ADALOGICS

Apache Log4Net & Log4CXX Security Audit 2024-2025

Audit Report

In collaboration with the Open Source Technology Improvement Fund

Adam Korczynski, David Korczynski

21st February 2025

Contents

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

Audit contacts

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Amir Montazery	Facilitator	OSTIF	amir@ostif.org
Derek Zimmer	Facilitator	OSTIF	derek@ostif.org
Helen Woeste	Facilitator	OSTIF	helen@ostif.org

Introduction

In December 2024 and January 2025, Ada Logics carried out a security audit of Apache Log4CXX and Log4Net. The audit was facilitated by the Open Source Technology Improvement Fund and had five weeks of auditing time allocated. This report presents the results from the audit.

Audit summary

The audit had three main goals. The first was to do a light threat modelling exercise to identify untrusted execution paths. The second was to audit the two libraries manually. The third was to improve the fuzz testing efforts of the code in scope where we saw it as meaningful. The fuzzing we did during the audit found several code-level issues in Log4CXX. All of these were triggerable only from trusted input. However, some of the issues had their root cause in general-purpose utility methods, and if they were triggerable from untrusted input, they could have a critical security impact. Over time, malicious contributors can attempt to expose such issues to untrusted input by using utility methods in contexts other than those currently used. The Log4CXX fixed all the issues we found during the audit. We integrated all of our fuzzing work into Log4CXX's OSS-Fuzz integration so that the project would benefit from the work even after the audit had finished.

Risk scoring

We use a simplified risk scoring system that considers risk exposure and risk impact. Exposure is the level at which an issue is exposed to an attacker. Impact is the level of privilege escalation an attacker can obtain by exploiting the security issue. We score both on a scale of 1-5 and add the two scores together for a final combined score. This score determines the severity of security issues. We assign the severity to the issues we find.

Risk Exposure

5: The security issue exists in core component(s) and is exposed in all use cases to untrusted input.
4: The security issue exists in widely used component(s) and is enabled by default. Users of the component(s) expose the issue by default to untrusted input.
3: The issue is exposed to authenticated and/or authorized users only.
2: The issue exists in component(s) that users need to enable to be affected.
1: The issue is only exposed to trusted users.

Risk Impact

5: An attack will have the highest possible impact.
4: An attack will have high impact with some constraints or limitations.

3: An attack can cause partial harm.
2: An attack can result in privilege escalation that will cause limited harm.
1: An attack can result in limited privilege escalation but requires further privilege escalation to cause harm.

We score each issue on both scales and then add the scores for a combined total score. The total score is the basis for the overall severity of found issues.

- 10: Critical
- 9 - 8: High
- 7 - 6: Moderate
- 5 - 4: Low
- 3 - 1: Informational

Scope

The audit included the code in the following code repositories:

1. https://github.com/apache/logging-log4cxx
2. https://github.com/apache/logging-log4net

The audit was not fixed to a particular commit; we worked constantly against the latest master branches.

The following commits are related to the audit and are either code contributions made by our team or fixes made by the upstream maintainers:

Log4CXX

https://github.com/apache/logging-log4cxx

Sorted by latest first

1. 0eac25dff2cf2a1737b72c4c8e65cd913b495f80
2. 32c131c8e4638e9309847c7994391ea527cfa536
3. 77142ede9b7c2a304f11b1dd245a091a9209c229
4. 1d7dda076acff30aa34e3751d445e822be1d73a7
5. e46c77ad1d259996228179c98600dc6b52f095a0
6. 323945bb7e2a9613baf8016886c7c53328ffdd4a
7. 1d53672ca5f415a064ad2bf08862ffd7f235e38f
8. 2cbae9546303bdfbbf05c7635a3079720b22fd96
9. 00cfd91e329b17603053568e61e04ac40510f331
10. a399da357b7b8db8369c943313a90337579b606a

11. 6463977c748b2bb8dd6912b38516dabd2f416267
12. ff5e6cb6eed59fedb485677b52fbdbd1ca4d08a8
13. b0775e7d4208d39007c33f7bc04f61eacff7979e
14. 537d25a11712861c92ca071830f370fb49d085bc
15. da0313eb333e4f76ed7ade48075025f64f9c29e9
16. 60f483ed80e973dd16cb51e110047a15f4a8e50f
17. cf78a3ace45dfda69228653d69ec45fc5a267ff5
18. 62ef32877b4a8b86aae58bad68857fd17b0724ec
19. dba1b0660e4cd64640d8a76d267f01e342c56854
20. 67de7fdfab79a1ee9f7489206b465191d95c3532
21. 4fb8904e051bc33f48eced1959f039305d645a73
22. d76e3db8f634c04742f42ea74890d4e72b6b8303

OSS-Fuzz

https://github.com/google/oss-fuzz

Sorted by latest first.

