Using the Bro SSL analyzer

Introduction

Secure Sockets Layer (SSL) and the newer Transport Layer Security (TLS) protocols are some of the most important encryption protocols currently in use in the Internet. In the last months and years, there has been a huge amount of development in the space of SSL. A lot of cryptographic algorithms that were frequently used in the past are not considered to be secure anymore. New versions of TLS, especially 1.2 have seen a quite wide deployment. Furthermore, there have been a number of security problems that have shown the impact security issues in SSL/TLS can have on the Internet. One of the most memorable was the recent Heartbleed attack.
In this exercise, we will take a look at the features of the SSL/TLS analyzer in Bro. We will use the analyzer to write a number of scripts that check that the SSL traffic in the local network fulfills certain basic security requirements.

New features of the SSL analyzer in Bro 2.3

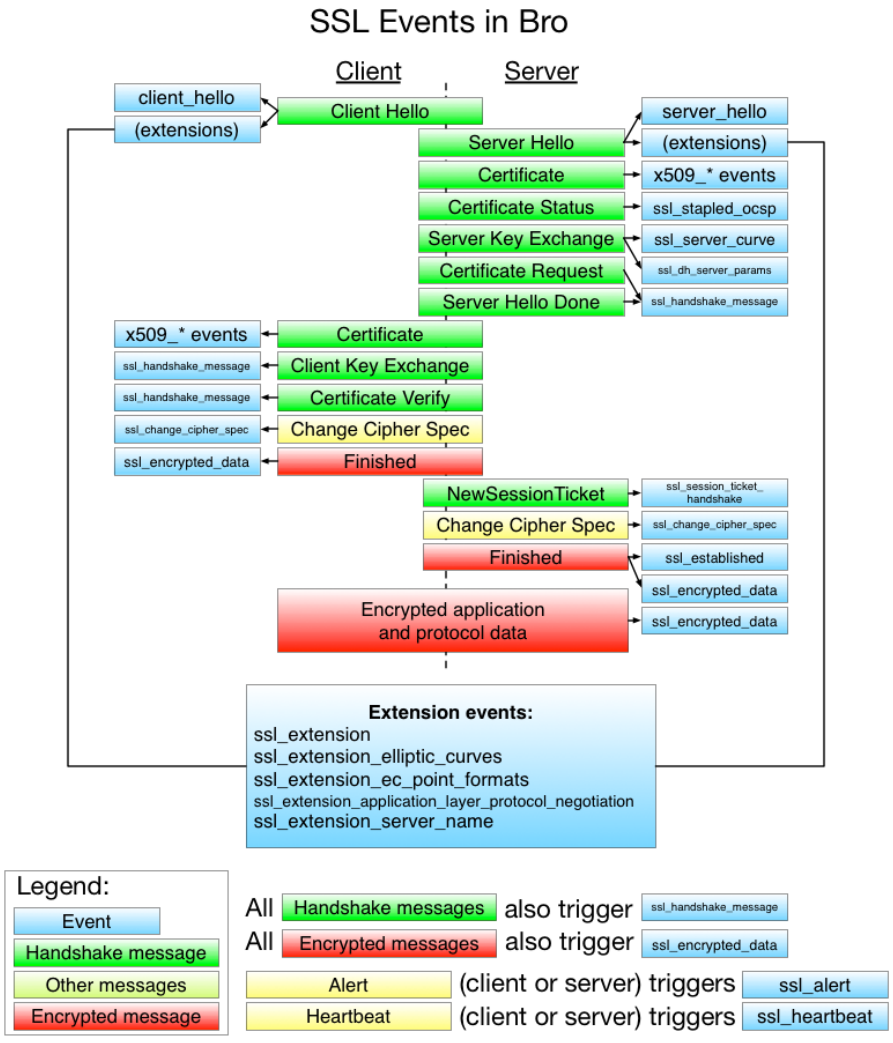
Before starting with the exercises, this section gives a short recap of the features of the SSL analyzer, with a special emphasis on new features added to Bro 2.3.
The Bro SSL/TLS Analyzer supports SSL 2.0 and 3.0 as well as TLS 1.0-1.2. From a user's perspective, there is no difference between SSL and TLS, the different protocol versions throw the exactly same events. For the remainder of this exercise, whenever we use SSL, we are either referring to SSL or to SSL and TLS. The SSL analyzer supports Bro's dynamic protocol detection (DPD). Thus, SSL connections are detected independently of the port on which they are established. Since Bro 2.3, STARTTLS support is automatically enabled for SMTP and POP3.
There is a number of new SSL events which have been added in the 2.3 release. Most of these will be covered in the next section. The new events cover features such as Diffie-Hellman (DH) key parameter size, Elliptic Curve (EC) choices, and several TLS extensions like Application Layer Next Protocol Negotiation (ALNP).
Bro also includes support for OCSP stapling, including verification of the OCSP messages. The X.509 Certificate analyzer was split from the SSL analyzer and also returns a wealth of new information about the cryptographic algorithms used in the certificates, as well as different extensions. The way Bro performs certificate validation also was significantly enhanced.
Bro 2.3 also features a few events for more basic analysis. There are events that are raised on every handshake message as well as every encrypted SSL record received.

