The State of SSL/TLS Certificate Usage in Malware C&C Communications

Mohamad Mokbel

Introduction

The nature of malware communications with its C&C server(s) has advanced over time, from using plain non-encrypted channels to using custom and standard symmetric (such as Blowfish, TEA, RC4, AES, DES, 3DES, XOR, ChaCha20, Salsa20) and asymmetric (such as RSA, DSA, ECC, DH) encryption algorithms and protocols (SSL/TLS) to hinder network inspection of such malicious traffic. Depending on the specifics of how a malicious encrypted channel is constructed, the engineering of detection rules changes accordingly, and requires different insight and capabilities.

Of particular interest is malware's increased adoption of secure HTTP (HTTPS) protocol comparted to unencrypted HTTP. This is no surprise as according to the Google Transparency Report, HTTPS adoption increased dramatically between 2014 and 2021, reaching more than 90% in certain countries. As more legacy systems and older services are phased out, newer systems and services will have the necessary modern encryption and protocols to support the latest versions of SSL/TLS.

Over the last six years there has been an increased shift by malware authors to secure their C&C communications using the SSL/TLS protocol to stymie detection and blend in with normal traffic. This shift is noticeable in commodity malware, as well as in APT type attacks. This is also seen in red teaming tabletop exercises to test the capabilities of different detection security layers, using frameworks such as Cobalt Strike, Metasploit and Core Impact, among others.

While a given malware variant uses HTTPS, other variants of the same family could be using HTTP, depending on the malware's configuration options. Therefore, detection rules must account for both scenarios whenever possible. Moreover, the adoption of SSL/TLS in malware is not restricted to the HTTPS protocol; other protocols, including SMTP and custom TCP protocols, were also found using SSL/TLS.

Without going into too much detail on how the SSL/TLS protocol works, it suffices for the purpose of this article that during the SSL/TLS handshake process, the server sends the client a digital/identity certificate (with the public key) for the client to use for encrypting the pre-master key, which the server receives and decrypts using its respective private key. It is the digital certificate that certifies and cryptographically authenticates ownership of the public key by a given entity/server. For the certificate to be valid, it must be issued and signed by a trusted third-party certificate authority (CA), as opposed to a self-signed certificate (excluding trusted root certificates). Such certificate is of leaf/end-entity type, meaning, it cannot be used to sign other certificates.

The public key infrastructure X.509 is the standard that defines the format of the public key certificates, which includes attributes such as:

```
Certificate:
        signedCertificate
            version:
            serialNumber:
            signature (sha256WithRSAEncryption)
                Algorithm Id:
            issuer: rdnSequence (0)
            validity
                notBefore: utcTime (0)
                notAfter: utcTime (0)
            subject: rdnSequence (0)
            subjectPublicKeyInfo
                algorithm (rsaEncryption)
                subjectPublicKey:
                    modulus:
                    publicExponent:
            extensions (optional):
```

- version
    - The certificate version number (v1(0x00), v2(0x01) or v3(0x02))
- serialNumber
    - A unique identifier for every certificate issued by the same CA. A non-negative integer
- signature
    - The algorithm used to sign the certificate (ex., sha256WithRSAEncryption)
- issuer
    - The name of the CA that issued the certificate using LDAP format
- validity (notBefore/From)
    - The certificate validation starting date and time
- validity (notAfter/To)
    - The certificate validation ending date and time
- subject
    - The name of the entity the CA issued it to using the LDAP format
- subjectPublicKeyInfo
    - RSA or ECC (different key sizes)
    - Public key
    - Exponent/parameters
- extensions
    - Depending on the version of the certificate, and in particular version 3(0x02), the certificate might include optional extensions such as CRL Distribution Points (CDP), Authority Information Access (AIA), Enhanced Key Usage (EKU) and Certificate Policies, among others

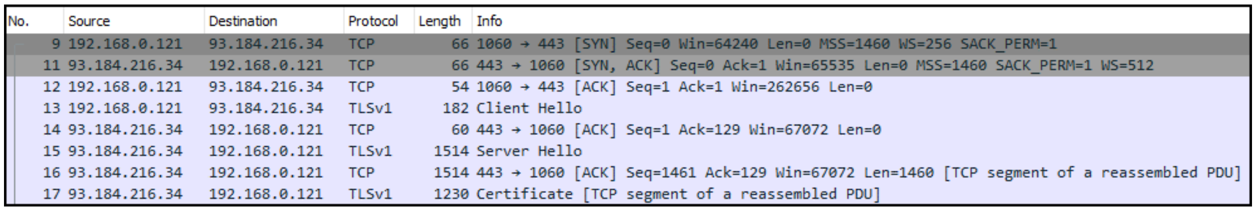
The certificate thumbprint, also known as fingerprint, is a cryptographic hash of the entire certificate, including the signature part. However, it is not part of the actual certificate attributes.

