HAL as a Hacking Tool

Merlin of DC949

A practical guide with source code is prefaced by a short history of Artificial Life development and comparison to security development.

Overview

- AI hackers have learned to leverage bottom-up design for solving complicated problems
- Many of the earliest hackers were found in or around AI labs
- As hackers we have a multitude of hard to define and therefore difficult to solve issues facing us; therefore, we need to attack these problems in the most effective and reproducable manner possible

Takeaway

- It is easy, but requires some forethought, to effectively create interoperable applications and systems that have higher value together than alone
- If each piece of the application or system does a few specific things really well (even with many options) it is easier to make that piece really good at what it does
- Lots of really effective pieces lead to ex-tremely effective systems

Historical Notes

- In the beginning, the AI community promised it could deliver amazingly complex intelligent systems quickly and easily
- This effort failed horrendously – top down cathedrals of code burned by complexity and ill defined problems. AV has recently re-peated this learning curve.
- Lisp (a relatively simple construct) evolved to help solve difficult problems in an ex-ploratory manner and was included in emacs

What's Working

- Many tools in unix/linux CLI userland are an especially good example of these concepts
- `cat` is great, so are `more` and `grep` but all three together in a shell with piping can do the same jobs many commercial GUI applications reimplement badly
- Firefox, MetaSploit, emacs, and such are programs that are extensible to great effect and in the case of emacs and firefox through many domains!

Why it Wins

- Top down design can never account for every little thing and can lead to rigid designs
- Bottom up design can be adaptable to all kinds of unanticipated requirements
- Minimum cost is a few minutes or hours before you start building your next project spent thinking from the bottom-up
- Current projects more easily leverage past work

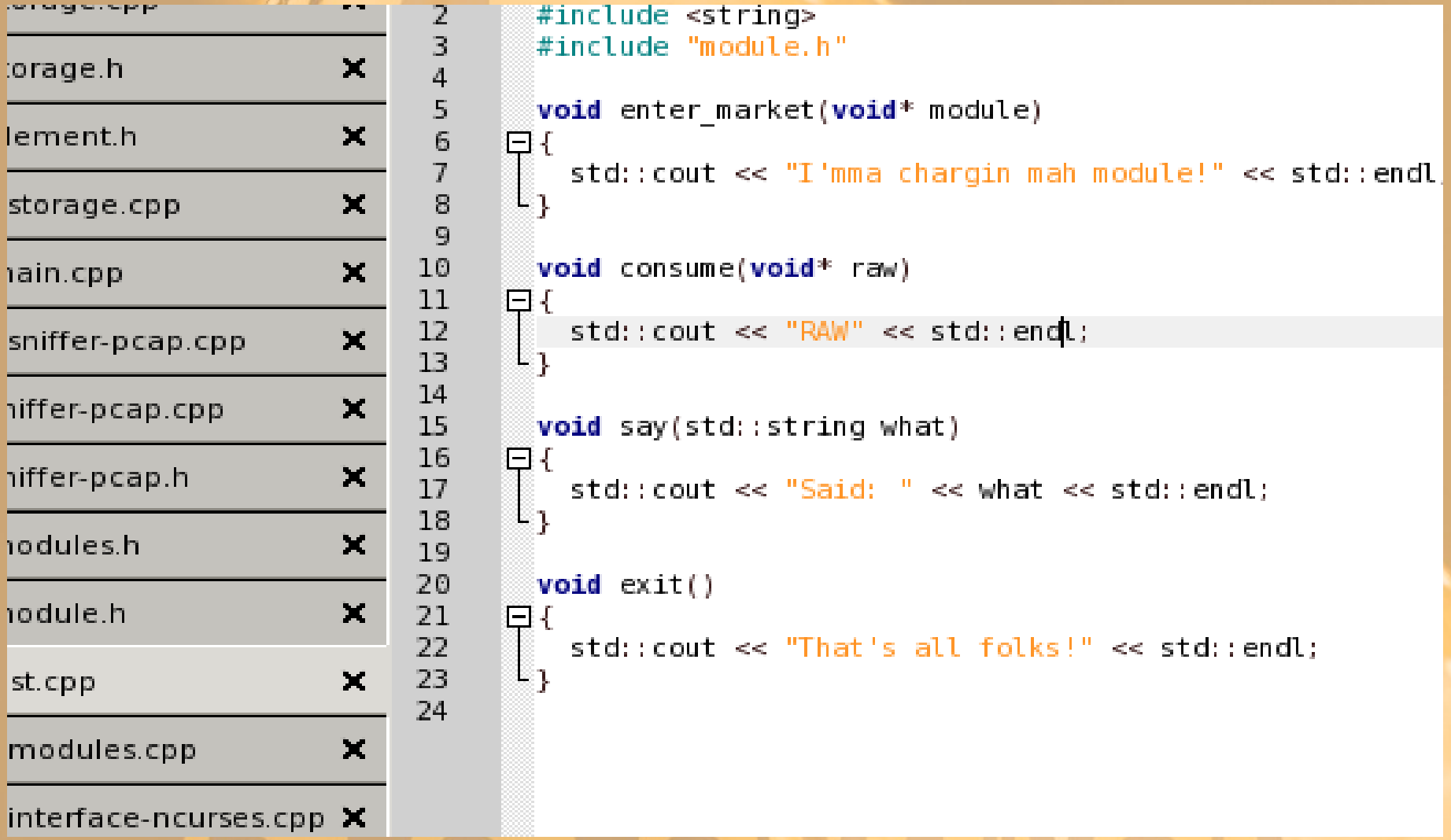
You might try...

