LOG4J - CVE-2021-44228, CVE-2021-45046, CVE-2021-45105

PUBLISHED: DECEMBER 14, 2021 | LAST UPDATE: JANUARY 31, 2022

SUMMARY

In December 2021, Apache Log4j (logging.apache.org) [1] published security advisories detailing a number of critical security issues. These vulnerabilities, if exploited, can allow attackers to construct specifically crafted packets which lead to Remote Code Execution or Denial of Service (DoS) of exposed systems.

The following vulnerabilities reported by Apache Log4j are addressed in this document.

Item #	Vulnerability ID	Score Source	Score	Summary
1	CVE-2021-44228	CVSS 3.0	10.0 Critical	log4j: Remote code execution in Log4j 2.x when logs contain an attacker-controlled string value [2]
2	CVE-2021-45046	CVSS 3.0	9.0 Critical	log4j-core: DoS in log4j 2.x with thread context message pattern and context lookup pattern (incomplete fix for CVE-2021-44228) [3]
3	CVE-2021-45105	CVSS 3.0	7.5 High	log4j-core: DoS in log4j 2.x with Thread Context Map (MDC) input data contains a recursive lookup and context lookup pattern [4]

AFFECTED PRODUCTS AND RELEASES

The table below indicates the vulnerability status of A10 products for these vulnerabilities. Unless specific models or product software releases are indicated, the vulnerability status should be considered to reflect all product models and software releases.

Product	Vulnerability Status
A10 Thunder	Not affected
A10 vThunder (Virtual Thunder)	Not affected
A10 cThunder (Container Thunder)	Not affected
A10 AX Series	Not affected
A10 aGalaxy	Releases Affected: 3.0.0 – 5.0.9 Releases Resolved: 5.0.10
A10 Harmony Controller	Not affected
A10 Enterprise License Manager (ELM)	Not affected

WORKAROUNDS AND MITIGATIONS

Even though A10 products are not affected by these vulnerabilities, A10 Thunder, vThunder and cThunder products can be used to mitigate exposures to these vulnerabilities for applications and systems in the infrastructure they serve by applying ACOS AFLEX policies.

