Vulnérabilités Log4j

CVE-2021-44228 - CVE-2021-45046 - CVE-2021-45105

RÈGLES DE DÉTECTION

TLP:WHITE

Version : 1.0.6
Date d'enregistrement : 07/01/2022
Nombre de pages : 10

Table des matières

1 Contexte

Plusieurs vulnérabilités ont été découvertes dans la bibliothèque de journalisation Apache log4j. Cette bibliothèque est très souvent utilisée dans les projets de développement d'application Java/J2EE ainsi que par les éditeurs de solutions logicielles sur étagère basées sur Java/J2EE.

La première de ces vulnérabilités, immatriculée CVE-2021-44228 et appelée « Log4Shell », permet à un attaquant de provoquer une exécution de code arbitraire à distance s'il a la capacité de soumettre une donnée à une application qui utilise la bibliothèque log4j pour journaliser l'évènement. Cette attaque peut être réalisée sans être authentifié, par exemple en tirant parti d'une page d'authentification qui journalise les erreurs d'authentification.

Des preuves de concept ont déjà été publiées. **Cette vulnérabilité fait désormais l'objet d'une exploitation active, et le CERT-FR a connaissance de compromissions par le biais de cette vulnérabilité**.

Pour plus d'information sur ces vulnérabilités ainsi que les mesures à prendre, veuillez-vous référer à l'alerte du CERT-FR [1].

2 Éléments de détection

L'ANSSI a développé des règles de détection pour l'outil Suricata afin de prevenir ou de détecter des compromissions liées à la vulnérabilité CVE-2021-44228. Bien qu'ayant été développées pour la CVE-2021-44228, ces signatures peuvent également prévenir ou détecter l'exploitation des vulnérabilités CVE-2021-45046 et CVE-2021-45105.

Les schémas de détection proposés dans la suite de ce document ne sont pas uniquement utilisables avec l'outil Suricata. En effet, les expressions régulières de ces schémas de détection peuvent être utilisées directement pour rechercher des motifs de compromission dans vos journaux. Elles peuvent aussi être utilisées sur un pare-feu applicatif (afin de bloquer les requêtes HTTP contenant les injections), sur un système de détection et de prévention d'intrusions ou sur un serveur *proxy* filtrant les flux à destination d'Internet.

Ces règles sont divisées en deux groupes :

- Les règles du groupe 1 ont pour but la détection d'injection de contenu vers des composants Log4j vulnérables.
- Les règles du groupe 2 ont pour but la détection d'exploitations de la vulnérabilité au travers d'interfaces Java comme JNDI ou JRMI.

2.1 Règles du groupe 1 : Détection d'injections liées à Log4Shell

Les règles 1357020 à 1357044 visent à détecter les tentatives d'injection liées à Log4Shell par la présence de motifs caractéristiques. Ces règles sont appliquées sur plusieurs protocoles pour maximiser les chances de détection. Les règles sur HTTP permettent de couvrir les injections transformées par HTTP que les règles sur TCP ne détecteraient pas (URI-encoding, etc.).

Ces règles peuvent produire un grand nombre de résultats dans le contexte actuel, étant donnée la fréquence des scans réalisés sur l'ensemble d'Internet pour la vulnérabilité Log4Shell. Les règles sur les variantes offusquées (1357040 à 1357044) peuvent générer des faux-positifs sur du contenu binaire (par exemple une photo JPEG) car le nombre d'octets signés est relativement faible (4) et il est donc probable que ce motif soit présent dans des flux légitimes au-delà d'un certain volume de trafic.

2.1.1 Règles signant les injections CVE-2021-44228 de type JNDI non offusquées

- 1357020 - `${jndi:` sur TCP
- 1357021 - `${jndi:` sur UDP
- 1357022 - `${jndi:` dans HTTP URI
- 1357023 - `${jndi:` dans HTTP Header
- 1357024 - `${jndi:` dans HTTP Body (requête)

2.1.2 Règles signant les injections CVE-2021-44228 utilisant des motifs de lookup

Les *lookups* contenant les motifs suivants : `lower`, `upper`, `sd`, `main`, `jvmrunargs`, `date`, `ctx`, `sys`, `bundle`, `marker`, `java`, `event`, `env`, `log4j`, `web`, `docker`, `kubernetes`, `spring`, `base64` sont susceptibles d'être utilisés pour l'offuscation de l'injection ou pour l'exfiltration de données.

