Domain-specific Static Analysis with "Lighthouse"

Joseph Birr-Pixton

28th October 2009

Contents

Contents

Contents

Introduction

Definitions (my own):

General purpose SA SA with respect to standard language features, standard libraries, or common OS interfaces. *Examples: Coverity, Klocwork, Fortify, most compilers.*

Domain-specific SA User-controlled SA of custom APIs. *Example: sparse.*

But:

Introduction

Definitions (my own):

General purpose SA SA with respect to standard language features, standard libraries, or common OS interfaces. *Examples: Coverity, Klocwork, Fortify, most compilers.*

Domain-specific SA User-controlled SA of custom APIs. *Example: sparse.*

But:

- Most general purpose SA systems support customisation.

Introduction

Definitions (my own):

General purpose SA SA with respect to standard language features, standard libraries, or common OS interfaces. *Examples: Coverity, Klocwork, Fortify, most compilers.*

Domain-specific SA User-controlled SA of custom APIs. *Example: sparse.*

But:

- Most general purpose SA systems support customisation.
- A domain-specific SA system is a general purpose SA system with some difficult bits missing.

Orientation

"Let's construct a useful domain-specific SA thing from bits we have lying around..."

Design

- We have a good, very well tested C/C++ parser.

Design

- We have a good, very well tested C/C++ parser. It's called GCC.

Design

- We have a good, very well tested C/C++ parser. It's called GCC.
- We have a wide selection of languages available for writing SA rules.

Design

- We have a good, very well tested C/C++ parser. It's called GCC.
- We have a wide selection of languages available for writing SA rules. 'Programming' languages, if you will.

Design

- We have a good, very well tested C/C++ parser. It's called GCC.
- We have a wide selection of languages available for writing SA rules. 'Programming' languages, if you will.
- GCC is *already* suitably integrated with whatever build system is used.

Design

- We have a good, very well tested C/C++ parser. It's called GCC.
- We have a wide selection of languages available for writing SA rules. 'Programming' languages, if you will.
- GCC is *already* suitably integrated with whatever build system is used.
- Compile and SA in one step will almost always be quicker than separate compile and SA processes: let's abolish the difference.

Design

Aims:

- Let's write as little C as possible.

Design

Aims:

- Let's write as little C as possible.
- Let's minimise the amount of stuff needing to be kept in sync with GCC's fluid internal APIs.

Design

Aims:

- Let's write as little C as possible.
- Let's minimise the amount of stuff needing to be kept in sync with GCC's fluid internal APIs.
- Let's not require any specific language for writing rules.

Design

- A shared object (written in a minimal amount of C) gets dynamically loaded into GCC, installs itself as a compilation pass and squirts detailed IR[1] at a subprocess ('lh-pipe').

[1] intermediate representation

Design

- A shared object (written in a minimal amount of C) gets dynamically loaded into GCC, installs itself as a compilation pass and squirts detailed IR[1] at a subprocess ('lh-pipe').
- Subprocess inherits GCC's `stdout`/`stderr` and produces arbitrary output.

[1] intermediate representation

Design

- A shared object (written in a minimal amount of C) gets dynamically loaded into GCC, installs itself as a compilation pass and squirts detailed IR[1] at a subprocess ('lh-pipe').
- Subprocess inherits GCC's `stdout`/`stderr` and produces arbitrary output.
- Non-zero subprocess exit causes GCC to fail in turn and end the build.

[1] intermediate representation

IR

- Based on GIMPLE (GCC's internal IR).

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)
  4. Switch (switch (expr) case val: goto block1; ...)

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)
  4. Switch (switch (expr) case val: goto block1; ...)
- Encodes control flow graph.

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)
  4. Switch (switch (expr) case val: goto block1; ...)
- Encodes control flow graph.
- XML-based.

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)
  4. Switch (switch (expr) case val: goto block1; ...)
- Encodes control flow graph.
- XML-based.
- Contains full source file for reporting purposes.

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)
  4. Switch (switch (expr) case val: goto block1; ...)
- Encodes control flow graph.
- XML-based.
- Contains full source file for reporting purposes.
- Details of all referenced external types and functions.

IR

- Based on GIMPLE (GCC's internal IR).
- Only 4(!) essential statements:
  1. Assignment (lhs = rhs, lhs = op rhs, lhs = rhs1 op rhs2)
  2. Function call (lhs = func(args, ...))
  3. Condition (if (expr) goto block1 else goto block2)
  4. Switch (switch (expr) case val: goto block1; ...)
- Encodes control flow graph.
- XML-based.
- Contains full source file for reporting purposes.
- Details of all referenced external types and functions.
- Actual alignment and size information for aggregate types.

