

BsidesSF 2017 - []root (crypto, 250 pts)

Our guy inside e-corp was able to get that packet capture of their backend PKI you asked for. Unfortunately it seems they're using TLS to protect the modulus fetch. Now, I have been told that the best crackers in the world can do this in 60 minutes. Unfortunately I need someone who can do it in 60 seconds.

Note: Flag does not follow the "Flag:" format but is recognizable

e_corp_pki.pcapng

We are provided with a packet capture file containing a TLS exchange. Loading it up in Wireshark, we can quickly identify a TLS handshake, with a "server hello" message containing the server certificate, itself containing the RSA public key (modulus + public exponent):

```
Transmission Control Protocol, Src Port: 443, Dst Port: 49258, Seq: 1, Ack: 518, Len: 1597
Secure Sockets Laver
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
TLSv1.2 Record Layer: Handshake Protocol: Certificate
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
      Length: 1497
     Handshake Protocol: Certificate
         Handshake Type: Certificate (11)
         Length: 1493
Certificates Length: 1490
        Certificates (1490 bytes)
Certificate Length: 1487
            Certificate: 308205cb308203b2a0030201020209009e6e0daa0910fafb... (pkcs-9-at-emailAddress=pki@e-corp.com,id-at-c
               signedCertificate
                  version: v3 (2)
serialNumber: -
                                    ,
-7030666944331056389
                  signature (sha256WithRSAEncryption)
                  issuer: rdnSequence (0)
                  validity
                  subject: rdnSequence (0)
                  subjectPublicKeyInfo
                     subjectPublicKey:
                                           3082020902820201726f6f7400000000000000000000000
                        publicExponent: 31337
            algorithmIdentifier (sha256WithRSAEncryption)
```

We need to recover the private key by factoring the modulus. We used Fermat's factorisation method to recover the two prime factors (script here):

This quickly yielded values for p and q. Plugging those values into rsatool, we were able to reconstruct the server's private key:

python ./rsatool.py -p 345709341936068338730678003778405323582109317075021198605451259081268526297654818935

...1

Saving PEM as priv.key

The next step was to load this private key back into Wireshark to see decrypted TLS traffic. We could see a GET /modulus HTTP request with some form of ASCII art:

Wireshark · Follow SSL Stream (tcp HTTP/1.0 200 ok Content-type: text/plain 72:6f:6f:74:00:00:00:00:00:00:00:00:00:00:00: 00:00:00:00:00:1f:ff:fb:00:00:00:00:00:00:00 00:00:00:00:1f:ff:00:ff:fb:00:00:00:00:00:00: 00:00:00:00:1f:00:00:00:fb:00:00:00:00:00:00 00:00:00:1f:00:00:00:00:fb:00:00:00:00:00 00:00:00:1f:00:00:00:00:fb:00:00:00:00:00 00:00:00:1f:00:00:00:00:00:fb:00:00:00:00:00: 00:00:00:1f:00:00:00:00:00:fb:00:00:00:00:00:

Nice ASCII art key we thought... But then we looked closely to the first non-zero bytes towards the end: 66 6c 61 67 . This looks like ASCII for "flag"! And indeed:

>>> '66:6c:61:67:3a:77:68:65:6e:5f:73:6f:6c:76:69:6e:67:5f:70:72:6f:62:6c:65:6d:73:5f:64:69:67:5f:61:74:5f: 'flag:when_solving_problems_dig_at_the_roots_instead_of_just_hacking_at_the_leaves'

© 2017 GitHub, Inc. Terms Privacy Security Status Help



Contact GitHub API Training Shop Blog About