- 1357030 - *lookup* sur TCP
- 1357031 - *lookup* sur UDP
- 1357032 - *lookup* dans HTTP URI
- 1357033 - *lookup* dans HTTP Header
- 1357034 - *lookup* dans HTTP Body (requête)

2.1.3 Règles signant les injections CVE-2021-44228 offusquées de type ${...${

- 1357040 - `${...${` sur TCP
- 1357041 - `${...${` sur UDP
- 1357042 - `${...${` dans HTTP URI
- 1357043 - `${...${` dans HTTP Header
- 1357044 - `${...${` dans HTTP Body (requête)

Ces signatures risquent de générer un volume important de faux-positifs. L'utilisation des signatures des autres sections peut permettre la levée de doute (la plupart des injections obfusquées utilisent aussi des *lookups* Log4j2). C'est pour cette raison qu'il convient de les déclarer en dernier.

2.2 Règles du groupe 2 : Détection d'exploitations utilisant l'interface JNDI

Les règles 1357000 à 1357014 visent à détecter les flux sortants correspondant à une exploitation réussie de la vulnérabilité, en signant trois protocoles disponibles via JNDI pour injecter des objets ou des classes Java dans l'application vulnérable. La règle 1357000 signe un usage par JNDI du protocole IIOP et ne devrait provoquer quasiment aucun faux positif

car l'usage de ce protocole est marginal dans l'écosystème Java. Les règles 1357003 et 1357004 signent le téléchargement d'objets Java sur LDAP et ne devraient pas non plus produire de faux positifs : il s'agit d'un usage peu courant de LDAP dans l'écosystème Java et une application Java bien conçue ne « déserialiserait »pas des données dont elle ne peut garantir ni l'intégrité ni la provenance en l'absence d'un transport de type TLS et d'une bonne gestion de certificats. La règle 1357007 signe le protocole JRMI qui est utilisable via JNDI, mais ce protocole étant couramment employé dans l'écosystème Java, il est probable que des alertes soient levées concernant des applications légitimes utilisant ce protocole. La présence de flux JRMI vers Internet est cependant un bon marqueur de compromission.

Les règles 1357010 à 1357014 visent à détecter dans des flux LDAP la présence d'objets possédant des attributs interprétables par JNDI et permettant de télécharger directement ou indirectement une charge potentiellement malveillante. Ces règles peuvent lever des faux positifs lorsque les noms d'attributs font partie de critères de recherche et non d'un objet dans un résultat de recherche, mais la présence de ces alertes dans un flux sortant vers Internet est là aussi un bon marqueur de compromission.

2.2.1 Règles signant une probable exploitation d'injections JNDI

- 1357000 - IIOP : connexion avec marqueurs JNDI
- 1357003 - LDAP : données sérialisées Java RFC 2713
- 1357004 - LDAP : références sérialisées RFC 2713
- 1357007 - RMI : communication RMI avec requête sérialisée

2.2.2 Règles signant une potentielle exploitation d'injections JNDI sur LDAP

Attention : Ces règles peuvent génerer des faux positifs sur des recherches LDAP sans réponse exploitable.