In this technical brief, we'll feature various malware families that communicate with their C&C server(s) over SSL/TLS, from the standpoint of the various attributes in the certificate. Additionally, we'll highlight certain interesting observations in the metadata of those attributes, over the lifetime of a given malware family. Moreover, whenever possible, detection techniques will be presented to recognize some of those "malicious" certificates on the network. Detecting malware C&C communications at the certificate level is a crucial step for stopping malware from communicating with its C&C server(s) at the earliest point. This is even more important in the absence of SSL/TLS man-in-the-middle proxy-based decryption, or if the original plain non- SSL/TLS encrypted traffic is not filterable.

It is important to note that in version 1.3 of the TLS protocol, all traffic after the server Hello message is encrypted, including the certificate. This poses a serious challenge for any inline fingerprinting of the certificate; however, this could be solved by proactively querying the server for the certificate for further inspection.

Like OpenSSL that could generate and parse certificates, Wireshark, the famous network protocol analyzer, has a full-featured dissector for the certificate that could be exported from the packet capture to a binary file, as DER encoded, under any of the file extensions, .der, .cer, and .crt.

Figure 1 shows TLSv1 packet dissection by Wireshark. Note frame No. 17, which contains the actual certificate received from the server, handshake type, certificate (0x11).



```
No.  Source          Destination     Protocol  Length Info
   9 192.168.0.121   93.184.216.34   TCP           66 1060 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
  11 93.184.216.34   192.168.0.121   TCP           66 443 → 1060 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM=1 WS=512
  12 192.168.0.121   93.184.216.34   TCP           54 1060 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
  13 192.168.0.121   93.184.216.34   TLSv1        182 Client Hello
  14 93.184.216.34   192.168.0.121   TCP           60 443 → 1060 [ACK] Seq=1 Ack=129 Win=67072 Len=0
  15 93.184.216.34   192.168.0.121   TLSv1       1514 Server Hello
  16 93.184.216.34   192.168.0.121   TCP         1514 443 → 1060 [ACK] Seq=1461 Ack=129 Win=67072 Len=1460 [TCP segment of a reassembled PDU]
  17 93.184.216.34   192.168.0.121   TLSv1       1230 Certificate [TCP segment of a reassembled PDU]
```

Figure 1. TLSv1 Handshake

Figure 2 shows Wireshark's dissection of the domain "example.com" trusted certificate. This makes it very easy to browse all the certificates' items in a structured and hierarchical manner. Moreover, the hex dump window (not shown) alongside the TLS certificate protocol dissection window provides the perfect combination for identifying key bytes from the certificate, for crafting a detection logic.

```
TLSv1 Record Layer: Handshake Protocol: Certificate
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 3978
  Handshake Protocol: Certificate
    Handshake Type: Certificate (11)
    Length: 3974
    Certificates Length: 3971
    Certificates (3971 bytes)
      Certificate Length: 1753
      [...]
        signedCertificate
          version: v3 (2)
          serialNumber: 0x0fbe08b0854d05738ab0cce1c9afeec9
          signature (sha256WithRSAEncryption)
              Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
          issuer: rdnSequence (0)
                RDNSequence item: 1 item (id-at-countryName=US)
                RDNSequence item: 1 item (id-at-organizationName=DigiCert Inc)
                RDNSequence item: 1 item (id-at-commonName=DigiCert TLS RSA SHA256 2020 CA1)
          validity
              notBefore: utcTime (0)
                  utcTime: 2020-11-24 00:00:00 (UTC)
              notAfter: utcTime (0)
                  utcTime: 2021-12-25 23:59:59 (UTC)
          subject: rdnSequence (0)
                RDNSequence item: 1 item (id-at-countryName=US)
                RDNSequence item: 1 item (id-at-stateOrProvinceName=California)
                RDNSequence item: 1 item (id-at-localityName=Los Angeles)
                RDNSequence item: 1 item (id-at-organizationName=Internet Corporation for Assigned Names)
                RDNSequence item: 1 item (id-at-commonName=www.example.org)
          subjectPublicKeyInfo
              algorithm (rsaEncryption)
                  Algorithm Id: 1.2.840.113549.1.1.1 (rsaEncryption)
             subjectPublicKey: 3082010a0282010100bafceeccca0a08ff0e931db3be0b9c0396229eb14f10ae5140fd53…
                modulus: 0x00bafceeccca0a08ff0e931db3be0b9c0396229eb14f10ae5140fd535fb3c461402804ee…
                publicExponent: 65537
          extensions: 10 items
     algorithmIdentifier (sha256WithRSAEncryption)
         Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
     Padding: 0
    encrypted: a72a10305cb86b7a1bf86638f6e9a00ad5138282f8658957a5b8eb13291d846cecfbe305…
```

Figure 2. Wireshark Dissection of example.com certificate

Analysis

In this section, we'll go through various malware families that use the SSL/TLS protocol with the goal of uncovering some interesting characteristics of different attributes in the certificates and self-signed certificates.