AFLEX MITIGATIONS

This AFLEX policy for mitigating exploits of the Apache Log4j CVE-2021-44228 and CVE-2021-45105 vulnerabilities is based on and has been tested based on information available to A10 at this time. Though A10 believes this policy can help mitigate common exploits of CVE-2021-44228 and CVE-2021-45105, it may not mitigate all scenarios or variants of exploits for this vulnerability.

```
##### AFLEX for Log4j CVE-2021-44228 and CVE-2021-45105 #####
#
# This AFLEX script attempts to match enabled RegEx pattern(s) specified in
# the list variable "patterns" and take action on them. Messages that match
# these patterns will be logged and dropped.
#
# Matching is performed the against the following:
#
#    1. HTTP URI       - URL-encoded
#                      - URI as clear text
#
#    2. HTTP Headers   - header name
#                      - header value
#
#    3. HTTP PUT/POST - payload
#
# Script Usage / Parameters
# -------------------------
#    1. Choose the pattern (or patterns) you want enable in the "patterns".
#       list variable by un-commenting it (or them).
#
#       NOTES:
#          * Enabling multiple patterns will increase the processing overhead
#            for this AFLEX script
#          * All patterns can potentially drop/log valid HTTP request, put,
#            and post messages.
#          * Patterns indicated as more aggressive, are more prone to
#            dropping/logging otherwise valid HTTP messages.
#          * Though multiple patterns are supported, some patterns can
#            be supersets of others.
#
#          * DEFAULT SETTING: Least aggressive pattern.
#
#    2. Select the number nesting levels to decode for URL encode URI
#       by setting the 'decodeTrials' variable to a value of 1 or larger.
#
#       NOTES:
#          * For 'decodeTrials’ settings:
#                - 1 will allow 1 nesting level (minimum setting)
#                - 2 will allow 2 nesting level
#                - ... and so on
#
#          * DEFAULT SETTING: 5 (to allow 5 nesting levels).
#
#    3. Select whether logging facility Tag to be applied if desired.
#       Options include no tag or facility ('local0', 'local1'...) with priority level (DEBUG, INFO...)
#
#       WARNING:
#          * Logging  matched string to SYSLOG servers could potentially put these servers
#            at risk if they are themselves exposed to this Log4j vulnerability.
#
#          * DEFAULT SETTING: "" (null - generic log)
#
#    4. Select whether to include logging of URI content upon matching
#       events by setting 'logUri' parameter to TRUE (1) or to disable
#       this content from being logged by setting to FALSE (0).
#
#       NOTES:
#          * URI can be very long, so this option is provided
#
#          * DEFAULT SETTING: TRUE (include URI for logging of matches
#                                   found in URI)
#          * Same for logPayload / logHeader
#
#   5. Choose the maximum amount that this AFLEX will scan into POST/PUT
#      payloads by setting the 'contentMax' variable to the number of
#      bytes to be inspected. A value of '0'indicates that the full
#      payload will be inspected.
#
#       NOTES:
#          * Selection of this value can notably impact overhead for
#            processing HTTP through the ACOS device for POST/PUT
#            operations.
#          * There is a trade-off here. The lower the payload size,
#            the lower the detectability for this CVE.
#          * Alternately, with larger sizes comes more processing
#            overhead, though better detection.
```

```
#
#          * DEFAULT SETTING: 0 (all payloads)
#
# Other Common Customizations
# ---------------------------
#     1. Monitor & Log Only
#          * To monitor and log matches for Log4j HTTP messages without
#            any actions taken on the traffic:
#                - Simply comment out the 'reject' operations in the script
#                  content.
#
# Other Notes
# -----------
#     1. Log messages in the local ACOS event log and as shared with
#        SYSLOG servers configured in the ACOS system will be truncated to
#        1024 bytes
#         * Accordingly, detected/matching information may not be
#           included  or may be partially included in log entries. This will
#           especially be the case where the matching content is deeper than
#           1024 bytes into the associated being scanned content.
#
# Revision History
# ----------------
#   v1.0  2021-12-20    Initial release
#   v2.0  2021-12-27    Update patterns for CVE-2021-45105 ctx:<name> case, raise decodeTrials to 5
#
#########################

when HTTP_REQUEST {

  ####### Set Pattern Parameters Here ####
# Select the pattern from this list and insert in set pattens below
# includes CVE-2021-45105 addition to detect ctx:<name> case
# least aggressive pattern (still fewer false positives dropped) with ctx case
#     {\$(\\)*\{\s*(j|c|(\$\{.+(j|c)\})+)+}
#     {\$\{\s*(j|c|(\$\{.+(j|c)\})+)+}
# less aggressive pattern (fewer false positives dropped)
#     {\$\{\s*(j|c|\$\{.+(-|:)(j|c)\})}
#     {\$\{\s*(j|c|\$?\{.+?\})}
#     {\$\{\s*(j|c|\$\{.+(j|c)\})}
# without CVE-2021-45105 addition to detect ctx:<name> case
#     {\$(\\)*\{\s*(j|(\$\{.+j\})+)+}
#     {\$\{\s*(j|(\$\{.+j\})+)+}
# less aggressive pattern (fewer false positives dropped)
#     {\$\{\s*(j|\$\{.+(-|:)j\})}
#     {\$\{\s*(j|\$?\{.+?\})}
#     {\$\{\s*(j|\$\{.+j\})}
# very aggressive pattern (probably many false positives dropped)
#     {\$\{.+}
# placeholder for future patterns
#     { placeholder }
  set patterns {
      {\$(\\)*\{\s*(j|c|(\$\{.+(j|c)\})+)+}
  }

  ### max number of times we want to decode nested URL encoding
  set decodeTrials 5

  ### logging format, default to true, set to 0 to enable remote
  set logTag ""
# set logTag "local0.6"
# set logTag "local0.5"
# set logTag "local1.5"

  ### logging detail URI, default to true, set to 0 to disable
  set logUri     1
  set logHeader  1
  set logPayload 1

  ### Max size to do collect, 0 if collect all
  ## default collect all
  set contentMax 0

  ###### End Parameters  ########
```

```
set decodeUri [HTTP::uri ]
set rawUri  ""
set origUri $decodeUri

# decoding nested uri
for {set i 0} {($i <= $decodeTrials) && ($decodeUri ne $rawUri) } {incr i} {

  # match at each round?
  foreach pattern $patterns {
    if {[string tolower $decodeUri] matches_regex $pattern} {
      if {$logUri == 1} {
        set logMsg $origUri
      } else {
        set logMsg ""
      }
      log $logTag "log4j2 -- decoded $i times, match uri $logMsg to regex $pattern"
      reject
      return
    }
  }
  set rawUri $decodeUri
  set decodeUri [URI::decode $rawUri]

}

if {$i >= $decodeTrials} {
  log $logTag "log4j2 -- exceeding max decode attempts $decodeTrials"
  reject
  return
}


### HTTP headers ###
### Checks header name and header value
foreach header [HTTP::header names] {
   set val [HTTP::header value $header]
   set vlower [string tolower $val]
   set nlower [string tolower $header]

   foreach pattern $patterns {
     if {$vlower matches_regex $pattern} {
       if {$logHeader == 1} {
         set logMsg $val
       } else {
         set logMsg ""
       }
       log $logTag "log4j2 -- match header value $logMsg to regex $pattern"
       reject
       return
    }

    if {$nlower matches_regex $pattern} {
       if {$logHeader == 1} {
         set logMsg $header
       } else {
         set logMsg ""
       }
       log $logTag "log4j2 -- match header name $logMsg to regex $pattern"
       reject
       return
     }

   }
}
### end header processing ###
### Checking for POST and PUT
if {([HTTP::method] eq "POST") || ([HTTP::method] eq "PUT")} {
  if { [HTTP::header exists Content-Length] } {
    set contentLen [HTTP::header Content-Length]
    if {$contentMax != 0} {
      if {$contentMax < $contentLen} {
        set contentLen $contentMax
      }
```

```
          }
          HTTP::collect $contentLen
        } else {
          if {$contentMax != 0} {
            HTTP::collect $contentMax
          } else {
            HTTP::collect
          }
        }
      }
    }

    when HTTP_REQUEST_DATA {
      set payload [HTTP::payload]
      set rawPayload ""
      set origPayload $payload

      for {set i 0} {($i <= $decodeTrials) && ($payload ne $rawPayload) } {incr i} {
        set plower [string tolower $payload]
        foreach pattern $patterns {
          if {$plower matches_regex $pattern} {
             if {$logPayload == 1} {
               set logMsg $origPayload
             } else {
               set logMsg ""
             }
             log $logTag "log4j2 -- match payload $logMsg to regex $pattern"
             reject
             return
          }
        }

        set rawPayload $payload
        set payload [URI::decode $rawPayload]
      }


      if {$i >= $decodeTrials} {
        log $logTag "log4j2 -- exceeding max decode attempts $decodeTrials"
        reject
        return
      }

    }
```

