|galois|

v3spa-lobster Phase 2 Report

James Bielman <jamesjb@galois.com> Jesse Hallett <jesse@galois.com>

Modules in Lobster

Lobster supports modules of 2016-02-09 (commit eee438).

Motivation

Our previous approach to namespaces was to group related functionality in domains, which resulted in domains nested in domains. When interacting with a nested domain, Lobster requires that connections go through ports that are explicitly defined in the parent domain. Instead of declaring a connection to a port in the target domain, one would have to declare a connection to a port in the target's parent domain, and declare another connection from the parent domain to the target. This resulted in a lot of ports, and introduced ambiguity over how connections between a domain and its parent line up with connections coming into and out of the parent.

In some cases it is useful the encapsulation that nested domains provide is useful - if inner domains are truly intended to be an implementation detail that are not visible from the outside. But for purposes of avoiding name collisions and grouping related functionality, domain nesting is unnecessarily complicated.

The new module system scopes domains so that multiple domains (or classes) with the same names can exist in separate modules. Connection statements may reference domains in other modules directly.

Usage

Module support is intended to be lightweight, and is based on the module system in Rust. Lobster supports a minimal subset of Rust's module features.

Modules are introduced with the `mod` keyword, a module name, and a block for module content. For example, to declare a module called `net`:

```
mod net {

  // statements here define module-scoped domains and classes

}
```

Modules are not coupled to files. A file may contain any number of modules, with any names - or a file may not contain any modules, in which case definitions go into global scope.

Modules may be nested.

© 2015 Galois, Inc., Proprietary

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, Oregon 97204
T 503.626.6616 | F 503.350.0833
www.galois.com

```
mod net {
  mod web {

    class WebServer () {
      port php_script;
    }

    domain web_server = WebServer();

  }
}
```

References to resources in another module are made by prefixing variable or type names with `mod_name::`, which multiple prefixes for references to nested modules. A reference to the `web_server` domain in the example above from any other module looks like this:

```
net::web::web_server
```

Cross-module references must always include an absolute module path from the root namespace. For example, there is no case where the reference above could be shortened by excluding the `net::` prefix.

Integration with v3spa

We have updated the API of the v3spa service to allow projects to be broken up into multiple files. The service accepts uploads of individually-changed files, and handles combining the changes with the rest of the project. We hope that support for modules will make spreading code over multiple files more manageable. These changes are documented in the **V3SPA 2.0 API Reference** document.

Example

This example defines three modules in a single source file. The last module, `app`, includes connection statements with qualified references to ports in domains in the `system` and `net` modules.

```
mod system {
  class Init () {
    port creator;
  }

  domain init = Init();

  class Syslog () {
    port log;
  }
```

© 2015 Galois, Inc., Proprietary

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, Oregon 97204
T 503.626.6616 | F 503.350.0833
www.galois.com

```
  domain syslog = Syslog();
}

mod net {
  class WebServer () {
    port php_script;
  }

  domain web_server = WebServer();

  class MailServer () {
    port send;
  }

  domain mail_server = MailServer();
}

mod app {
  class Application () {
    port inp : {position = subject};
    port create : {position = object};
    port outp : {position = subject};
    port log : {position = subject};
  }

  domain application = Application();

  net::web_server.php_script --> application.inp;
  application.outp --> net::mail_server.send;

  system::init.creator --> application.create;

  application.log --> system::syslog.log;
}
```

Enhanced SELinux Import/Export

Lobster 2.x contains a mechanism for supporting constructs in the SELinux language that are not part of the type enforcement graph. These statements are represented as domains in the special `selinux__` module, which is ignored by normal analysis. Arguments to these statements are stored in annotations on these domains.

For example, given the following declarations:

```
role system_r httpd_passwd_t;
```

© 2015 Galois, Inc., Proprietary

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, Oregon 97204
T 503.626.6616 | F 503.350.0833
www.galois.com

```
typealias httpd_unconfined_rw_content_t { httpd_unconfined_script_rw_t httpd_unconfined_content_rw_t };
role_transition unconfined_r direct_init_entry system_r;
```

Lobster will generate the following module and domains:

```
mod selinux__ {
  [SysDomain(role), Name(system_r), Types(httpd_passwd_t)]
  domain dom40 = {};

  [SysDomain(type_alias), Name(httpd_unconfined_rw_content_t),
   Aliases(httpd_unconfined_script_rw_t, httpd_unconfined_content_rw_t)]
  domain dom51 = {};

  [SysDomain(role_transition), CurrentRoles(unconfined_r), Types(direct_init_entry), NewRole(system_r)]
  domain dom77 = {};
}
```

These special annotations used by the Lobster export code to reconstruct these statements in their original form.

© 2015 Galois, Inc., Proprietary

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, Oregon 97204
T 503.626.6616 | F 503.350.0833
www.galois.com