1. 71c2ec046aec9c0ba382a6ca777250480f38c833
2. 20c2e35aea99b0db3d9fb1efd707d289e5b463e8
3. e6e6af44088204b1438e481174048d7819dc7a75

Found issues

ID	Name	Severity	Status
ADA-APCH-LOG4-1	Global buffer overflow when reading optional configuration string	Moderate	Fixed
ADA-APCH-LOG4-2	Stack buffer overflow in SimpleDateFormatter	Moderate	Fixed
ADA-APCH-LOG4-3	Heap buffer overflow in SimpleDateFormatter	Moderate	Fixed
ADA-APCH-LOG4-4	Heap buffer overflow read from ill-formed XML entry in DOM Configurator	Moderate	Fixed
ADA-APCH-LOG4-5	NULL-dereference SEGV from non-existent appender	Low	Fixed
ADA-APCH-LOG4-6	A deeply nested XML value can cause a stack overflow	Low	Fixed
ADA-APCH-LOG4-7	Integer overflow from long minimum output length in pattern parser field	Low	Fixed
ADA-APCH-LOG4-8	Integer overflow from high number of minutes	Low	Fixed

Global buffer overflow when reading optional configuration string

Severity	Moderate
Status	Fixed
ID	ADA-APCH-LOG4-1
Library	Log4CXX

Log4CXX's `StringHelper::equalsIgnoreCase` is susceptible to a global buffer overflow if the formatted string contains a well-placed NULL byte. `StringHelper::equalsIgnoreCase` is widely used throughout the library, however only in cases where the trusted Log4CXX user provides the value. As such, the input is trusted and poses no security threat in Log4CXX's current state. Nonetheless, a malicious third-party contributor may try to expose the API to untrusted input in the future. The method in which the issue exists is a general-purpose utility function that reads a string that can be uppercase or lowercase. Log4CXX currently only uses it for reading optional parameters in configurations.

The issue triggers on line 43 below:
https://github.com/apache/logging-log4cxx/blob/b5bc99aa32aa4f1aef212898427a1b9ce9eaeb50/src/main/cpp/stringhelper.cpp#L31-L44

```
31 bool StringHelper::equalsIgnoreCase(const LogString& s1, const logchar*
       upper, const logchar* lower)
32 {
33     for (LogString::const_iterator iter = s1.begin();
34         iter != s1.end();
35         iter++, upper++, lower++)
36     {
37         if (*iter != *upper && *iter != * lower)
38         {
39             return false;
40         }
41     }
42
43     return (*upper == 0);
44 }
```

References:
- Vulnerable commit: b5bc99aa32aa4f1aef212898427a1b9ce9eaeb50
- OSS-Fuzz issue: https://issues.oss-fuzz.com/issues/391975637
- Upstream fix: https://github.com/apache/logging-log4cxx/pull/477

Stack buffer overflow in SimpleDateFormatter

Severity	Moderate
Status	Fixed
ID	ADA-APCH-LOG4-2
Library	Log4CXX

Log4CXX's `SimpleDateFormat::format` is susceptible to a stack-based buffer overflow from oddly formatted options and an empty log string.

The issue exists on line 859 below:

https://github.com/apache/logging-log4cxx/blob/cf78a3ace45dfda69228653d69ec45fc5a267ff5/src/main/cpp/simpledateformat.cpp#L850-L862

```
850 void SimpleDateFormat::format( LogString& s, log4cxx_time_t time, Pool&
        p ) const
851 {
852     apr_time_exp_t exploded;
853     apr_status_t stat = m_priv->timeZone->explode( & exploded, time );
854
855     if ( stat == APR_SUCCESS )
856     {
857         for ( PatternTokenList::const_iterator iter = m_priv->pattern.
                begin(); iter != m_priv->pattern.end(); iter++ )
858         {
859             ( * iter )->format( s, exploded, p );
860         }
861     }
862 }
```

The issue triggers from the following use case:

```
1  std::vector<log4cxx::LogString> options =
2      { LOG4CXX_STR("SZZ`+")
3      , LOG4CXX_STR("GMT1900853223")
4      , LOG4CXX_STR("")
5      , LOG4CXX_STR("")
6      , LOG4CXX_STR("")
7      };
8
9  log4cxx::LogString logger = LOG4CXX_STR("");
10 log4cxx::LevelPtr level = log4cxx::Level::getInfo();
```

```
log4cxx::spi::LoggingEventPtr event = log4cxx::spi::LoggingEventPtr(
        new log4cxx::spi::LoggingEvent(
                logger, level, LOG4CXX_STR(""), LOG4CXX_LOCATION));

DatePatternConverter* converter = new DatePatternConverter(options);
converter->format(event, logger, p);
```