We start off by presenting Table 1, which shows the percentage of different protocols used by malware that communicate over SSL/TLS. These were found between May 2019 and June 2021, for a total of 264 unique malware families (categories: backdoor, worm, Stealer, ransomware, and downloader). Each one of these malware families is fully reverse engineered, and an IPS/IDS signature/filter was written for detecting the decrypted traffic. Moreover, this is only a subset of filters written for such malware.

Note that this only accounts for malware with filterable traffic and does not imply that only 264 malware families communicate over SSL/TLS were seen in-the-wild between those dates.

	ip.tcp	**ip.tcp.http**	**ip.tcp.smtp**
Total	8 (3%)	177 (67%)	79 (~30%)

Table 1. A set of unique malware families that communicate over SSL/TLS across different underlying protocols

- ip.tcp indicates a custom protocol on top of TCP
- ip.tcp.http indicates malware communicating over the HTTP protocol to the attacker's own HTTP server, or to other popular and legitimate services that use the HTTP protocol using their respective REST API, such as Discord, Telegram, Slack, Pastebin, and GitHub
- ip.tcp.smtp indicates malware that exfiltrate data over the SMTP protocol to any of the email services, such as gmail.com, mail.ru, outlook.com, and yandex.com. Gmail is the most widely used for exfiltrating data

It is no surprise that most of the C&C communications use the HTTP protocol since it blends perfectly with legitimate traffic.

Table 2 breaks down the same list of malware, per the platform it targets and the type. Windows is the targeted platform, with various programming languages.

Windows		**Linux**		**Cross-platform**		**Other**
Type	Total	Type	Total	Type	Total	JS (Browser Skimmer): 1
MSIL (.NET)	191	ELF	5	Python	4	
PowerShell	6	Shell	5			
Win32 (C/C++/Delphi)	27	**Sum**	10			
Win64 (C/C++/Delphi)	14					
Visual Basic for Applications (VBA)	1					
VBScript (VBS)	2					
JavaScript (JS)	4					
Python	4					
Sum	249 (94.3%)					

Table 2. Target platforms of malware that use SSL/TLS

For a TLS certificate to be valid and trusted, it must be issued by a trusted CA, such as Sectigo SSL, DigiCert SSL, Symantec SSL, RapidSSL, GeoTrust SSL, Thawte SSL, ZeroSSL or Let's Encrypt. While all recognized CAs charge a fee to acquire a legitimate certificate, Let's Encrypt, a non-profit CA, and ZeroSSL, do it for free, in an automated fashion, with no major oversight on the entity requesting it, with the caveat that the certificate is only valid for 90 days. Therefore, the certificate needs to be renewed after 90 days for it to be valid.

The SSL Blacklist (SSLBL) is a project of abuse.ch that aims to detect malicious TLS certificates used by botnet C&C servers. It tracks close to 100 malware families, but it doesn't provide the actual certificate nor enough metadata to uncover certain patterns that might emerge after juxtaposing it with different fields from different CAs. To get the actual certificate, we created a tool that queries Censys and crt.sh certificate search engines and aggregate all data in one CSV file. The results show some interesting patterns that are worthy of documentation.

As of June 29, 2021, only 1,767 out of 4,093 certificates are available on those search engines. Malware families include BitRAT, ServHelper, Cobalt Strike, AsyncRAT, Gozi, BitRAT, AceRAT, Trickbot, IcedID, Upatre, Vawtrak, Zeus, OrcusRAT, Zeus and NanoCore, among others.

The following are some observations:

- 1,067 (~60.4%) of 1,767 are self-signed certificates

    a. Only 96 (~5.4%) of 1,067 are based on version 1 of the x.509 certificate
        i. The majority of them having the Common Name (CN) set to "localhost", and Organization set to either "MyCompany Ltd." or "Internet Widgits Pty Ltd".
        ii. Malware families that use this version include Zeus, Gootkit, Shifu, Quakbot, and others, dating back to 2015-2017. For 2020, only ZLoader and a variant of Dridex use this version.

    b. 982 (92%) of 1,067 are based on version 3 of the x.509 certificate.
        i. Some malware families that use this version include TorrentLocker, IcedID, Gozi, Dridex, PandaZeuS, TrickBot, AsyncRAT, Malware, Vawtrak, CobaltStrike, FindPOS, Gootkit, Qadars, BuerLoader, BitRAT, OrcusRAT, Chthonic, BazaLoader, ZLoader, Ostap, Quakbot, QuasarRAT, RockLoader, ZeuS, NanoCore, GuLoader and Adwind.

