A Bro Script Case Study
• No deep detail now, just enough to understand basic constructs.
• Important to focus on script structure and data flow.
Script layout changes in 2.0
Important script directories.

Found at: <prefix>/share/bro/
• Everything is loaded by default.
  • Possible to disable with a Bro command line argument, but not recommended.
• The scripts are only meant to enable analyzers, collect state, generate protocol logs, and provide reusable frameworks and function libraries.
• base/ is not in the default $BROPATH!
policy/ directory

- Nothing here is loaded by default.
- This is where many of the detections that Bro does out of the box take place.
- Almost any functionality that doesn’t fit into base/ goes here.
site/ directory

- This is where local configuration goes.
- Files are not overwritten during installation.
- We include a “suggested” configuration in site/local.bro
- It’s mostly just a long list of @load statements.
SSL Base Scripts
Quick aside about module layout

- \_load\_.bro is an auto load file. We can now load directories.
- main.bro is a convention we use for consistency. There is no special language support for it.

Found at: <prefix>/share/bro/base/protocols/
Create the skeleton

```plaintext
module SSL;

export {
    redef enum Log::ID += { LOG };
    type Info: record {
    };
    global log_ssl: event(rec: Info);
}

redef record connection += {
    ssl: Info &optional;
};

event bro_init() &priority=5 {
    Log::create_stream(SSL::LOG, [$columns=Info, $ev=log_ssl]);
}

redef dpd_config += {
    [[ANALYZER_SSL]] = [$ports = ports]
};
```
Define the log

type Info: record {
    ts: time &log;
    uid: string &log;
    id: conn_id &log;
    version: string &log &optional;
    cipher: string &log &optional;
    server_name: string &log &optional;
    session_id: string &log &optional;
    subject: string &log &optional;
    not_valid_before: time &log &optional;
    not_valid_after: time &log &optional;
    cert: string &optional;
    cert_chain: vector of string &optional;
};
Create a helper function

```php
function set_session(c: connection)
{
    if (!c?ssl)
        c$ssl = [$ts=network_time(), $uid=c$uid,
        $id=c$id, $cert_chain=vector()];
}
```
SSL Client Hello

```rust
event ssl_client_hello(c: connection, version: count, possible_ts: time,
session_id: string, ciphers: count_set) &priority=5
{
    set_session(c);

    // Save the session_id if there is one set.
    if (session_id != /^\x00{32}$/ )
        c$ssl$session_id = bytestring_to_hexstr(session_id);
}
```
SSL Server Hello

event ssl_server_hello(c: connection, version: count, possible_ts: time,
session_id: string, cipher: count,
comp_method: count) &priority=5
{
    set_session(c);

c$ssl$version = version_strings[version];
c$ssl$cipher = cipher_desc[cipher];
}
Certificates

```cpp
event x509_certificate(c: connection, cert: X509, is_server: bool, 
                         chain_idx: count, chain_len: count, 
                         der_cert: string) &priority=5
{
    set_session(c);
    if (chain_idx == 0)
    {
        # Save the primary cert.
        c$ssl$cert = der_cert;
        # Also save other certificate information about the primary cert.
        c$ssl$subject = cert$subject;
        c$ssl$not_valid_before = cert$not_valid_before;
        c$ssl$not_valid_after = cert$not_valid_after;
    }
    else
    {
        # Otherwise, add it to the cert validation chain.
        c$ssl$cert_chain[|c$ssl$cert_chain|] = der_cert;
    }
}
```
server_name extension

```c
event ssl_extension(c: connection, code: count,
    val: string) &priority=5
{
    set_session(c);

    if ( extensions[code] == "server_name" )
        c$ssl$server_name = sub_bytes(val, 6, lvall);
}
```
Finish the log

event ssl_established(c: connection) &priority=5
   {
      set_session(c);
   }

event ssl_established(c: connection) &priority=-5
   {
      Log::write(SSL::LOG, c$ssl);
      delete c$ssl;
   }