When invoking the `DatePatternConverter` in such a manner, the following stack trace leads to the buffer overflow:

```
==14==ERROR: AddressSanitizer: stack-buffer-overflow on address 0
    x7f9fa85e630d at pc 0x557ef57a42f5 bp 0x7ffe39ca5d00 sp 0
    x7ffe39ca54c8
WRITE of size 1 at 0x7f9fa85e630d thread T0
SCARINESS: 46 (1-byte-write-stack-buffer-overflow)
    #0 0x557ef57a42f4 in memset /src/llvm-project/compiler-rt/lib/asan
        /../sanitizer_common/sanitizer_common_interceptors_memintrinsics
        .inc:87:3
    #1 0x557ef5a19c4f in std::__1::basic_string<char, std::__1::
        char_traits<char>, std::__1::allocator<char>>::append(unsigned
        long, char) (/out/PatternConverterFu
zzer+0x521c4f)
    #2 0x557ef586af24 in log4cxx::helpers::SimpleDateFormat::format(std
        ::__1::basic_string<char, std::__1::char_traits<char>, std::__1
        ::allocator<char>>&, long, lo
g4cxx::helpers::Pool&) const /src/logging-log4cxx/src/main/cpp/
    simpledateformat.cpp:859:16
    #3 0x557ef58d3694 in log4cxx::pattern::CachedDateFormat::
        findMillisecondStart(long, std::__1::basic_string<char, std::__1
        ::char_traits<char>, std::__1::allocat
or<char>> const&, std::__1::shared_ptr<log4cxx::helpers::DateFormat>
    const&, log4cxx::helpers::Pool&) /src/logging-log4cxx/src/main/cpp/
    cacheddateformat.cpp:202:16
    #4 0x557ef58d4d2d in log4cxx::pattern::CachedDateFormat::format(std
        ::__1::basic_string<char, std::__1::char_traits<char>, std::__1
        ::allocator<char>>&, long, log4cxx::helpers::Pool&) const /src/
        logging-log4cxx/src/main/cpp/cacheddateformat.cpp:307:30
```

This issue triggers independently of the log event string. For example, the following log event which contains a different untrusted log string:

```
log4cxx::spi::LoggingEventPtr event = log4cxx::spi::LoggingEventPtr(
        new log4cxx::spi::LoggingEvent(
                logger, level, LOG4CXX_STR(""), LOG4CXX_LOCATION));
```

As such, this issue is nort triggerable from untrusted input.

References:

- https://issues.oss-fuzz.com/issues/391935580
- Vulnerable commit: cf78a3ace45dfda69228653d69ec45fc5a267ff5

Heap buffer overflow in SimpleDateFormatter

Severity	Low
Status	Fixed
ID	ADA-APCH-LOG4-3
Library	Log4CXX

SimpleDateFormatter is vulnerable to a heap buffer overflow read from a misformed RFC822 timezone token. The issue exists when calling `SimpleDateFormat::format` with an ill-formed `log4cxx_time_t time`. `SimpleDateFormat::format` creates an `RFC822TimeZoneToken` on line 859:

https://github.com/apache/logging-log4cxx/blob/cf78a3ace45dfda69228653d69ec45fc5a267ff5/src/main/cpp/simpledateformat.cpp#L857-L860

```
857     for ( PatternTokenList::const_iterator iter = m_priv->pattern.begin
            (); iter != m_priv->pattern.end(); iter++ )
858     {
859         ( * iter )->format( s, exploded, p );
860     }
```

Which in turn will result in the heap overflow on line 620 below:

https://github.com/apache/logging-log4cxx/blob/cf78a3ace45dfda69228653d69ec45fc5a267ff5/src/main/cpp/simpledateformat.cpp#L601-L621

```
601     apr_int32_t off = tm.tm_gmtoff;
602     size_t basePos = s.length();
603     s.append( LOG4CXX_STR( "+0000" ) );
604
605     if ( off < 0 )
606     {
607         s[basePos] = 0x2D; // '-'
608         off = -off;
609     }
610
611     LogString hours;
612     StringHelper::toString( off / 3600, p, hours );
613     size_t hourPos = basePos + 2;
614
615     //
```

```
616      //   assumes that point values for 0-9 are same between char and
             wchar_t
617      //
618      for ( size_t i = hours.length(); i-- > 0; )
619      {
620          s[hourPos--] = hours[i];
621      }
```

The issue was fixed by setting a limit to the timezone range: https://github.com/apache/logging-log4cxx/pull/464.