Example IR

```
#include <stdio.h>

int main(void)
{
  printf("hello world!\n");
  return 0;
}
```

Compiled with:

```
$ gcc -fplugin=lighthouse-client.so -o helloworld helloworld.c
```

Produces (other than the desired executable)...

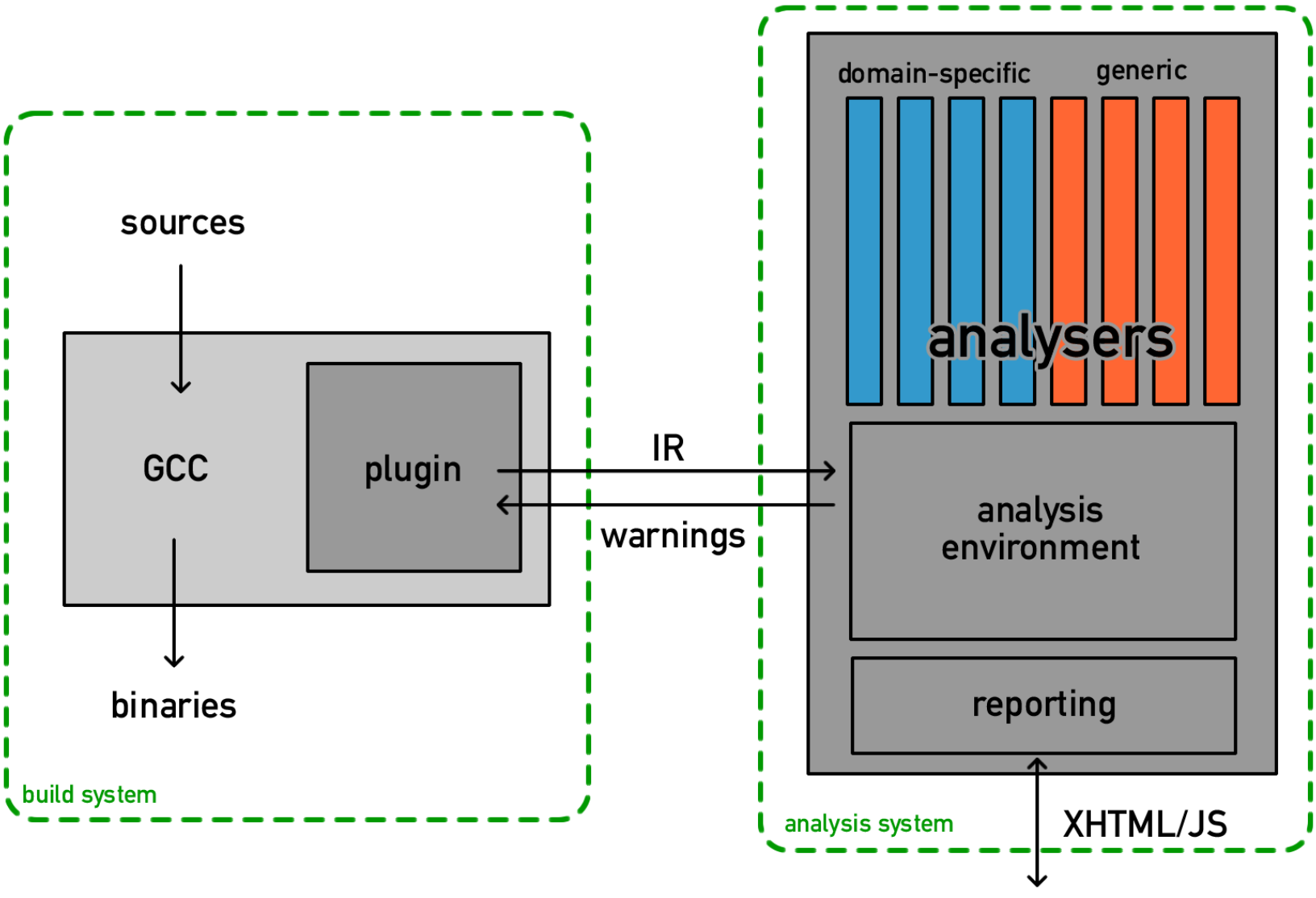
```
<lh-translation-unit client-version="0.1" filename="helloworld.c" language="C">
  <raw-source>#include &lt;stdio.h&gt;

int main(void)
{
  printf("hello world!\n");
  return 0;
}
</raw-source>
  <referenced-types>
  </referenced-types>
  <function-bodies>
    <function body-begin="4" body-end="7" location="helloworld.c:3" name="main">
      <returns>
        <integer name="int" precision="32" />
      </returns>
      <args>
      </args>
      <body entrypoint="2">
        <locals>
          <local location="helloworld.c:6:3">
            <binding id="2472" />
            <type alignment="32" size="32">
              <integer name="int" precision="32" />
            </type>
          </local>
        </locals>
```