- 1357010 - LDAP : attribut javaClassName
- 1357011 - LDAP : attribut javaObject
- 1357012 - LDAP : attribut javaSerializedObject
- 1357013 - LDAP : attribut javaMarshalledObject
- 1357014 - LDAP : attribut javaNamingReference

3 Utilisation des signatures

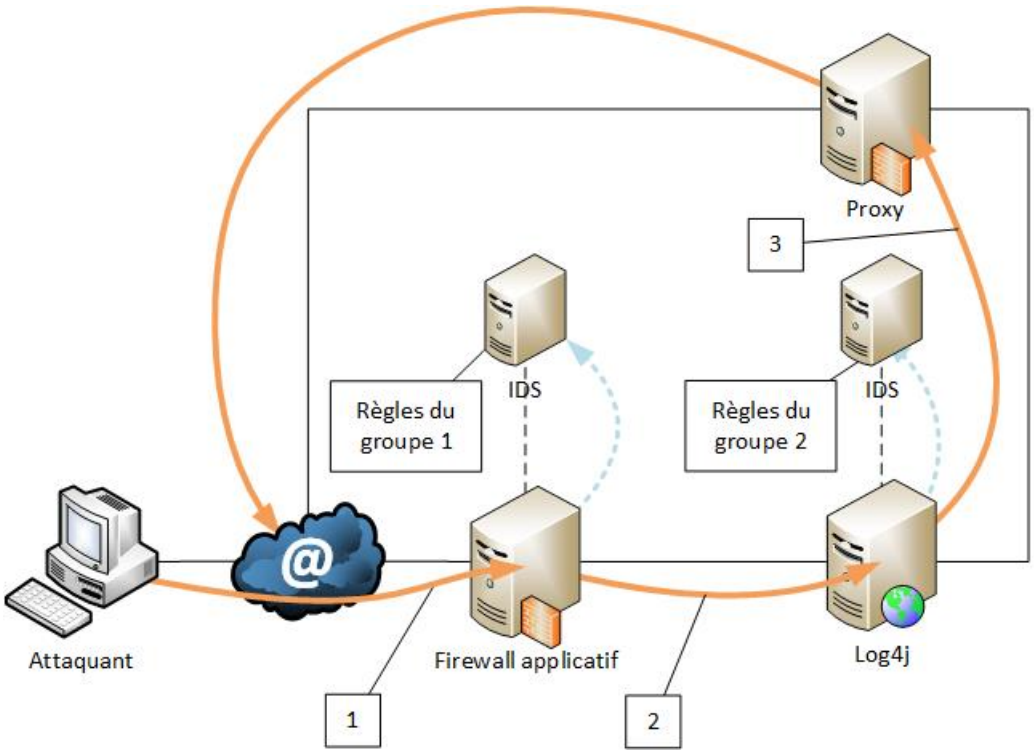

Figure 1 – Un exemple simplifié d'architecture réseau et d'exploitation de la vulnérabilité Log4Shell

(1) L'attaquant émet une requête HTTP à destination du composant Log4j vulnérable, cette requête transite sur un pare-feu applicatif qui la propage vers un détecteur d'intrusion. Les signatures à utiliser sont celles du groupe 1. Il est aussi possible de récupérer les expressions régulières contenues dans ce groupe afin de créer des règles de blocage spécifiques au pare-feu applicatif.

(2) La requête HTTP est transmise au composant Log4j vulnérable qui la propage vers un système de détection d'intrusions, les expressions à utiliser sont celles du groupe 2.

(3) Le composant Log4j vulnérable émet une réponse HTTP qu'il adresse à un serveur *proxy* configuré pour faire de l'inspection de paquets. Il est possible d'utiliser les expressions régulières contenues dans le groupe 2 afin de créer des règles de blocage spécifiques au *proxy*.

4 Description des règles Suricata

Les règles ont été testées sur un environnement Suricata 6 installé sur Debian Buster. **Attention** :

- les SIDs générés pour ces règles peuvent rentrer en conflit avec des SIDs locaux :

```
# SIDs for these rules start from 1357000 which is within the local use range.
# See http://doc.emergingthreats.net/bin/view/Main/SidAllocation, and change
    them if they conflict with yours.
```

- Afin d'éviter toute dégradation de performance, ces signatures doivent être qualifiées sur un environnement adéquat avant tout passage en production.