References:

- https://issues.oss-fuzz.com/issues/392279305

Heap buffer overflow read from ill-formed XML entry in DOM Configurator

Severity	Low
Status	Fixed
ID	ADA-APCH-LOG4-4
Library	Log4CXX

If the attribute of an XML entry in the DOM Configurator is over 2000 characters long and ends with a comma, the XML iterator can increment past the end of the string which results in a heap buffer overflow read.

The following XML string can cause the issue:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Aog4j:configuration SYSTEM "l!g5j.dtd">
<!--
 Licensed to the Apa4407370chy Software Foundation (ASF) under one or more
 contributor license agreements.  See the NOTICE file distre.

-->

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="A1" Class="org.apache.log4~.FileApp3AAAAorg.apachd.log4j.filAAAAAAAAAAAAAAAAAAAAAAAAAHAAAAAAAAAAAAAAAAAAAAAAAAAAA"
     class="org.apachd.log4j.FileAppender">
    <param name="File" value="output\temp.A2.2" />
    <param name="Append" value="false" />
    <layout class="oarg.pache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{
          ooooonnnnnnnnnnnnnnnnnnnn~nnnnnnnnnnnnnnnAAAAzAAAA~
          nnnnnnnnnnnnnnnnn.nnnnnT WANARTRIES \fffffffffffff\f\\\\f
          \\\\\
          xmlnseadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAA2mlnsAAAAAAAAAAAAAA
           AAAAAAAAAAAAnnnnnnnnnnnnn.
          nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnAAAAAAAApzAAzAAAAn
           nnnnn nnnnnnnnnnnnnnnnnnAAAAAA
          \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
          fffffffffFffff\f\\\\fX\\\\\
          AthreadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxmlnsAAAAAAAAnnnnnnd
           ting, software
```

```
15  distributed qnder the License
       nn$nnnnnnnnnAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAthreadted qnder
       the License is distribuveorg.apache.log3j.pattnnnnnnnnnorg.apache.
       log4j.patternlayoutT WANARTRIES OR (CONDI:IONS OF ANY KIND, either
       express o.......nnAAAAAAAAnnnnnnnnnnnnnnnnnnn.nnn ?lnnnnnnnnnnnnnnnn
       ~
       nnnnnnnAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAthreadnamAAAAAAAAAAAAA
        AAAAAAAAAAAAAAAAAorg.apache.
       lollllllllllllllllAAAAAnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn~
       nnnnnnnnnnnnnnnAAAAzAAAA~nnnnnnnnnnnnnnnnn.nnNnnT WANARTRIES \
       fffffffffffffff\f\\\\f\\\\\\
       AthreadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxmlnsAAAAAAAAAAAAAAAAAAAA
        AAAAAAAnnnnnnnnnnnnn.
       nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnAAAAAAAAApzAAzAAAAn
       nnnnn nnnnnnnnnnnnnnnnnnnAAAAAA
       \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
       fffffffffffffff\f\\\\f\\\\\\
       AthreadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxmlnsAAAAAAAAAAAAAAAAAAAA
        AAAAAAAnnnnnnnnnnnnn.
       nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnAAAAA\
       AthreadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxmlnsAAAAAAAAnnnnnndting,
        software
16  distributed qnder the License
       nn$nnnnnnnnnAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAthreadted qnder
       the License is distribuve%org.apache.log3j.pattnnnnnnnnnorg.apache.
       log4j.patternlayoutT WANARTRIES OR (CONDI:IONS OF ANY KIND, either
       express o.......nnAAAAAAAAnnnnnnnnnnnnnnnnnnn.nnn ?lnnnnnnnnnnnnnnnn
       ~
       nnnnnnnAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAthreadnamAAAAAAAAAAAAA
        AAAAAAAAAAAAAAAAAorg.apache.
       lollllllllllllllllAAAAAnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn~
       nnnnnnnnnnnnnnnAAAAzAAAA~nnnnnnnnnnnnnnnnn.nnNnnT WANARTRIES \
       fffffffffffffff\f\\\\f\\\\\\
       AthreadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxmlnsAAAAAAAAAAAAAAAAAAAA
        AAAAAAAnnnnnnnnnnnnn.
       nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnAAAAAAAAApzAAzAAAAn
       nnnnnnnnnnnnnnnnnnnnnnnnAAAAAA
       \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
       fffffffffffffff\f\\\\f\\\\\\
       AthreadnamAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxmlnsAAAAAAAAAAAAAAAAAAAA
        AAAAAAAnnnnnnnnnnnnn.
       nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnAAAAAAAAApzAAzAAAAn
       nnnnn
       eAAAAAAAAAAAAAAAAKAAAAAAAAAAAAAAnnnnnnonnnnnnnnnnnnnnnnnnnnwobject
       nnnnnnnnnnnn1111111111111111nnnnnnnnnnnnnnorg.
       AAAApzAAzAAAAnnnnnneAAAAAAAAAAAAAAAKAAAAAAAAAAAAAAnnnnnnonnnnnnnnn
       nnnnnnnnnnnnwobjectnnnnnnnnnnnn1111111111111111nnnnnnnnnnnnnorg.
       apach_.log0j.
       methodnbayoutnnnnnnnAnnnnnnnnnnnnloggernnnnnnnnnISO8601} [%t] %-5
       lopz- %m%n"/>
17     </layout>
```