What stands out from those self-signed certificates is the possibility to detect such malicious traffic due to the use of unique identifying values in the Subject field. For example, in the case of AsyncRAT, and in most of the variants, the Subject Common Name is set to "AsyncRAT Server" or "AsyncRAT Server CA". In the case of BitRAT, the Subject Common Name is set to "BitRAT".

Table 3 shows the certificate Subject Distinguished Name (DN) field values in LDAP format for different malware families. Such field includes identifying Relative Distinguished Names (RDNs) of the owner of the certificate, such as Country Name (C), certificate owner's Common Name (CN), Organization (O), Organization Unit (OU), Locality (L), and State/Province (ST), among others.

Malware Family	Subject (DN)
TorrentLocker	• C=US, ST=Denial, L=Springfield, O=Dis
IcedID	• CN=localhost, C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=XX, L=Default City, O=Default Company Ltd
Gozi	• C=XX, ST=1, L=1, O=1, OU=1, CN=* • CN=localhost, C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=XX, L=Default City, O=Default Company Ltd
PandaZeus	• C=XX, L=Default City, O=Default Company Ltd • domain.com/O=My Company Name LTD./C=US • C=IT, ST=Some-State, O=Internet Widgits Pty Ltd
Trickbot	• C=GB, ST=London, L=London, O=Global Security, OU=IT Department, CN=example.com • C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=XX, L=Default City, O=Default Company Ltd • CN=localhost.localdomain • C=AU, ST=3t2t3rgeg, L=2ehsdgsdfxjh, O=3wrwsts, OU=wefwstwe645gfhuy, CN=fg2eq34df
AsyncRAT	• CN=AsyncRAT Server
Vawtrak	• C=xx, L=Default City, O=Default Company Ltd • C=ff, L=Default City, O=Default Company Ltd • C=ss, L=Default City, O=Default Company Ltd • C=aa, L=Default City, O=Default Company Ltd • C=zz, L=Default City, O=Default Company Ltd
FindPOS	• C=XX, L=Default City, O=Default Company Ltd
BuerLoader	• C=aa, ST=aa, L=a, O=Internet Widgits Pty Ltd • C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=XX, ST=1, L=1, O=1, OU=1, CN=* • C=XX, ST=, O= • C=XX, L=Default City, O=Default Company Ltd • C=XX, ST=C, L=W, O=Internet Widgits Pty Ltd
BitRAT	• CN=BitRAT
OrcusRAT	• CN=Orcus Server • CN=OrcusServerCertificate
Chthonic	• C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • C=aa, L=Default City, O=Default Company Ltd
QuasarRAT	• CN=Anony96 • CN=Quasar Server CA
NanoCore	• CN=unk
DcRAT	• CN=DcRat Server

Table 3. Certificate Subject RDNs for different malware families

Note that for the Subject, the values "C=AU, ST=Some-State, O=Internet Widgits Pty Ltd" and "C=XX, L=Default City, O=Default Company Ltd" are the default values used when generating a self-signed certificate with OpenSSL. As shown, it is unsurprising that multiple malware families share the same values, albeit with slight variations on the Country (C) field value.

The validity period of those certificates varies, from as little as one month up to a year (the most common, which is the default setting in OpenSSL), five years, 10 years, to as high as 99 years. Of interest is the certificate (SHA256-fingerprint: 76dae43fa4216639e9f5706affc255306528397b02f02915c6782902d1d34d40) used by FindPOS malware, which has the validity period in the negative, from “2015-11-19 14:51” to “1979-09-19 9:23”, which means it’s already expired. OpenSSL does not accept negative values when generating certificates (it is possible that older versions of OpenSSL might allow it). This is likely to have been altered at the binary level. A certificate with such anomaly should be flagged and investigated.

The lifetime of some of those certificates could span more than five years, or the lifetime of the malware family itself. For example, in the case of Gozi, the Subject “C=XX, ST=1, L=1, O=1, OU=1, CN=*” has been in use since 2018 till 2021, with 100s of variants that use it. Additionally, most of Gozi’s certificates serial number length is 8 bytes, and only very few have it at 20 bytes, which is the default length for OpenSSL. Moreover, in all the cases, Gozi’s validity period is set to 10 years.