The SSL protocol, Bro events and functions

This section gives a short recap on the SSL protocol and the different events that will be used in the remainder of this section. For more thorough documentation, see the SSL events documentation as well as the documentation of the SSL::Info record provided in c$ssl. For certificate related events, also see the the X509 analyzer Documentation as well as the documentation of the X509::Info record.
The setup of each SSL protocol starts with the SSL handshake, which is used to set up the cryptographic key material of the client and the server. Most of the SSL analysis in Bro is performed in the handshake phase, before the SSL content encryption kicks in.

Bro events in an SSL session

The following image lists all events that are supported by the Bro SSL analyzer and shows when they are triggered in the protocol flow.



Each SSL session begins with an SSL `client hello` message, in which the client sends information about the protocol versions it supports as well as the supported cryptographic algorithms.

```
event ssl_client_hello(c: connection, version: count,
    possible_ts: time, client_random: string, session_id: string
    , ciphers: index_vec);
```

In this event, the two main points of interest are the version as well as the list of ciphers. The conversion between the cipher numerals and the human-readable SSL versions is maintained in the table `SSL::version_strings`. The version represents the maximal TLS protocol version the client supports. Ciphers is a numeric vector of all the different cipher suites the client can support. Bro contains a mapping between the numerals and the string representation, which is maintained in the table `SSL::cipher_desc`.
More interestingly, the client may send a number of different extensions together with its client hello. Bro has one generic event which is generated for each extension that is sent either by the client after the client hello or by the server after the server hello:

```
event ssl_extension(c: connection, is_orig: bool, code: count,
    val: string);
```

`code` contains the number identifying the extension. It can be converted to a string using the `SSL::extensions` record. `val` contains the raw binary data contained in the extension.
For some extensions, Bro offers more specialized events that parse more information out of the extension data. The most interesting of these extensions is the Server Name Indication (SNI) which contains the name of the host the client wants to connect to:

```
event ssl_extension_server_name(c: connection, is_orig: bool,
    names: string_vec);
```

The server name is given as a vector of strings. In theory, a client can send a number of different server names. In practice, clients usually only send exactly one server name.
Another example of an extension where a specialized event is provided is the Application Layer Next Protocol Negotiation (ALNP). It is used by the client to signalize which protocol it understands. It is used to negotiate support for Google's SPDY or HTTP/2. In this case, the extension contains a list of protocol names supported:

```
event ssl_extension_application_layer_protocol_negotiation(c:
    connection, is_orig: bool, protocols: string_vec);
```

After the client has sent its `client hello` message, the server replies with the `server hello`.

```
event ssl_server_hello(c: connection, version: count,
    possible_ts: time, server_random: string, session_id: string
    , cipher: count, comp_method: count);
```

The server hello message contains two interesting pieces of information. The first is the SSL version used for the remainder of the session. This version has to be equal to or less than the maximum version supported by the client.
Second, the server hello contains the cipher suite that the server chose to be used for the remainder of the session. The server can only choose one of the algorithms supported by the clients.