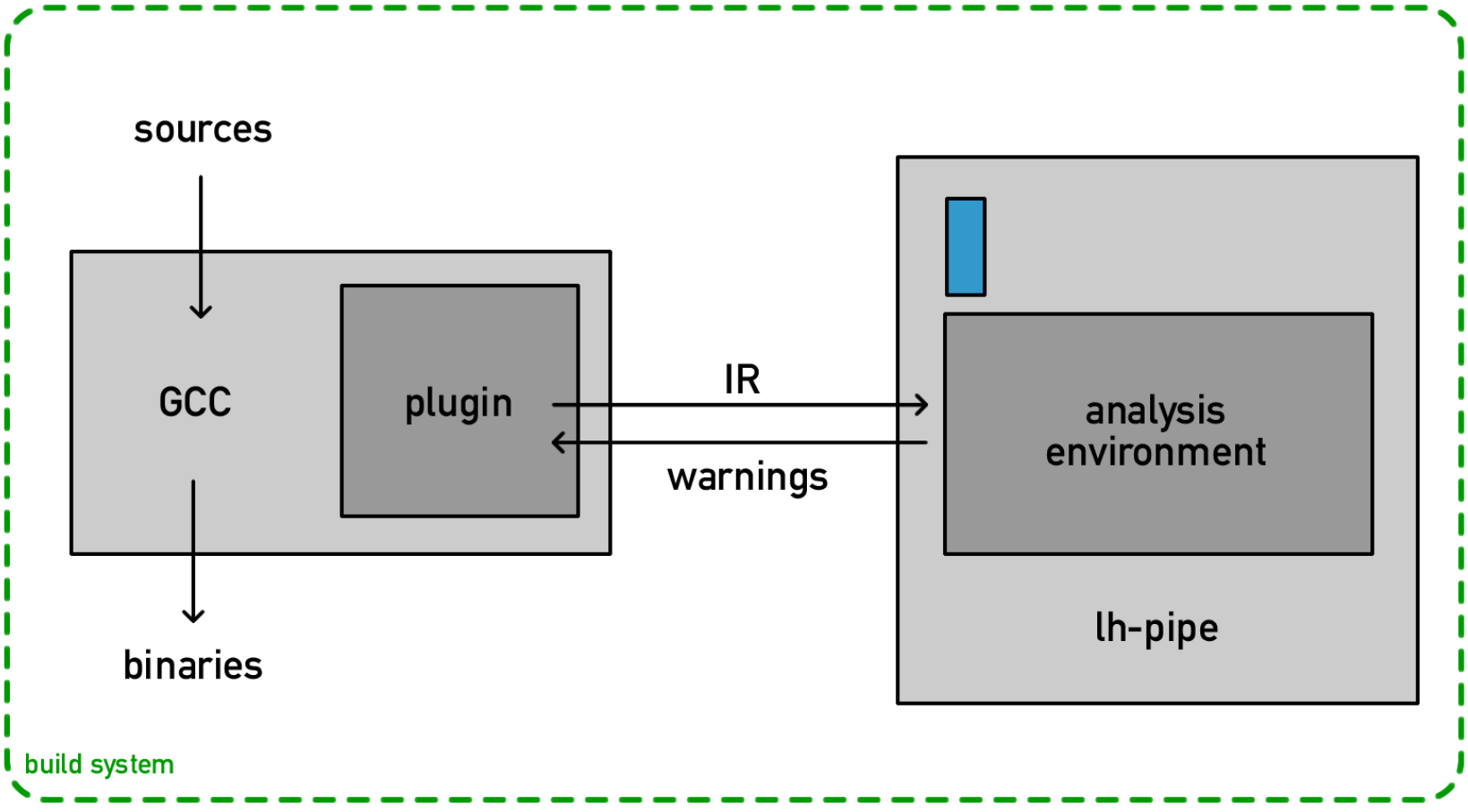
```
<block id="2">
  <call location="helloworld.c:5:9">
    <function id="739" name="__builtin_puts" />
    <lhs>
      <void />
    </lhs>
    <args>
      <addr-of>
        <item-ref>
          <array>
            <constant>
              <array id="1521">
                <type>
                  <integer constant="1" name="char" precision="8" />
                </type>
                <domain>
                  <integer max="12" min="0" precision="32" />
                </domain>
              </array>
              <string-literal>hello world!\x00</string-literal>
            </constant>
          </array>
          <index>
            <constant>
              <integer name="int" precision="32" />
              <integer-literal value="0" />
            </constant>
          </index>
        </item-ref>
      </addr-of>
    </args>
  </call>
```

```
        <assign location="helloworld.c:6:3">
          <lhs>
            <bound id="2472" />
          </lhs>
          <rhs>
            <constant>
              <integer name="int" precision="32" />
              <integer-literal value="0" />
            </constant>
          </rhs>
        </assign>
        <return>
          <bound id="2472" />
        </return>
      </block>
    </body>
    <externals>
      <external location="&lt;built-in&gt;:0:0">
        <binding id="739" name="__builtin_puts" />
        <type alignment="8">
          <function attributes="nonnull">
            <return>
              <integer name="int" precision="32" />
            </return>
            <arguments>
              <addr-of>
                <integer constant="1" name="char" precision="8" />
              </addr-of>
              <void />
            </arguments>
          </function>
        </type>
      </external>
    </externals>
  </function>
 </function-bodies>
</lh-translation-unit>
```