In the cases of AsyncRAT, BitRAT, OrcusRAT, QuasarRAT, the serial number length is fixed to 15 bytes, for all variants. Most of the malware families have their certificates serial number of length 8 or 15 bytes, except for few such as IcedID that has it set to 4 bytes, Vawtrak to either 4 or 8 bytes and Ostap to 2 bytes. Table 4 shows variants of the malware families Quakbot and Gozi that have the serial number length of 1 byte.

Malware	Certificate SHA-1 Fingerprint	Serial Number
Quakbot	• 562e7f2f7b3d5913a6ca64f25854d131e56c4ff7 • 101b0f5b4f4e336a2e9cc82b8e00d397f4939e6b	• 0x00 • 0x01
Gozi	• 1d1b4f67f070df5cae8f39553978f0adb3abd0c0	• 0x00

Table 4. Malware with 1-byte length Serial Number

By comparison, there is no trusted certificate out of the 90k top WordPress sites that has a serial number length less than 8 bytes.

The following two certificates of Zeus (as shown in the graph below) stand out due to the x.509 certificate version used. It is set to 4, however there is no version 4 of the x.509 certificate. As shown in the graph below, OpenSSL highlights the version number as unknown in both cases. Note that the certificate version number starts at 0x00, with the following mapping: 0x00→1, 0x01→2, 0x02→3. It is very likely that the threat actor had manipulated the version value surgically, but for reasons unknown. Moreover, the serial number for the first certificate is actually “0xb305962f”, but OpenSSL (version 1.1.1k, released on March 25, 2021) displays it as a negative number. This is because the leading byte 0xb3 is >= 0x80, and should have been prefixed with a leading zero 0x00 in order to avoid incorrectly representing it as negative number. On Windows, the certificate viewer does not display it as a negative integer. Checking the certificates of the top 90k WordPress sites and the top 317k Alexa sites on the web show no such anomalies. This nonetheless makes detecting this certificate easy, along with the check on the serial number.

```
Certificate (sha256 fingerprint:
11f92d23039c199e82355b9a419fb88e40d8011e52902f1351303bd82d2c591b):
   Data:
      Version: Unknown (3)
      Serial Number: -1291479505 (-0x4cfa69d1)
      Signature Algorithm: md5WithRSAEncryption
      Issuer: CN = Cyxuzoidv
      Validity
         Not Before: Nov 24 19:01:13 2015 GMT
         Not After : Nov 23 19:01:13 2016 GMT
      Subject: CN = Cyxuzoidv
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            RSA Public-Key: (1024 bit)
            …
```

First 16-byte of the certificate
0000h: 30 82 01 9F 30 82 01 08 A0 03 02 01 **03** 02 04 B3

```
Certificate (sha256 fingerprint:
d54b3bff7196c2201a3fe60fac8aa7601175db092268227b8b72c33910fc9dc5):
   Data:
      Version: Unknown (3)
      Serial Number: 151965358 (0x090eceae)
      Signature Algorithm: sha1WithRSAEncryption
      Issuer: CN = Wureuzisen
      Validity
         Not Before: Feb  3 18:37:43 2016 GMT
         Not After : Feb  2 18:37:43 2017 GMT
      Subject: CN = Wureuzisen
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            RSA Public-Key: (1024 bit)
            …
```

First 16-byte of the certificate
0000h: 30 82 01 A1 30 82 01 0A A0 03 02 01 **03** 02 04 09

As shown in Table 5, all the malware that use self-signed certificates were found to be using the RSA public key type, with varying key sizes:

Key Size	Total Malware
1024	183 (17%)
2048	803 (75%)
3072	1 (0.09%) – only one malware • Variant of Adwind • Certificate sha-1 fingerprint: 51a405e1791e14af11208387348a1399e6e63195 • Serial number: 0x66548813
4096	80 (7.5%)

Table 5. Malware self-signed certificate public key size and type

696 of 1,067 self-signed certificates already expired as of August 6, 2021.

More than 50% of the malware SSL/TLS communications are carried over TLSv1.2, around 12% over SSLv3, 20% over TLSv1, and none over TLSv1.3, but that does not mean they do not exist. All the malware surveyed in the SSL Blacklist (SSLBL) are Windows based, and the version is dependent on the OS version, network library used, and the server the malware is communicating with.