```xml
  </appender>

  <logger name="orgac.aph.elog3j.xml">
    <level value="debug" />
    <appender-ref ref="A1" />
  ndc</logger>

  <root>
    <priority value ="debug" />
    <appender-rezf ref="A0" />
    <appender-fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff ffffffffffffffffodbcapffffffffffffffffffffffffffffffref ref="A4" />
  </root>

</log4j:configuration>
```

With the above XML string in a file, we configured the DOM Configurator as such:

```cpp
DOMConfigurator::configure("conf.xml");
```

Which resulted in the following stack trace:

```
    ==410==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x51d000003058 at pc 0x5a16ad7aecb4 bp 0x7ffc8ae7c760 sp 0x7ffc8ae7c758
READ of size 1 at 0x51d000003058 thread T0
SCARINESS: 12 (1-byte-read-heap-buffer-overflow)
    #0 0x5a16ad7aecb3 in log4cxx::helpers::OptionConverter::convertSpecialChars(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>> const&) /src/logging-log4cxx/src/main/cpp/optionconverter.cpp:86:7
    #1 0x5a16ad7bc30d in log4cxx::PatternLayout::setOption(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>> const&, std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>> const&) /src/logging-log4cxx/src/main/cpp/patternlayout.cpp:147:31
    #2 0x5a16ad7f1af4 in log4cxx::config::PropertySetter::setProperty(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>> const&, std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>> const&, log4cxx::helpers::Pool&) /src/logging-log4cxx/src/main/cpp/propertysetter.cpp:98:12
    #3 0x5a16ad6d5c16 in log4cxx::xml::DOMConfigurator::setParameter(log4cxx::helpers::Pool&, std::__1::shared_ptr<log4cxx::helpers::CharsetDecoder>&, apr_xml_elem*, log4cxx::config::PropertySetter&) /src/logging-log4cxx/src/main/cpp/domconfigurator.cpp:811:13
    #4 0x5a16ad6d694a in log4cxx::xml::DOMConfigurator::parseLayout(log4cxx::helpers::Pool&, std::__1::shared_ptr<log4cxx::helpers::CharsetDecoder>&, apr_xml_elem*) /src/logging-log4cxx/src/main/
```