Desired end architecture



Current status



Results: type checking for `dsUnpackMap` et al

- Problem: in C, arguments to `varargs`[2] functions cannot be type-checked by the compiler.

[2] Think `printf`, `sscanf`

Results: type checking for `dsUnpackMap` et al

- Problem: in C, arguments to `varargs`[2] functions cannot be type-checked by the compiler.
- In the oscar codebase, we have a family of such functions to compactly construct or deconstruct d3s messages.

[2] Think `printf`, `sscanf`

Results: type checking for `dsUnpackMap` et al

- Problem: in C, arguments to `varargs`[2] functions cannot be type-checked by the compiler.
- In the oscar codebase, we have a family of such functions to compactly construct or deconstruct d3s messages.

Checker written in 180-odd lines of python.

[2] Think `printf`, `sscanf`

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Error: Value argument (index 0) to dsPackListB is not a "int"
                   as specified, but a "const unsigned int".
   call-site at: ../as_ecc.c:373:7
*** 368      DSList *poly = dsNewList(ctx->cc.a, NULL, 0);
*** 369
*** 370      /* Run through the coefficients. */
*** 371      for (i = 0; i < D->E.field.field.poly.num_terms; i++)
*** 372      {
>>> 373        NOERR( dsPackListB(&ctx->cc, poly, "i", D->E.field.field.poly.uterms[i]) );
               ^----- call-site
*** 374      }
*** 375
*** 376      field = dsMakeListB(ctx->cc.a, "hm", sym_AS_Binary, dsFreezeList(&poly));
*** 377    }
*** 378
```

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Error: Value argument (index 2) to dsUnpackList is not a "int*"
                   as specified, but a "unsigned int*".
   call-site at: ../as_mech_hmac.c:227:21
*** 222    DSMessage *msg_hash;
*** 223    const ASMethods_CHASH *hmethods;
*** 224    unsigned int outlen;
*** 225
*** 226    /* Our constructor is [HMAC, chash, outlen] */
>>> 227    if ( !dsUnpackList(&ctx->cc, DS_UNPACK_CHKSIZE, mech, "-mi", &msg_hash, &outlen) ||
                          ^----- call-site
*** 228         (priv->hash = asBuildMech(msg_hash, asCHASH, ctx))==NULL )
*** 229    {
*** 230      asError(sym_AS_BadMechParams, mech, ctx);
*** 231      goto x_fail;
*** 232    }
```