SOFTWARE UPDATES

Not available.

VULNERABILITY DETAILS

The following table shares brief descriptions for the vulnerabilities addressed in this document.

Vulnerability ID	Description
CVE-2021-44228	Apache Log4j2 <=2.14.1 JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. In previous releases (>2.10) this behavior can be mitigated by setting system property "log4j2.formatMsgNoLookups" to “true” or it can be mitigated in prior releases (<2.10) by removing the JndiLookup class from the classpath (example: zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class).
CVE-2021-45046	It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allows attackers with control over Thread Context Map (MDC) input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, $${ctx:loginId}) or a Thread Context Map pattern (%X, %mdc, or %MDC) to craft malicious input data using a JNDI Lookup pattern resulting in a denial of service (DOS) attack. Log4j 2.15.0 restricts JNDI LDAP lookups to localhost by default. Note that previous mitigations involving configuration such as to set the system property

	`log4j2.noFormatMsgLookup` to `true` do NOT mitigate this specific vulnerability. Log4j 2.16.0 fixes this issue by removing support for message lookup patterns and disabling JNDI functionality by default. This issue can be mitigated in prior releases (<2.16.0) by removing the JndiLookup class from the classpath (example: zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class).
CVE-2021-45105	Apache Log4j2 versions 2.0-alpha1 through 2.16.0 (excluding 2.12.3) did not protect from uncontrolled recursion from self-referential lookups. This allows an attacker with control over Thread Context Map data to cause a denial of service when a crafted string is interpreted. This issue was fixed in Log4j 2.17.0 and 2.12.3.

RELATED LINKS

Ref #	General Link
[1]	Apache Log4j Security Vulnerabilities
[2]	NIST NVD CVE-2021-44228
[3]	NIST NVD CVE-2021-45046
[4]	NIST NVD CVE-2021-45105

ACKNOWLEDGEMENTS

None

MODIFICATION HISTORY

Revision	Date	Description
1.0	2021-12-14	Initial Publication
1.1	2021-12-20	Changed aGalaxy from not affected to being further investigated. Added AFLEX v1.0 mitigation policy.
2.0	2021-12-27	Update patterns for CVE-2021-45105 ctx:<name> case, raise decodeTrials to 5
2.1	2022-01-31	Update aGalaxy exposed and resolved releases