As with the client hello, there is a huge number of extensions that can be sent by the server. Many extensions can be sent by both the server and the client. An example for this is the aforementioned ALNP. When the clients sends the extension, it contains a list of supported application layer protocols. The server can, in turn, reply with the exact same extension. In this case, it contains exactly one protocol, which is the application layer protocol the server chose to use.
Usually, the server also sends an X509 certificate when establishing the connection. The SSL analyzer automatically passes X509 certificates to the X509 analyzer, which raises an event for each certificate that is encountered.

```
event x509_certificate(f: fa_file, cert_ref: opaque, cert: X509
    ::Certificate)
```

The certificate is provided as an opaque reference to an object. This reference can be fed into several different functions, e.g. to verify the validity of a certificate. These will be explained below. The `cert` record contains parsed information about the certificate like the subject, the issuer, the cryptographic algorithms, the key length, etc. Note that a list of all certificates encountered in a connection will automatically be held in the connection record (see next section).
If supported by the client, the server can also choose to send a stapled OCSP reply to prove that the server certificate has not yet expired.

```
event ssl_stapled_ocsp(c: connection, is_orig: bool, response:
    string;
```

After the server and client handshake completes without errors, the SSL connection is established. Bro raises a dedicated event when this occurs:

```
event ssl_established(c: connection);
```

The c$ssl record

The SSL scripts provided by Bro as part of the installation automatically store a lot of the data provided by the different events in the connection-record and usually accessible as `c$ssl`. The full documentation of all fields that are provided in the default installation can be found here.
This record contains several pieces of useful information by default, including the full certificate chain sent from the server in `cert_chain` The certificate chain is especially relevant to this exercise.

SSL and X509 related functions

Bro also offers a few functions for handling X509 certificates. The most interesting of them is the `x509_verify` function, which tries to validate a given certificate chain against a list of root certificates.

```
function x509_verify(certs: x509_opaque_vector, root_certs:
    table_string_of_string, verify_time: time &default=
    network_time()): X509::Result
```

By default, Bro ships with the current Mozilla root store as used by Firefox, Thunderbird, etc. This store is automatically loaded into the SSL::root_certs table.
`x509_verify` returns a record containing the result of the certificate verification:

```
type X509::Result: record {
  ## OpenSSL result code
  result: int;
  ## Result as string
  result_string: string;
  ## References to the final certificate chain, if verification
       successful. End-host certificate is first.
  chain_certs: vector of opaque of x509 &optional;
};
```

`result_string` is `ok` when the validation succeeded. Errors that can occur are e.g. `expired` or `issuer certificate unknown` if no valid certificate chain can be built.
If the certificate validation succeeded, `chain_certs` contains all the certificates from the end-host certificate to the root-certificate that was used for validation.
There are a number of other utility functions that can be used with opaque certificate values. `x509_get_certificate_string` can be used to convert a Bro opaque certificate value back to its binary representation (default), or to a base64 encoded pem-form (when `pem = T`).:

```
function x509_get_certificate_string(cert: opaque of x509, pem:
     bool &default=F): string
```

Bro also offers several other functions, e.g. to support the verification of ocsp replies.

A) Survey the top used cipher-suites

The system administrators on your network are thinking about enforcing stricter security requirements on the servers and clients on their network. One of the steps they want to take is to enforce the use of secure cipher-suites in all SSL connections. As a first step, you are tasked to create an overview which SSL cipher suites you see in active use in your network.
For this exercise please use exercise_traffic.pcap..

Exercise 1

Level beginner
Generate the Bro logs for exercise_traffic.pcap. Note that one of the columns in ssl.log gives the cipher suite that was used in the SSL connection. Use bro-cut and other unix tools (like sort and uniq) to get the distributions of the ciphers that are used in different connections.
Level intermediate
Create a bro script that counts the use of all ciphers in a capture file. At the end of the script, output the number of times each cipher was used to the standard output. If you need further assistance, a skeleton script is provided in survey-skel.bro. Test your script using exercise_traffic.pcap.
Level advanced
Create a script that outputs the number of connections in which each cipher suite was seen in 15-minute intervals using the summary statistics framework. Either output the number of times each cipher was seen using a print statement, or write the resulting output into a new logfile ssl_ciphers.log. Test your script using exercise_traffic.pcap.

B) Check for weak ciphers

Another step in making your network more secure is to identify servers that are using certificates with unsafe key lengths.

Exercise 2

Level beginner
Bro ships with a policy script that can alert when encountering weak keys: policy/protocols/ssl/weak-keys.bro. Load the script, review the script documentation for weak keys and set it to alert when encountering certificates with a key_length < 2048 bits on any host. Try your script using exercise_traffic.pcap.

C) Certificate expiry

Another SSL pitfall to watch are expiring certificates in their network. Even big sites are sometimes bitten by one of their host certificates expiring.