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Error: Symbol argument (index 0) to dsUnpackMap is not a DSSymbol,
                   but a "struct DSMessage**".
   call-site at: dddstest.c:2661:19
*** 2656   WANT_RC(rc, ERR_THAWED, "Check for thawed map passed on dsGetAggregateValueType");
*** 2657
*** 2658   rc = dsGetMapKeyType(msgs[i_Map][thw], &type);
*** 2659   WANT_RC(rc, ERR_THAWED, "Check for thawed map passed on dsGetMapKeyType");
*** 2660
>>> 2661   rc = dsUnpackMap(&ctx, DS_UNPACK_ANY, msgs[i_Map][thw], "m", &hm);
                         ^----- call-site
*** 2662   WANT_RC(rc, DS_FALSE, "Check for thawed map passed on dsUnpackMap");
*** 2663   CHK_ERR( rc, sym_Err_DSThawedError );
*** 2664
*** 2665   CHK_RC( dsMapAdd(Map[mut], dsIncRef(str), dsIncRef(rndmsg[0])) );
*** 2666   FREEZE_DO_AND_THAW( Map, rc = OK );
```

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Error: Value argument (index 3) to dsUnpackMap is not a "uint8_t*"
                   as specified, but a "int8_t*".
   call-site at: dddstest.c:2502:3
*** 2497   m = dsFreezeMap(&map);
*** 2498   map = NULL;
*** 2499   FILLZERO(ctx);
*** 2500   ctx.a = a;
*** 2501   FILLZERO(uvout);
>>> 2502   CHK_OK(dsUnpackMap(&ctx, 0, m,
           ^----- call-site
*** 2503                      "v(s)v(l)v(max)v(8)v(16)v(32)v(64)",
*** 2504                      sym_Tst_First, &uvout.s,
*** 2505                      sym_Tst_Second, &uvout.l,
*** 2506                      sym_Tst_Third, &uvout.max,
*** 2507                      sym_Tst_Fourth, &uvout.u8,
```

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Error: Unknown format string 'U'
   call-site at: dddstest.c:2235:3
*** 2230   CHK_OK( dsPackListB(&ctx, dslist2, "v", src2.u) );
*** 2231
*** 2232   check_equal_list(&dslist, &dslist2, "Lists created by dsPackList() differ" );
*** 2233   /* This shares code with dsPackMapMsgB(), I'm not bothering with the full invalid-format check
*** 2234
>>> 2235   CHK_ERR( dsPackListB(&ctx, dslist2, "U"), sym_Err_FormatStringError );
           ^----- call-site
*** 2236
*** 2237   /* Now unpack some lists */
*** 2238
*** 2239   mlist = dsFreezeList(&dslist);
*** 2240
```

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Warning: Call to dsUnpackMap has variable format string.
                     Verify it will always correspond with the passed in types.
   call-site at: dddstest.c:2003:9
*** 1998       case 'e': case 'E':
*** 1999       case 'u': case 'U':
*** 2000       case 'd': case 'D': break;
*** 2001
*** 2002       case 0:
>>> 2003         CHK_OK( dsUnpackMap(&ctx, 0, m, fmt) );
                 ^----- call-site
*** 2004         break;
*** 2005
*** 2006       default:
*** 2007         CHK_ERR( dsUnpackMap(&ctx, 0, m, fmt), sym_Err_FormatStringError );
*** 2008         break;
```

Results: type checking for `dsUnpackMap` et al

```
lighthouse: Error: Value argument (index 0) to dsUnpackMap is not a "uint32_t*"
                   as specified, but a "ntl_connection_t*".
   call-site at: ../ntl_remote.c:1143:20
*** 1138
*** 1139   FILLZERO(ctx);
*** 1140   ctx->a = inst->xa;
*** 1141   NTL_MUTEX_ACQUIRE(ntl__remote_mutex); locked = 1; {
*** 1142     /* Extract the connection handle */
>>> 1143     if(!dsUnpackMap(ctx, 0, reply, "v(32)",
                           ^----- call-site
*** 1144                     sym_NTL_RPC_CONNECTION, conn_out)) {
*** 1145       D(("dsUnpackMap failed"));
*** 1146       status = NTL_ERROR_RPC_PROTOCOL;
*** 1147       goto error;
*** 1148     }
```

Results: build performance

A full `buildme` in `nglibs` took:

Results: build performance

A full `buildme` in `nglibs` took:

119 CPU seconds Without plugin loaded.

Results: build performance

A full `buildme` in `nglibs` took:

119 CPU seconds Without plugin loaded.

134 CPU seconds With plugin loaded.

Things for the future

- Write checker for `PyArg_ParseTuple()` et al and compile lots of python extensions.

Things for the future

- Write checker for `PyArg_ParseTuple()` et al and compile lots of python extensions.
- Write proper data flow analysis tools for use by checkers, so they can do analysis of resource lifetimes, etc.

Things for the future

- Write checker for `PyArg_ParseTuple()` et al and compile lots of python extensions.
- Write proper data flow analysis tools for use by checkers, so they can do analysis of resource lifetimes, etc.
- Build out separate analysis system so we can do proper inter-procedural analysis.

Fin

Questions?