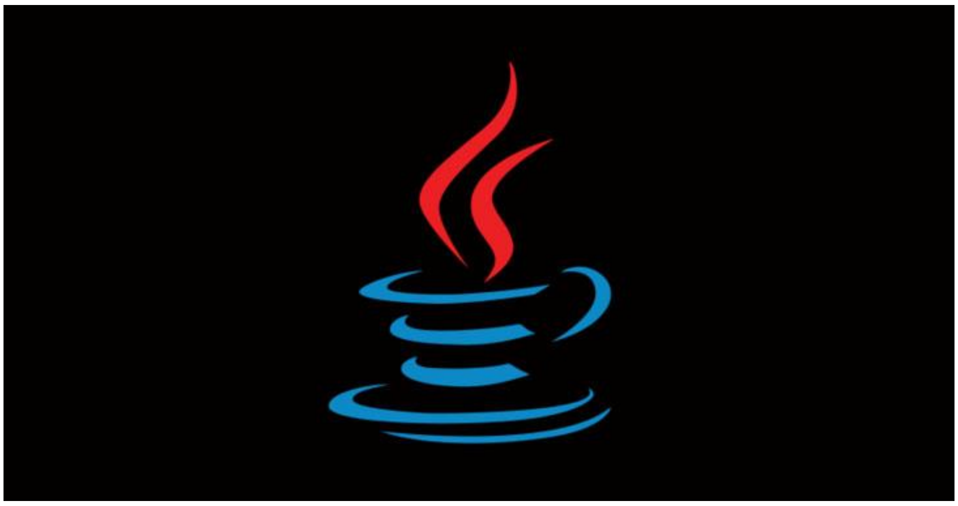
CVE-2022-21449- Psychic Signatures Vulnerability In Java

The vulnerability, identified as CVE-2022-21449, is a vulnerability caused by the incorrect implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA), which is widely used in some recent versions of Java.



With a CVSS score of 7.5, this flaw affects the following version of Java SE and Oracle GraalVM Enterprise Edition;

- Oracle Java SE: 17.0.2, 18
- Oracle GraalVM Enterprise Edition: 21.3.1, 22.0.0.2

It allows threat actors to forge TSL certificates and signatures, two-factor authentication codes, authorization credentials, and the like.

ECDSA, as described by ARS Technica, is an algorithm that uses Elliptic Curve Encryption principles to digitally authenticate messages. Because the keys it generates are smaller in size compared to RSA or other encryption algorithms, it often makes it ideal for use in standards such as FIDO's two-factor authentication, Security Assertion Markup Language, OpenID, and JSON.

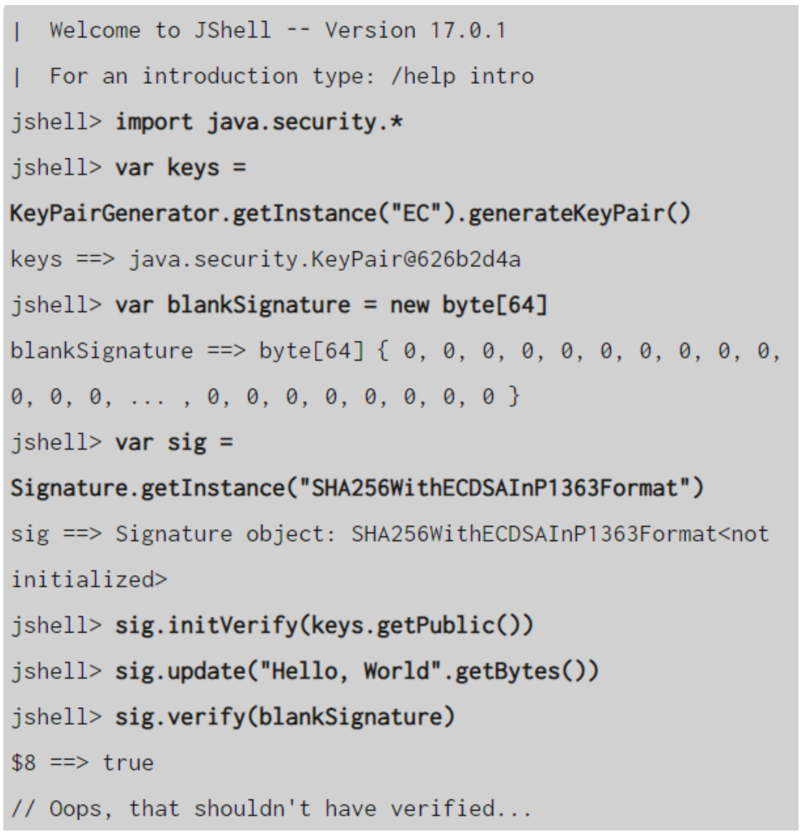
The vulnerability was first discovered by ForgeRock security firm researcher Neil Madden, who compared the vulnerability to a blank ID card from the BBC science fiction television series Doctor Who. In the series, the person looking at the ID card allows the Doctor to see what he wants him to see, even though the card is blank.