4.1 Groupe 1 : Détection d'injections

```
# CVE-2021-44228 - JNDI injection - Matching plain ${jndi: over various protocols
alert tcp any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Plain␣TCP"; content:"|24
    ␣7B|jndi:"; target: dest_ip; classtype: string-detect; sid: 1357020; rev: 1;)
alert udp any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Plain␣UDP"; content:"|24
    ␣7B|jndi:"; target: dest_ip; classtype: string-detect; sid: 1357021; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Plain␣HTTP␣URI";
    content:"|24␣7B|jndi:"; http_uri; target: dest_ip; classtype: string-detect; sid:
    1357022; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Plain␣HTTP␣Header";
    content:"|24␣7B|jndi:"; http_header; target: dest_ip; classtype: string-detect;
    sid: 1357023; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Plain␣HTTP␣Request␣Body
    "; content:"|24␣7B|jndi:"; http_client_body; target: dest_ip; classtype: string-
    detect; sid: 1357024; rev: 1;)

# CVE-2021-44228 - Use of lookup in injection - Matching the most widespread lookup
    names
alert tcp any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Use␣of␣lookups␣TCP";
    content:"|24␣7B|"; pcre:"/\$\{(?:lower|upper|sd|main|jvmrunargs|date|ctx|sys|
    bundle|marker|java|event|env|log4j|web|docker|kubernetes|spring|base64|:):/";
    target: dest_ip; classtype: string-detect; sid: 1357030; rev: 1;)
alert udp any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Use␣of␣lookups␣UDP";
    content:"|24␣7B|"; pcre:"/\$\{(?:lower|upper|sd|main|jvmrunargs|date|ctx|sys|
    bundle|marker|java|event|env|log4j|web|docker|kubernetes|spring|base64|:):/";
    target: dest_ip; classtype: string-detect; sid: 1357031; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Use␣of␣lookups␣HTTP␣URI
    "; http.uri; content:"|24␣7B|"; pcre:"/\$\{(?:lower|upper|sd|main|jvmrunargs|date|
    ctx|sys|bundle|marker|java|event|env|log4j|web|docker|kubernetes|spring|base64|:)
    :/"; target: dest_ip; classtype: string-detect; sid: 1357032; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Use␣of␣lookups␣HTTP␣
    Header"; http.header; content:"|24␣7B|"; pcre:"/\$\{(?:lower|upper|sd|main|
    jvmrunargs|date|ctx|sys|bundle|marker|java|event|env|log4j|web|docker|kubernetes|
    spring|base64|:):/"; target: dest_ip; classtype: string-detect; sid: 1357033; rev:
     1;)
```

```
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Use␣of␣lookups␣HTTP␣
    Request␣Body"; http.request_body; content:"|24␣7B|"; pcre:"/\$\{(?:lower|upper|sd|
    main|jvmrunargs|date|ctx|sys|bundle|marker|java|event|env|log4j|web|docker|
    kubernetes|spring|base64|:):/"; target: dest_ip; classtype: string-detect; sid:
    1357034; rev: 1;)

# CVE-2021-44228 - Obfuscated injection - Matching the recursive ${..${ sequences
alert tcp any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Obfuscated␣TCP"; content
    :"|24␣7B|"; pcre:"/\$\{[␣-|~]*\$\{/"; target: dest_ip; classtype: string-detect;
    sid: 1357040; rev: 1;)
alert udp any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Obfuscated␣UDP"; content
    :"|24␣7B|"; pcre:"/\$\{[␣-|~]*\$\{/"; target: dest_ip; classtype: string-detect;
    sid: 1357041; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Obfuscated␣HTTP␣URI";
    http.uri; content:"|24␣7B|"; pcre:"/\$\{[␣-|~]*\$\{/"; target: dest_ip; classtype:
     string-detect; sid: 1357042; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Obfuscated␣HTTP␣Header"
    ; http.header; content:"|24␣7B|"; pcre:"/\$\{[␣-|~]*\$\{/"; target: dest_ip;
    classtype: string-detect; sid: 1357043; rev: 1;)
alert http any any -> any any (msg:"CVE-2021-44228␣injection␣-␣Obfuscated␣HTTP␣Request
    ␣Body"; http.request_body; content:"|24␣7B|"; pcre:"/\$\{[␣-|~]*\$\{/"; target:
    dest_ip; classtype: string-detect; sid: 1357044; rev: 1;)
```