Exercise 3

Level beginner
Bro ships with a policy script that can alert when encountering an SSL certificate that is close to expiring: policy/protocols/ssl/expiring-certs.bro. Load the script, review the script documentation for expiring certs and set it to alert when encountering a certificate expiring within the next 90 days on any host. Try your script using exercise_traffic.pcap.

D) Hostnames

As previously stated, modern clients send the Server Name Indication (SNI) extension, which contains the hostname of the server they want to connect to. As part of a survey, you will collect the hostnames the clients provide when connecting to your servers.

Exercise 4

Level beginner
Use the same ssl.log file as in the last exercise and extract all used server_names from it using unix tools.
Level intermediate Create a bro script that outputs all used ciphers when bro terminates. Look at the ssl_extension_server_name event to implement this. If needed, a skeleton is provided in sni_skel.bro
Level advanced Use the summary statistics framework to collect usage information about each SNI and output the information into a new log file every 15 minutes. Test your script using exercise_traffic.pcap.
As a second step, you will create a list of all hostnames for which the certificates on all servers are valid.

Exercise 5

Level advanced
Create a script that collects a list of all valid hostnames for all servers in the local network. When encountering a new hostname for a server, output a notice that a new hostname has been encountered. Test your script on using excercise_traffic.pcap.
Hint: Look for the first certificate in a connection, which is stored in csslcert_chain. Extract the common names from borth the subject (by searching for an element containing CN=) and in the subject alternative name extension, which is stored in the san field of X509::Info.

E) Blacklisting certificates

One of the changes in Bro 2.3 is that certificates encountered in connections are handled as files and forwarded to the File Analysis Framework. This means that Bro handles certificates like any other file. Hence, if you load a list of blacklisted SSL certificates into the Bro Intelligence Framework, it will alert when encountering the certificates in any connection.
For this example we will use the SSL blacklist provided by abuse.ch. The blacklist is provided as a CSV file. For this exercise, we provide this list in the Bro intelligence framework format as ssl-blacklist.txt.
If you want to recreate this file with newer data in the future, the conversion script is available on github.

Exercise 6

Level beginner
Create a bro-script that imports the ssl-blacklist.txt blacklist into the intelligence framework to trigger. Afterwards test your solution using ssl-blacklist.pcap
For documentation on how to load intelligence data into Bro, see the Intelligence Framework Documentation or last years Intelligence Framework exercises.

F) Certificate validation

A new policy has been enacted on your network and all SSL servers have to serve valid certificates.

Exercise 7

Level beginner
Bro provides a policy script which enables validation of all certificates in `policy/protocols/ssl/validate-certs.bro`. When it is loaded, the results of certificate validation are added to `ssl.log`.
Load the script and identify all servers using invalid certificate chains.
Level advanced Create a script that outputs a warning when an invalid certificate chain is encountered when accessing a (local) server. A skeleton is provided in verify-skel.bro.

Exercise 8

Level advanced
Change the last script to find all servers where the certificate chain is valid, but contains superfluous certificates which are not needed for validation.

G) OCSP stapling

One of the more problematic points of SSL/TLS and Certificates is certificate revocation. If the key of a certificate has been compromised (e.g., because a machine containing the public key has been hacked), the associated certificate has to be revoked.
Traditionally, certificate authorities distribute certificate revocation information either with Certificate Revocation Lists (CRLs) or via the Online Certificate Status Protocol (OCSP). Both of these approaches have the disadvantage that the client accessing a website either first has to download a potentially huge CRL list, or has to make an OCSP request to a (potentially slow) CA server.
One of the solutions to this problem is to use OCSP stapling. In this case, the SSL server sends a current OCSP response together with the X509 certificate when the connection is established. This approach eliminates the additional download or connection to the certificate authority.

Exercise 9

Level intermediate
Find all local servers that do not yet support OCSP stapling. You can test your script using ssl-blacklist.pcap, which contains several connections to servers that support / do not support ocsp stapling.
Hint: OCSP stapling has to be supported both by the server and client. First check if the client supports OCSP stapling by checking for the "status_request" extension being sent by the client. If the client supports OCSP stapling and the server sends a certificate, the server also has to send a stapled OCSP reply.
Optionally, a partially filles out skeleton for this exercise if provided in ocsp-exercise-skel.bro.