An attacker can easily forge SSL certificates and handshake types (allows communications to be interrupted and modified), authentication processes (like JWT), OIDC identity tokens, and even WebAuthn authentication messages, potentially allowing them to hijack communications and messages that should be encrypted .

The signature verification algorithm uses a mathematical equation consisting of the signer's public key, a hash of the message, and the two values (r and s) used in ECDSA signatures. To verify the ECDSA signature, the verifier checks an equation containing r, s, the signer's public key and a hash of the message. A signature is verified when ***both sides of the equation*** are equal.

Neither ***r*** nor ***s*** can ever be zero for the process to work correctly. This is because one side of the equation is ***r*** and the other side is ***r*** and multiplied by a value from ***s***. The malpractice introduced in Java 15 does not check if the ***r*** and ***s*** values are 0 and that's where the problem starts. If the values are both 0, it will cause both sides of the used equation to be zeroed. At 0 = 0, the signature is validated. This means that an attacker only needs to send a blank signature to successfully pass the verification check.

Below is the Jshell session of the vulnerable application created by Madden. This session validates the message and public key and accepts a null signature as valid:



```
|  Welcome to JShell -- Version 17.0.1
|  For an introduction type: /help intro
jshell> import java.security.*
jshell> var keys =
KeyPairGenerator.getInstance("EC").generateKeyPair()
keys ==> java.security.KeyPair@626b2d4a
jshell> var blankSignature = new byte[64]
blankSignature ==> byte[64] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, ... , 0, 0, 0, 0, 0, 0, 0, 0 }
jshell> var sig =
Signature.getInstance("SHA256WithECDSAInP1363Format")
sig ==> Signature object: SHA256WithECDSAInP1363Format<not
initialized>
jshell> sig.initVerify(keys.getPublic())
jshell> sig.update("Hello, World".getBytes())
jshell> sig.verify(blankSignature)
$8 ==> true
// Oops, that shouldn't have verified...
```

A ***PoC*** has been published by security researcher Khaled Nassar to demonstrate the Psychic Signatures Vulnerability in Java vulnerability. In this PoC, Nassar used a vulnerable client and a malicious TLS server.

Nassar has set up the valid SSL certificate chain for www.google.com with the ECDSA broadcast key on the malicious server. He also modified the crypto/ecdsa package with r=s=0 to provide an inappropriate signature. The vulnerable client accepts this invalid signature, allowing the rest of the TLS negotiation to continue.

First of all, if you are using Java 15 or later, please go and update to the latest version to get the fix for this issue. In case it is impossible to upgrade Java to the next version, it is recommended to use another signature algorithm. ECDSA variations that should not be used are given below.

NONEwithECDSA
SHA1withECDSA

SHA224withECDSA
SHA256withECDSA
SHA384withECDSA
SHA512withECDSA
SHA3-224withECDSA
SHA3-256withECDSA
SHA3-384withECDSA
SHA3-512withECDSA
NONEwithECDSAinP1363Format
SHA1withECDSAinP1363Format
SHA224withECDSAinP1363Format
SHA256withECDSAinP1363Format
SHA384withECDSAinP1363Format
SHA512withECDSAinP1363Format
SHA3-224withECDSAinP1363Format
SHA3-256withECDSAinP1363Format
SHA3-384withECDSAinP1363Format
SHA3-512withECDSAinP1363Format

It is recommended to use EdDSA or Ed25519 signature algorithms instead of these variations. You can refer to the ***Java documentation*** for a list of supported algorithms.

Organizations using any of the affected Java versions to verify signatures should give high priority to patching. It will also be important to follow advice from app and product manufacturers to see if any of their products have been made vulnerable. Although the CVE-2022-21449 threat appears to be limited to new Java versions, its severity is high enough to warrant caution.