When crafting detection logic for a certificate for an IPS/IDS system such as Snort or Suricata, it is important to keep in mind whether to use the system's decoder primitive to account for different attributes in the certificate, or to use, plain *content* match against a specific attribute. In most cases, Suricata's protocol detection is port agnostic, and has decent support for detecting certain attributes in the certificate via special keywords (ex., tls.cert_subject, tls.cert_issuer, tls.cert_serial, tls.sni and tls_cert_notbefore, among others), but not so for Snort, which is port dependent by default and does not provide special keywords for detecting any attributes in the certificate. This is important because not all malware use the default SSL/TLS 443 port number when communicating with their C&C server(s) over SSL/TLS.

Nonetheless, each of the certificate attributes are binary encoded and has a defined ASN.1 notation that follows the Type-Class Length Value (TLV) structure. For example, to detect the Organization Name attribute with the value "Microsoft" in the certificate Subject field, without enforcing the sequence of headers, you take these bytes |06 03 55 04 0A 13 09 4D 69 63 72 6F 73 6F 66 74|, which translate to:

```
X509AttributeType Type: IdAtOrganizationName (2.5.4.10) -> Bytes |06 03 55 04 0A|
X509AttributeValue Value: Microsoft -> Bytes |13 09 4D 69 63 72 6F 73 6F 66 74|
       - X520OrganizationName IdAtOrganizationName: Microsoft
        - AsnBerPrintableString PrintableString: Microsoft
         - AsnBerOctetString String: Microsoft
          - AsnBerInfo AsnOctetStringHeader:
           + AsnBerIdentifier AsnId: PrintableString type (Universal 19)
           + AsnBerLength AsnLen: Length = 9, LengthOfLength = 0
            AsciiString OctetStream: Microsoft
```

Certificate Pinning

Relying on a valid and trusted certificate from the server during TLS/SSL handshake is crucial and functions as a proof to the validity and identity of the server, for the client to trust it. However, such a setup is not enough to guarantee the server's authenticity, since the attacker could install a rogue root CA certificate on the user device or perform a man-in-the-middle attack to get the traffic unencrypted. Thus, why SSL/TLS certificate pinning is important in the client side to avoid such attacks.

One approach for certificate pinning is by embedding a list of trusted certificates in the client application itself, and at runtime during handshake, compare them against the server certificates, such that if a mismatch happens, the connection is aborted. Other approaches include comparing only the public key (stored hashed), serial number, or any other items in the certificate. Depending on what is exactly being pinned from the certificate in the client application, certificate pinning might sever the connection with the server, if the server certificate has been updated in ways the pinned certificate/items cannot account for.

In the case of malware, only a handful use certificate pinning, including IcedID, AsyncRAT, DcRAT, Vawtrak and PhantomNet. Since they all use self-signed certificates, with unique values in the Subject field, detection is still possible, while man-in-the-middle attack might not work.

- IcedID
    - Communicates over HTTPS
    - Uses a self-signed certificate

- In a nutshell, and as described by Checkpoint, the malware hashes the public key in the certificate and compares it against the serial number of the certificate. If it fails to match, the connection is terminated
- The length of the serial number for all variants that use such type of certificate pinning is 4 bytes, and the Subject field contains the value “CN=localhost, C=AU, ST=Some-State, O=Internet Widgits Pty Ltd”. Since the Subject value might not be unique to IcedID, checking the length of the serial number such that it is only 4 bytes, along with the Subject field value, will reduce the likelihood of false positive hits on other malware families that use the same Subject field

- AsyncRAT
  - Open source
  - Uses custom protocol over TCP. Latest versions encrypt traffic with SSL/TLS
  - Uses a self-signed certificate
  - Executing the builder for the first time, it asks for the “Certificate Name” to generate a new PKCS #12 type certificate under the filename “ServerCertificate.p12”, which contains the actual certificate and the private key.
    - The default certificate name is set to “AsyncRAT Server”
    - The certificate Subject uses only the Common Name (CN) field
    - Certificate validity is always from the date the certificate was generated until December 31, 9999 7:59:59 PM
    - Serial number is always 15 bytes in length, but value changes every time the certificate is generated
    - Uses version 3 of the x.509 certificate
    - Signature algorithm: sha512RSA
    - Public Key: RSA (4096 Bits). Value changes every time the certificate is generated
    - Public Key parameters: 05 00
  - The self-signed certificate is stored AES encrypted and base64 encoded in the malware sample
  - Once the malware is executed, it starts by RSA verifying the digital signature of the embedded certificate’s public key against a precomputed sha-256 hash value. If verification fails, it terminates itself
  - Once communication with the server is established, and the certificate is received from the server, it compares it against the stored certificate, and in case they are not equal, the malware terminates itself

- DcRAT
  - Open source.
  - This RAT is based on the source code of AsyncRAT and uses the same certificate validation techniques
  - The only difference is that the default certificate Subject Common Name (CN) is set to “DcRat Server”