```
          cpp/domconfigurator.cpp:606:5
9       #5 0x5a16ad6d0825 in log4cxx::xml::DOMConfigurator::parseAppender(
          log4cxx::helpers::Pool&, std::__1::shared_ptr<log4cxx::helpers::
          CharsetDecoder>&, apr_xml_elem*, apr_xml_doc*, std::__1::map<std
          ::__1::basic_string<char, std::__1::char_traits<char>, std::__1
          ::allocator<char>>, std::__1::shared_ptr<log4cxx::Appender>, std
          ::__1::less<std::__1::basic_string<char, std::__1::char_traits<
          char>, std::__1::allocator<char>>>, std::__1::allocator<std::__1
          ::pair<std::__1::basic_string<char, std::__1::char_traits<char>,
           std::__1::allocator<char>> const, std::__1::shared_ptr<log4cxx
          ::Appender>>>>&) /src/logging-log4cxx/src/main/cpp/
          domconfigurator.cpp:246:25
10      #6 0x5a16ad6ce66e in log4cxx::xml::DOMConfigurator::
          findAppenderByName(log4cxx::helpers::Pool&, std::__1::shared_ptr
          <log4cxx::helpers::CharsetDecoder>&, apr_xml_elem*, apr_xml_doc
          *, std::__1::basic_string<char, std::__1::char_traits<char>, std
          ::__1::allocator<char>> const&, std::__1::map<std::__1::
          basic_string<char, std::__1::char_traits<char>, std::__1::
          allocator<char>>, std::__1::shared_ptr<log4cxx::Appender>, std::
          __1::less<std::__1::basic_string<char, std::__1::char_traits<
          char>, std::__1::allocator<char>>>, std::__1::allocator<std::__1
          ::pair<std::__1::basic_string<char, std::__1::char_traits<char>,
           std::__1::allocator<char>> const, std::__1::shared_ptr<log4cxx
          ::Appender>>>>&) /src/logging-log4cxx/src/main/cpp/
          domconfigurator.cpp:153:15
11      #7 0x5a16ad6ce9e9 in log4cxx::xml::DOMConfigurator::
          findAppenderByName(log4cxx::helpers::Pool&, std::__1::shared_ptr
          <log4cxx::helpers::CharsetDecoder>&, apr_xml_elem*, apr_xml_doc
          *, std::__1::basic_string<char, std::__1::char_traits<char>, std
          ::__1::allocator<char>> const&, std::__1::map<std::__1::
          basic_string<char, std::__1::char_traits<char>, std::__1::
          allocator<char>>, std::__1::shared_ptr<log4cxx::Appender>, std::
          __1::less<std::__1::basic_string<char, std::__1::char_traits<
          char>, std::__1::allocator<char>>>, std::__1::allocator<std::__1
          ::pair<std::__1::basic_string<char, std::__1::char_traits<char>,
           std::__1::allocator<char>> const, std::__1::shared_ptr<log4cxx
          ::Appender>>>>&) /src/logging-log4cxx/src/main/cpp/
          domconfigurator.cpp:159:14
12      #8 0x5a16ad6d452f in log4cxx::xml::DOMConfigurator::
          findAppenderByReference(log4cxx::helpers::Pool&, std::__1::
          shared_ptr<log4cxx::helpers::CharsetDecoder>&, apr_xml_elem*,
          apr_xml_doc*, std::__1::map<std::__1::basic_string<char, std::
          __1::char_traits<char>, std::__1::allocator<char>>, std::__1::
          shared_ptr<log4cxx::Appender>, std::__1::less<std::__1::
          basic_string<char, std::__1::char_traits<char>, std::__1::
          allocator<char>>>, std::__1::allocator<std::__1::pair<std::__1::
          basic_string<char, std::__1::char_traits<char>, std::__1::
          allocator<char>> const, std::__1::shared_ptr<log4cxx::Appender
          >>>>&) /src/logging-log4cxx/src/main/cpp/domconfigurator.cpp
          :190:14
13      #9 0x5a16ad6de902 in log4cxx::xml::DOMConfigurator::
```

