Zeek Plugins

Zeek provides a plugin API that allows you to extend Zeek's functionality without changing the core packages.

You can add the following functionality:

- Zeek scripts
- Builtins, i.e functions/events/types
- Protocol analysers
- File analysers
- Logging / Input framework backends

Writing a plugin

We can show an example of how plugins work by writing our own rot13 implementation.

We use the Zeek helper script auxil/zeek-aux/plugin-support/init-plugin to create a template for our plugin.

This takes the following arguments

```
$ init-plugin ./rot13-plugin Demo Rot13
```

The first argument is the directory where we will create the plugin, the second argument is the namespace, and the third argument is the name of the plugin.

Zeek uses a combination of namespace and name to identify plugins to avoid conflicts. For example you can have two different plugins at Demo/Rot13 and Demo1/Rot13 and there won't be any conflict.

Save our file as a BIF (built in function) file:

```
$ cat src/rot13.bif
```

```
%%{
```

```
#include <cstring>
#include <cctype>
#include "zeek/util.h"
#include "zeek/ZeekString.h"
#include "zeek/Val.h"
%%}

module Demo;

function rot13%(s: string%) : string
    %{
    char* rot13 = util::copy_string(s->CheckString());

    for ( char* p = rot13; *p; p++ )
        {
        char b = islower(*p) ? 'a' : 'A';
        char d = *p - b + 13;

        if ( d >= 13 && d <= 38 )
            *p  = d % 26 + b;
        }

    zeek::String* zs = new zeek::String(1,
reinterpret_cast<byte_vec>(rot13),
                                        strlen(rot13));
    return make_intrusive<StringVal>(zs);
    %}
```

```
——————————————————————————————————————————--------------
----
```

Now we are ready to compile our plugin. The configure script needs to be able to find the location of a Zeek installation or source tree.

When you are using a installation-tree, you need to add the associated zeek-config to your PATH environment variable

```
# which zeek-config
/usr/local/zeek/bin/zeek-config
# export PATH=“/usr/local/zeek/bin/:$PATH”
# cd rot13-plugin
# ./configure && make
[... cmake output ...]
```

When instead building a plugin against a Zeek source-tree (which must be built first), the configure script must be told its location:

```
# cd rot13-plugin
# ./configure --zeek-dist=/path/to/zeek/dist && make
[... cmake output ...]
```

This builds the plugin in the subdirectory build/, which Zeek treats as a dynamic plugin. We can verify that it indeed treats the build/ subdirectory like this with the -N option, which shows new plugins

```
# export ZEEK_PLUGIN_PATH=/path/to/rot13-plugin/build
# zeek -N
[...]
Demo::Rot13 - <Insert description> (dynamic, version
0.1.0)
[...]
```

We can add a description to tell users what the plugin is about by editing the config.description line in src/Plugin.cc

```
[...]
plugin::Configuration Plugin::Configure()
    {
    plugin::Configuration config;
    config.name = "Demo::Rot13";
    config.description = "Caesar cipher rotating a
string's letters by 13 places.";
    config.version.major = 0;
    config.version.minor = 1;
    config.version.patch = 0;
    return config;
    }
[...]
```

Now we can rebuild and verify that the description is visible:

```
# make
[...]
# zeek -N | grep Rot13
Demo::Rot13 - Caesar cipher rotating a string's letters
by 13 places. (dynamic, version 0.1.0)
```

Now we can use our plugin!:

```
# zeek -e ‘print Demo::rot13(“Hello”)’
>>> Uryyb
```

Before distributing the plugin, it is important to edit README, VERSION and CHANGES. You should also add a licence at COPYING.

Testing Plugins

The init-plugin script buts in place a basic test-suite to start with. Firstly it comes with a single test that checks that Zeek has loaded the plugin correctly:

```
# cd tests
# btest -A
[  0%] rot13.show-plugin ... ok
all 1 tests successful
```

Now we can add a custom script that ensures our rot13 is actually rotating by 13 as designed:

```
# cd tests
# cat >rot13/bif-rot13.zeek

# @TEST-EXEC: zeek %INPUT >output
# @TEST-EXEC: btest-diff output

event zeek_init()
    {
    print Demo::rot13("Hello");
    }
```

Now install the baseline and run the test suite:

```
# btest -U rot13/bif-rot13.zeek
all 1 tests successful
# btest
all 2 tests successful
```