4.2 Groupe 2 : Détection d'exploitations

```
# JNDI - IIOP request - Connection attempt over GIOP with JNDI markers (versions 1.0
    to 1.2)
alert tcp any any -> any any (msg:"JNDI␣IIOP␣-␣NamingContext"; content:"GIOP|01|";
    pcre:"/GIOP\x01[\x00-\x02].\x00/"; content: "NEO|00|"; content:"IDL:omg.org/
    SendingContext/CodeBase:1.0"; content:"IDL:omg.org/CosNaming/NamingContext:1.0";
    target: src_ip; classtype: bad-unknown; sid: 1357000; rev: 1;)

# JNDI - RFC 2713 - Java serialized stream in LDAP searchResponse which can be used to
     deliver payloads.
# Variant 1: Matching the LDAP representation of the "javaSerializedData" attribute
    name followed by the Java serialization magic (ACED0005).
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣Serialized␣stream␣in␣searchResult␣
    -␣javaSerializedData"; content:"|04␣12|javaSerializedData"; nocase; content:"|AC␣
    ED␣00␣05|"; target: src_ip; classtype: bad-unknown; sid: 1357003; rev: 1;)

# Variant 2: Matching the LDAP representation of the "javaReferenceAddress" attribute
    name followed by the Base64-encoding of the Java serialization magic (rO0AB*) in a
     way which tolerates whitespace (MIME decoder)
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣Serialized␣stream␣in␣searchResult␣
    -␣javaReferenceAddress"; content:"|04␣14|javaReferenceAddress"; nocase; pcre:"/r\s
    *O\s*0\s*A\s*B/"; target: src_ip; classtype: bad-unknown; sid: 1357004; rev: 1;)

# JRMI - Java RMI stream protocol v2 followed by Java RMI request
alert tcp any any -> any any (msg:"JRMI␣-␣Java␣RMI␣request"; content:"JRMI|00␣02␣4B|";
     content:"|50␣AC␣ED␣00␣05|"; target: src_ip; classtype: bad-unknown; sid: 1357007;
```

```
     rev: 1;)

# JNDI - RFC 2713 - Java attributes in LDAP which may lead to payload delivery.
# Matching the RFC 2713 mandatory attributes as they are represented in LDAP protocol.
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣attributes␣-␣javaClassName";
    content:"|04␣0D|javaClassName"; nocase; target: src_ip; classtype: bad-unknown;
    sid: 1357010; rev: 1;)
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣attributes␣-␣javaObject"; content:
    "|04␣0A|javaObject"; nocase; target: src_ip; classtype: bad-unknown; sid: 1357011;
     rev: 1;)
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣attributes␣-␣javaSerializedObject"
    ; content:"|04␣14|javaSerializedObject"; nocase; target: src_ip; classtype: bad-
    unknown; sid: 1357012; rev: 1;)
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣attributes␣-␣javaMarshalledObject"
    ; content:"|04␣14|javaMarshalledObject"; nocase; target: src_ip; classtype: bad-
    unknown; sid: 1357013; rev: 1;)
alert tcp any any -> any any (msg:"JNDI␣LDAP␣-␣Java␣attributes␣-␣javaNamingReference";
     content:"|04␣13|javaNamingReference"; nocase; target: src_ip; classtype: bad-
    unknown; sid: 1357014; rev: 1;)
```

Références

[1] Alerte 22 du CERT-FR : https://www.cert.ssi.gouv.fr/alerte/CERTFR-2021-ALE-022/

[2] Bulletin de sécurité Apache : https://logging.apache.org/log4j/2.x/security.html

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de la Tour-Maubourg - 75700 PARIS 07 SP

www.ssi.gouv.fr