- Vawtrak
  - First, it checks the certificate's Common Name such that the sum of all the characters (in hex) except for the last character before the first occurrence of the '.' character, modulo 0x1A + 0x61, is equal to the last character, as shown in the following decompiled output:

```
INTERNET_CERTIFICATE_INFO cert_info; // [esp+14h] [ebp-28h] BYREF
dwBufferLength = 40;
is_valid = 0;

if ( InternetQueryOptionA(hInternet,
INTERNET_OPTION_SECURITY_CERTIFICATE_STRUCT, &cert_info, &dwBufferLength) )
{
  is_valid = verify_cn(cert_info.lpszSubjectInfo);
  LocalFree(cert_info.lpszSubjectInfo);
  LocalFree(cert_info.lpszIssuerInfo);
  LocalFree(cert_info.lpszProtocolName);
  LocalFree(cert_info.lpszSignatureAlgName);
  LocalFree(cert_info.lpszEncryptionAlgName);
  if ( !is_valid )
    return 0;
}
…
BOOL __fastcall verify_cn(LPTSTR *cert_cn)
{
  unsigned __int8 sum;
  unsigned __int8 nchar;

  sum = 0; nchar = 0;
  while ( *cert_cn && *cert_cn != '.' )
  {
    sum += nchar;
    nchar = *cert_cn;
    cert_cn = (cert_cn + 1);
  }
  return sum % 0x1A + 0x61 == nchar;
}
```

  - Second, RSA verifies the signature hash of the certificate's Subject Key Identifier attribute value using the certificate's public key, as shown in the following decompiled output:

```
dwBufferLength = 4;
if ( !InternetQueryOptionA(hInternet,
INTERNET_OPTION_SERVER_CERT_CHAIN_CONTEXT, &pChainContext, &dwBufferLength)
  || !pChainContext )
...
pCertContext = chain_element->pCertContext;
 if ( pCertContext )
   is_skid_valid = verify_cert_skid(pCertContext);
--->
BOOL __thiscall verify_cert_skid(PCCERT_CONTEXT pCertContext)
{
  PCERT_INFO pCertInfo;
  BOOL result;
  char pvStructInfo[20];
  char pdata[128];
  BYTE pvData[128];
  DWORD pcbStructInfo;

  pcbStructInfo = 148;
  result = pCertContext
        && (pCertInfo = pCertContext->pCertInfo) != 0
        && pCertInfo->SubjectPublicKeyInfo.PublicKey.pbData
        && CryptDecodeObject(
             X509_ASN_ENCODING__PKCS_7_ASN_ENCODING,
             RSA_CSP_PUBLICKEYBLOB,
             pCertInfo->SubjectPublicKeyInfo.PublicKey.pbData,
             pCertInfo->SubjectPublicKeyInfo.PublicKey.cbData,
             0, pvStructInfo, &pcbStructInfo) && pcbStructInfo == 148
        && (pcbStructInfo = 128,
            CertGetCertificateContextProperty(pCertContext,
            CERT_KEY_IDENTIFIER_PROP_ID, pvData, &pcbStructInfo))
        && pcbStructInfo == 128 && verify_sig(pdata, 128u, pvData);

  return result;
}
```

Trusted Certificates

As shown in Table 6, Let's Encrypt CA tops the list in terms of malware that use certificates generated by it, with Gozi alone claiming 150 of those certificates, followed by 61 for QNodeService, 29 for BazaLoader, and 28 for ZLoader. Note that we didn't observe a given certificate to have been renewed after the three-month validity period expiration date, for a given malicious domain. Instead, we found two different certificates to have been generated for the same domain, in an overlapping validity period, for a few malicious domains.

The data shows that multiple trusted CAs are vulnerable to issuing certificates to nonlegitimate entities.

Certificate Authority	***Total***
Let's Encrypt Authority X3	458
COMODO RSA Domain Validation Secure Server CA	41
RapidSSL CA	19
EssentialSSL CA	18
cPanel, Inc. Certification Authority	13
Others	26

Table 6. Trusted Certificate Authorities (CA) certificates used by different malware families

The topic of whether it is the CA's responsibility to combat malware and phishing sites is debatable. Let's Encrypt, notably, does not believe that certificate authorities should police the contents of domains. With TLS enabled by default across all domains, encryption would be an essential feature of all network traffic.

Table 7 shows the percentage of the trusted certificates public key size and type for some of the surveyed malware, with the RSA algorithm, size 2048 Bits, accounting for 462/542. ECC's adoption rate is very low (only 5.9%) compared to RSA, and the reason could be due to minimum operating system version and browser version requirements. Since all the surveyed malware for this table are Windows based, the minimum Windows version required for ECC is Windows Vista and Explorer 7. It is very likely for ECC adoption rate to increase in the future.