```
            parseChildrenOfLoggerElement(log4cxx::helpers::Pool&, std::__1::
            shared_ptr<log4cxx::helpers::CharsetDecoder>&, apr_xml_elem*,
            std::__1::shared_ptr<log4cxx::Logger>, bool, apr_xml_doc*, std::
            __1::map<std::__1::basic_string<char, std::__1::char_traits<char
            >, std::__1::allocator<char>>, std::__1::shared_ptr<log4cxx::
            Appender>, std::__1::less<std::__1::basic_string<char, std::__1
            ::char_traits<char>, std::__1::allocator<char>>>, std::__1::
            allocator<std::__1::pair<std::__1::basic_string<char, std::__1::
            char_traits<char>, std::__1::allocator<char>> const, std::__1::
            shared_ptr<log4cxx::Appender>>>>&) /src/logging-log4cxx/src/main
            /cpp/domconfigurator.cpp:542:27
14      #10 0x5a16ad6ddbb2 in log4cxx::xml::DOMConfigurator::parseLogger(
            log4cxx::helpers::Pool&, std::__1::shared_ptr<log4cxx::helpers::
            CharsetDecoder>&, apr_xml_elem*, apr_xml_doc*, std::__1::map<std
            ::__1::basic_string<char, std::__1::char_traits<char>, std::__1
            ::allocator<char>>, std::__1::shared_ptr<log4cxx::Appender>, std
            ::__1::less<std::__1::basic_string<char, std::__1::char_traits<
            char>, std::__1::allocator<char>>>, std::__1::allocator<std::__1
            ::pair<std::__1::basic_string<char, std::__1::char_traits<char>,
             std::__1::allocator<char>> const, std::__1::shared_ptr<log4cxx
            ::Appender>>>>&) /src/logging-log4cxx/src/main/cpp/
            domconfigurator.cpp:459:2
15      #11 0x5a16ad6e94bb in log4cxx::xml::DOMConfigurator::parse(log4cxx
            ::helpers::Pool&, std::__1::shared_ptr<log4cxx::helpers::
            CharsetDecoder>&, apr_xml_elem*, apr_xml_doc*, std::__1::map<std
            ::__1::basic_string<char, std::__1::char_traits<char>, std::__1
            ::allocator<char>>, std::__1::shared_ptr<log4cxx::Appender>, std
            ::__1::less<std::__1::basic_string<char, std::__1::char_traits<
            char>, std::__1::allocator<char>>>, std::__1::allocator<std::__1
            ::pair<std::__1::basic_string<char, std::__1::char_traits<char>,
             std::__1::allocator<char>> const, std::__1::shared_ptr<log4cxx
            ::Appender>>>>&) /src/logging-log4cxx/src/main/cpp/
            domconfigurator.cpp:1165:4
16      #12 0x5a16ad6e58c1 in log4cxx::xml::DOMConfigurator::doConfigure(
            log4cxx::File const&, std::__1::shared_ptr<log4cxx::spi::
            LoggerRepository>) /src/logging-log4cxx/src/main/cpp/
            domconfigurator.cpp:896:4
17      #13 0x5a16ad6ea92e in log4cxx::xml::DOMConfigurator::configure(std
            ::__1::basic_string<char, std::__1::char_traits<char>, std::__1
            ::allocator<char>> const&) /src/logging-log4cxx/src/main/cpp/
            domconfigurator.cpp:913:27
18      #14 0x5a16ad6cd547 in log4cxx::xml::DOMConfigurator::configure(char
             const*) /src/logging-log4cxx/src/main/include/log4cxx/xml/
            domconfigurator.h:223:76
```

References:

- Vulnerable commit: b2f7aac6a35bdc9d4337d4335f27b0629bf63d90
- https://issues.oss-fuzz.com/issues/393411512

NULL-dereference SEGV from non-existent appender

Severity	Low
Status	Fixed
ID	ADA-APCH-LOG4-5
Library	Log4CXX

If the user attempts to use a non-existent appender, Log4CXX can trigger a SEGV. This is only triggerable by the trusted Log4CXX user.

The crash happens on line 229 in case the appender does not exist:

https://github.com/apache/logging-log4cxx/blob/b9de415c392908e826c592ae40511f70f5fe9492/src/main/cpp/domconfigurator.cpp#L210-L229

```
210 AppenderPtr DOMConfigurator::parseAppender(Pool& p,
211     LOG4CXX_NS::helpers::CharsetDecoderPtr& utf8Decoder,
212     apr_xml_elem* appenderElement,
213     apr_xml_doc* doc,
214     AppenderMap& appenders)
215 {
216
217     LogString className(subst(getAttribute(utf8Decoder, appenderElement
        , CLASS_ATTR)));
218     if (LogLog::isDebugEnabled())
219     {
220         LogLog::debug(LOG4CXX_STR("Class name: [") + className +
            LOG4CXX_STR("]"));
221     }
222
223     try
224     {
225         ObjectPtr instance = ObjectPtr(Loader::loadClass(className).
            newInstance());
226         AppenderPtr appender = LOG4CXX_NS::cast<Appender>(instance);
227         PropertySetter propSetter(appender);
228
229         appender->setName(subst(getAttribute(utf8Decoder,
            appenderElement, NAME_ATTR)));
```

References:

- https://issues.oss-fuzz.com/issues/393164896

- https://github.com/apache/logging-log4cxx/pull/471/files

A deeply nested XML value can cause a stack overflow

Severity	Low
Status	Fixed
ID	ADA-APCH-LOG4-6
Library	Log4CXX

If an appender-ref tag is added with no ref, parsing the XML file gets stuck in a recursive loop until the stack overflows. Check to make sure that the ref attribute is in the XML file.

When the DOMConfigurator parses the appender, it can call into `findAppenderByReference` on line 307if the tag name is an appender reference tag:

https://github.com/apache/logging-log4cxx/blob/b9de415c392908e826c592ae40511f70f5fe9492/src/main/cpp/domconfigurator.cpp#L294-L308

