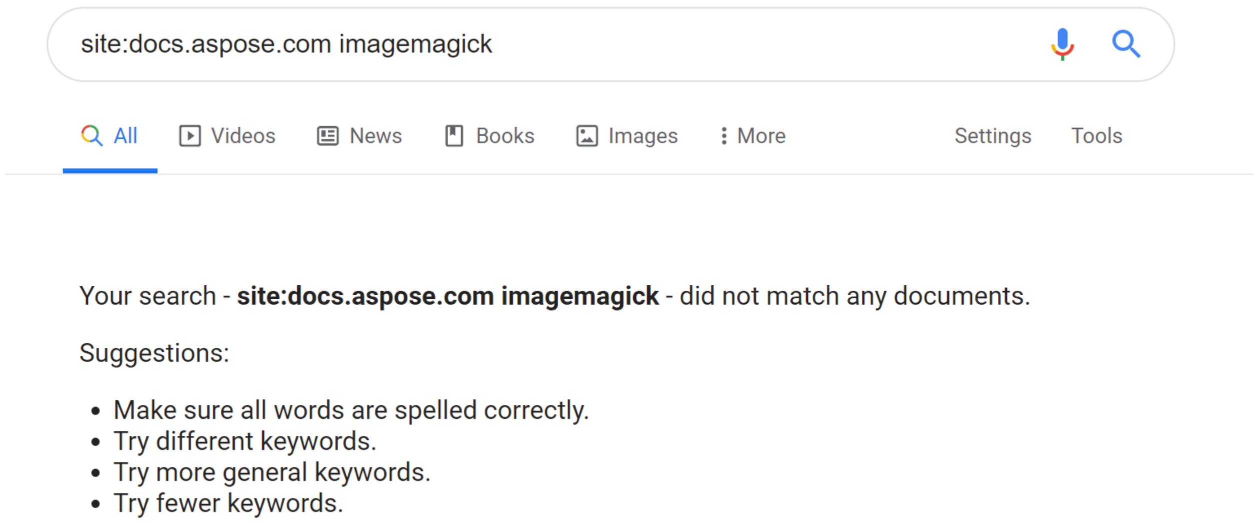
[2] https://apireference.aspose.com/java/words/com.aspose.words/loadoptions#LoadFormat

[3] https://www.freedesktop.org/wiki/Software/HarfBuzz/

[4] http://im4java.sourceforge.net/

[5] https://imagemagick.org/index.php

dependencies in the Aspose documentation, nor instructions for downloading and including them in an application from safe, trusted repositories:



Your search - **site:docs.aspose.com imagemagick** - did not match any documents.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.

The only way to discover this dependency is via an error message in the stack trace printed when the application fails to process an image.

This lack of documentation risks users downloading the requisite dependencies from untrusted repositories that may contain backdoored or otherwise malicious code, as well as exposing the application to the risks of using additional native libraries to handle potentially attacker-controlled data.

While simply *using* a third-party library is not itself a security risk, using *native* libraries does reduce many of the benefits of using “safe” languages such as Java for application development, as the underlying native libraries can and often are affected by classic security issues such as buffer overflows that do not affect “safe” languages. This in itself is not a reason to not use such libraries, especially in the case of HarfBuzz, which is essentially the only major text shaping library available. However, each additional library, and especially each additional *native* library, incurs additional risk and requires overhead in version tracking to ensure that security issues are patched in a timely manner. If a vendor such as Aspose fails to disclose their underlying dependencies properly, users of the software will be unable to perform this tracking and will be left at risk. In the case of Aspose, the HarfBuzz binary included in the Aspose.Words library lacks version information, so even a user that knows of the dependency will be unable to accurately track which version is in use, again increasing the risk of a security incident occurring.

Recommendation: Document Dependencies

ISE recommends that Aspose update their documentation to include all third-party dependencies (native or otherwise) of their libraries, including version numbers, so that users will be able to more accurately track the dependencies of their overall project and ensure that they patch security issues as they arise.

XML External Entities Not Explicitly Disabled

Severity: Informational

The Aspose.Words library appears to use the ‘woodstox’ XML parser, which by default turns off dangerous XML features. ISE recommends explicitly disabling dangerous XML features in case future versions of Aspose.Words enable dangerous XML features or swap the XML to another one which has dangerous features enabled by default.

Recommendation: Explicitly Disable Dangerous XML Features

ISE recommends that Aspose add explicit configuration directives to any XML parser created inside their library to 1) disable external entity and DTD retrieval, and 2) disable or limit to a reasonable depth entity expansion.

About ISE

ISE is an independent security firm headquartered in Baltimore, Maryland. We are dedicated to providing clients proven scientific strategies for defense. On every assessment our team of analysts and developers use adversary-centric approaches to protect digital assets, harden existing technologies, secure infrastructures, and work with development teams to improve our clients' overall security.

Assessing the client through the eyes of potential attackers allows us to understand possible threats and how to protect against those attacks. Our security analysts are experienced in information technology and product development which allows them to understand the security problems the client faces at every level. In addition, we conduct independent security research which allows us to stay at the forefront of the ever-changing world of information security.

Attacks on information systems cannot be stopped, however with robust security services provided by an experienced team, the effects of these attacks can often be mitigated or even prevented. We appreciate the confidence placed in us as a trusted security advisor. Please don't hesitate to get in touch for additional assistance with your security needs.

Independent Security Evaluators

4901 Springarden Drive

Suite 200

Baltimore, MD 21209

(443) 270-2296

contact@ise.io