Key Size	**Total Malware (542)**
RSA - 2048	462 (85%)
RSA - 4096	48 (8.9%)
ECC - 256	32 (5.9%) • Let's Encrypt Authority X3; CloudFlare Inc ECC CA-2 • 28/32 are QNodeService malware.

Table 7. Trusted Certificates Public Key Size

Table 8 shows two sets of certificates that share the same serial number. Moreover, validating top 90k WordPress sites certificates show three sets of self-signed certificates that share the same serial number; they are: 0x00(7), 0x01(4), 0x02(3).

Malware	Serial Number	Certificate
Malware C&C	0xEB	• SHA1 fingerprint: 56a9aa61d3667c96a3ffeb941cff22ee9ea8da10 • Issuer (DN): O=Agency of Protected Certification Services, L=Phoenix, ST=AZ, C=US, CN=Agency of Protected Certification Services • Subject (DN): C=US, ST=AZ, O=Agency of Protected Certification Services, CN=www.hot-sex-tube.com • Validity: 2010-02-20 18:41/2015-02-19 18:41
Shifu C&C	0xEB	• SHA1 fingerprint: b821b99a945a8ab05a8518c4c1ec2f45f1ed6065 • Issuer (DN): O=Agency Protocols Management of Internet, E=info@apmi.com, L=Sacramento, ST=CA, C=US, CN=Agency Protocols Management of Internet SSL CA • Subject (DN): C=US, ST=CA, O=Agency Protocols Management of Internet, CN=bestylish.com, E=info@apmi.com • Validity: 2011-10-19 16:31/2016-10-17 16:31
Downloder-Bot C&C	0xEB	• SHA1 fingerprint: 84c4d012ae29024ed37d4680fae29e1663b3abcf • Issuer (DN): O=Self Certification Services, E=certs_division@sslslf.info, L=Jacksonville, ST=FL, C=US, CN=Self Certification Services SSL CA • Subject (DN): C=US, ST=FL, O=Self Certification Services, CN=*.sify.com, E=certs_division@sslslf.info • Validity: 2011-10-19 16:31/2016-10-17 16:31

Quakbot C&C	0x00	• SHA1 fingerprint: 562e7f2f7b3d5913a6ca64f25854d131e56c4ff7 • Issuer (DN): C=IN, ST=Karnataka, L=Bangalore, O=Multitech, OU=ODC, CN=localhost.localdomain • Subject (DN): C=IN, ST=Karnataka, L=Bangalore, O=Multitech, OU=ODC, CN=localhost.localdomain • Validity: 2006-04-24 10:38/2007-04-24 10:38
Gozi C&C	0x00	• SHA1 fingerprint: 1d1b4f67f070df5cae8f39553978f0adb3abd0c0 • Issuer (DN): C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • Subject (DN): C=AU, ST=Some-State, O=Internet Widgits Pty Ltd • Validity: 2016-12-22 16:01/2017-12-22 16:01

Table 8. Malware self-signed certificates with same serial number

Conclusion

As malware adoption rate of the SSL/TLS protocol increases with time, so does the need for defenders to come up with different detection rules. SSL/TLS encrypted traffic hinders detecting malware C&C communication traffic. However, a lot of malware use self-signed certificates with unique values for some of the attributes, which make detecting and blocking C&C traffic at the certificate level a viable option. Moreover, we’ve shown some interesting and exceptional anomalies in some of the fields in multiple certificates used by different malware families, which help in identifying the malware C&C server. We also noted the use of trusted and valid certificates issued by trusted CAs by several malware families, which shows that threat actors are actively searching for ways to avoid detection and blend more seamlessly with normal network traffic. This also calls for the need for better vetting and screening processes by CAs when issuing such certificates.

We’ve also documented a couple malware families that use certificate pinning in their code to prevent man-in-the-middle type attacks, or for interacting with the C&C server without the proper certificate handshake.

This work has resulted in the creation of multiple types (specific or generic) of IDS/IPS signatures/filters that attempt to detect different malware families at the certificate handshake level.

For future work, and for a more generic detection approach, a supervised classifier could be developed to detect malicious certificates in malware traffic, based on specific sets of features extracted from the certificate’s attributes.

We hope that we have contributed tangibly to this topic and shed light on some corner cases of different malware families’ usage of public certificates, helping defenders better secure their networks.

Acknowledgment

We would like to thank Cory Ford from Trend Micro Threat Research for reviewing this technical brief and providing valuable comments.