```
294         else if (tagName == APPENDER_REF_TAG)
295       {
296           LogString refName = subst(getAttribute(utf8Decoder,
                 currentElement, REF_ATTR));
297
298           if (appender->instanceof(AppenderAttachable::getStaticClass()))
299           {
300               AppenderAttachablePtr aa = LOG4CXX_NS::cast<
                     AppenderAttachable>(appender);
301               if (LogLog::isDebugEnabled())
302               {
303                   LogLog::debug(LOG4CXX_STR("Attaching appender named [")
                          +
304                       refName + LOG4CXX_STR("] to appender named [") +
305                       appender->getName() + LOG4CXX_STR("]."));
306               }
307               aa->addAppender(findAppenderByReference(p, utf8Decoder,
                     currentElement, doc, appenders));
308           }
```

`findAppenderByReference` calls into `findAppenderByName` on line 190:
https://github.com/apache/logging-log4cxx/blob/b9de415c392908e826c592ae40511f70f5fe9492/src/main/cpp/domconfigurator.cpp#L188-L196

```
188       else if (doc)
189       {
```

```
190         appender = findAppenderByName(p, utf8Decoder, doc->root, doc,
                appenderName, appenders);
191
192         if (appender)
193         {
194             appenders.insert(AppenderMap::value_type(appenderName,
                    appender));
195         }
196     }
```

`findAppenderByName` can call into `parseAppender` on line 153:

https://github.com/apache/logging-log4cxx/blob/b9de415c392908e826c592ae40511f70f5fe9492/src/main/cpp/domconfigurator.cpp#L149-L155

```
149     if (tagName == APPENDER_TAG)
150     {
151         if (appenderName == getAttribute(utf8Decoder, element,
                NAME_ATTR))
152         {
153             appender = parseAppender(p, utf8Decoder, element, doc,
                    appenders);
154         }
155     }
```

References:

- https://github.com/apache/logging-log4cxx/pull/472
- https://issues.oss-fuzz.com/issues/393164849

Integer overflow from long minimum output length in pattern parser field

Severity	Low
Status	Fixed
ID	ADA-APCH-LOG4-7
Library	Log4CXX

Log4CXX users can misconfigure format modifiers in a way that causes Log4cxx to trigger an integer overflow. The Log4CXX user can set a high number of minimum characters to output. This number in itself will not overflow, but on the following lines, the formatted minimum length gets multiplied with 10 which can overflow:

https://github.com/apache/logging-log4cxx/blob/0eac25dff2cf2a1737b72c4c8e65cd913b495f80/src/main/cpp/patternparser.cpp#L223-L233

```
223      case MIN_STATE:
224          currentLiteral.append(1, c);
225
226          if ((c >= 0x30 /* '0' */) && (c <= 0x39 /* '9' */) &&
                 minDigitCount < 3)
227          {
228              formattingInfo = std::make_shared<FormattingInfo>(
229                          formattingInfo->isLeftAligned(),
230                          (formattingInfo->getMinLength() * 10) + (c - 0
                                x30 /* '0' */),
231                          formattingInfo->getMaxLength());
232              ++minDigitCount;
233          }
```

The issue was fixed in https://github.com/apache/logging-log4cxx/pull/475 by setting a limit to the minimum character length such that the value is no longer than 3 digits.

This issue triggers when invoking `PatternParser::parse` with the oddly configured `fields` parameter. We did not find any use of `PatternParser::parse` that triggers this bug from untrusted data.

References:

- https://issues.oss-fuzz.com/issues/393666942

Integer overflow from high number of minutes

Severity	Low
Status	Fixed
ID	ADA-APCH-LOG4-8
Library	Log4CXX

Log4CXX users can misconfigure format modifiers in a way that causes Log4CXX to trigger an integer overflow. The Log4CXX user can set a high number of minutes in the timezone date format. This number in itself will not overflow, but on the following lines, it gets multiplied with 60 which can result in the integer overflow:

https://github.com/apache/logging-log4cxx/blob/b0775e7d4208d39007c33f7bc04f61eacff7979e/src/main/cpp/timezone.cpp#L279-L281

```
279      s.append(mm);
280      apr_int32_t offset = sign * (hours * 3600 + minutes * 60);
281      return std::make_shared<helpers::TimeZoneImpl::FixedTimeZone>( s,
            offset );
```

The issue was fixed in https://github.com/apache/logging-log4cxx/pull/476 by adding a check that ensures that the minute integer is not higher than 60.

This issue triggers when invoking `TimeZone::getTimeZone` with the high minute integer. We did not find any use of `TimeZone::getTimeZone` that triggers this bug from untrusted data.

References:
- Vulnerable commit: b0775e7d4208d39007c33f7bc04f61eacff7979e
- https://issues.oss-fuzz.com/issues/393411535