- Begin or continue more consciously engaging in bottom-up design of systems
- Keep connections simple – it is easy to add unneeded complexity
- Make your next tool interoperable with others via API, shell, or other methods
- Everyone can do these things and everyone benefits from it

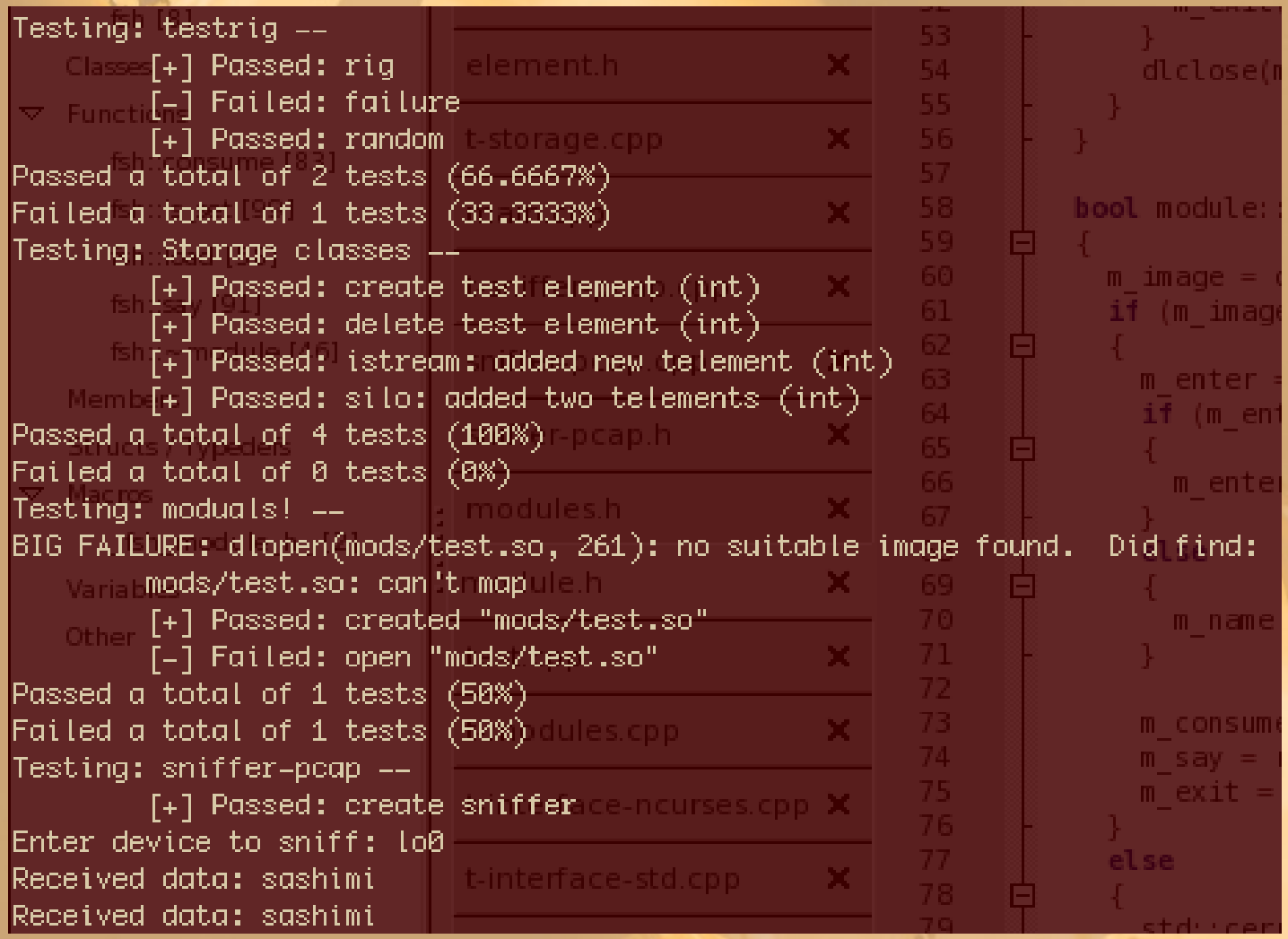
Interface

```
namespace fsh
{
  class module
  {
    public:
      module();
      module(std::string path);
      ~module();

      bool load(std::string path);
      bool consume(fsh::element* roll);
      bool say(std::string what);

      bool is_set() const;

    private:
      typedef void (* mod_void)();
      typedef void (* mod_param)(void* module);
      typedef void (* mod_string)(std::string what);

      void* m_image;
      mod_param m_enter, m_consume;
      mod_string m_say;
      mod_void m_exit;
      std::string m_path, m_name;
  };
```

Implementation



The Build



- It didn't really work