World of Warcraft - 10,000,000 Lua users and growing!

James Whitehead II
jnwhiteh@gmail.com

July 15, 2008

Outline

Who am I?

- Graduate of Syracuse University (BSc - 2002, MSc - 2004)
- DPhil student at University of Oxford
  - Studying the visualization of programs for computer science education
- Addon developer for World of Warcraft
- Co-Author of two books about World of Warcraft:
  - Hacking World of Warcraft
  - World of Warcraft Programming: A Guide and Reference for Creating WoW Addons
- Developer of `ircbot` Lua module
- Contributer to Sputnik project
- Project lead for TNT: An AOL Instant Messenger client in Emacs ;-)

What is World of Warcraft

- Massively Multiplayer Online Role-Playing Game
- Currently holds 62% of the MMORPG market
- 10,000,000 subscribers worldwide
- Social gaming
  - Single player mode
  - Small groups (1-5 people)
  - Small raid groups (10 people)
  - Large raid groups (25 people)

What does this have to do with Lua?

- User Interface defined by Lua scripts and XML definitions
  - Frames and frame handlers can be defined in XML files
  - Behavior and code defined in a series of Lua scripts
- 140 XML files - 68,240 LoC
- 142 Lua files - 66,357 LoC
- Not only is code freely available, but serves as model implementation for all aspects of the user interface and API.
- Function calls in Lua can be self-documenting:

```
texture, itemCount = GetContainerItemInfo(containerId, slotId)
```

WoW Addon System: How does it work?

- When the game client is opened, the `Interface/AddOns` directory is scanned
- Addons must reside within their own subdirectory, with a table of contents file
  - Lua scripts
  - XML files
  - Fonts
  - Graphics
  - Other media...
- Addons are loaded sequentially
  - Load order determined by default sort order
  - Dependencies resolved
  - Addons can be enabled or disabled individually by user

Demonstration of Addons

Limitations: Input

- Only those files that existed when client loaded can be loaded by the game, although the contents of those files may change. In order to add files, you need to exit the client.
- Persistent data (referred to as "saved variables") are loaded when the player logs in or reloads their user interface. Reloading the interface takes between 5 and 10 seconds, so it's not a viable means of communication.
- Most important functionality requires hardware input (keyboard, mouse)
  - Although a third-party program could be written to inject keystrokes, Blizzard has a two-stage system called "Warden" designed to prevent that type of exploit.

Limitations: Output

- No file input/output API
- No operating system level API
- All saved settings are buffered or only occur when the user logs off, or reloads their user interface.

Limitations: Gameplay

- Character movement API functions are “protected” and require hardware event
- Spellcasting
- Targeting

Changes to Lua: Security Model

World of Warcraft includes a security model designed to *protect* certain functions so they can only be called from a secured codepath. Initially, all Blizzard code is untainted, and all user supplied (addon) code is tainted. Whenever a *protected* function is called from a tainted codepath, an error is thrown and the call fails.

Security Model: Tainted values

- Added a taint flag to all values
- All Blizzard code is untainted by default
- Any user supplied code (addons) is tainted during the loading process
- C API exists to “scrub” a value, removing taint so it can be used in untainted code.

Security Model: Execution Path

- Execution path has a taint flag that is set by reading a tainted variable
- A write to a variable from a tainted execution path stores that taint in the variable
- This model allows for protection of API functions with minimal problems for the end-user due to addon author mistakes
- Initial introduction caused some issues due to legacy Blizzard code that would read from potentially tainted values. These problems have been alleviated as they have been recognized

Security Model: "Free" Memory Profiling

- The taint flag is actually stored as an unsigned short
- Taint flag actually indicates which specific addon has tainted a given value
- Taint thus acts as a form of ownership
- Provides broad information about addon memory usage

Lua 4.X string.format selectors

- Older version of Lua included selectors for `string.format`:
  - `> print(string.format("%2$d, %1$d, %d", 13, 17))`
    `17, 13, 17`
- Used in localization:
  - Spell: Shadow Word: Pain
  - Actor: Cladhaire
  - Action: Spell being removed
  - In English:

    `"%s's %s is removed."`
  - In German:

    `"'%2$s' von %1$s wurde entfernt."`
  - Results:
    - Cladhaire's Shadow Word: Pain is removed.
    - 'Shadow Word: Pain' von Cladhaire wurde entfernt.

String functions: split, join, trim

- `str = string.join(sep, ...)`
- `...  = string.split(sepChars, str)`
- `str = string.trim(str)`

Lua 5.1 in World of Warcraft

When the first expansion for World of Warcraft was released in January, 2007 Blizzard took the opportunity to upgrade from Lua 5.0 to Lua 5.1. This gained us a number of features that helped immensely with addon development:

- Vararg system
- Long string change allowing for nesting
- Incremental garbage collection

Unfortunately due to the sandboxed nature of addon development in WoW, we don't make use of the package/require system, so those changes didn't benefit us directly

Lua 5.1: Incremental GC

- In Lua 5.0 users were experiencing frequent freezes in their user interface during heavy combat situations due to the atomic garbage collection
- Users wrote a number of addon solutions that performed garbage collection as pre-defined "safe" times, such as when entering a city or exiting combat
- When incremental GC was introduced the system seemed to be much smoother and caused less issues for users
- After further testing and observation the garbage collector wasn't aggressive enough, since it only took a step when new functions or tables were created
- I created a simple addon solution that called a step of the garbage collector periodically when the player was not in combat
- GCTweak was integrated into the WoW game client, and the GC system works quite well

Community (Lua) Code

The World of Warcraft community is somewhat unique in that we cannot use modules or external packages and all code must be written in Lua. As a result we have written quite a lot of painful Lua code to do things that are much easier to do in other languages.

- Compression/Decompression
- Encryption/Decryption
- Data Structures
- Algorithms

Community Code: Examples

- Prism - Lua implementation of SHA-256 and RSA algorithms. Keypairs must be generated externally (i.e. openSSL) due to the cost of probabalistic calculation of primes in Lua. This library includes a BigNum implementation. This code has been used to distribute signed code to remote users, but hasn't seen much use in practice due to the volatile nature of remote execution.
  http://www.curse.com/downloads/details/8018/
- Traveling Salesman - Lua implementation of the Ant Colony Optimization of the Traveling Salesman Problem. This is included in an addon called Routes which is used to calculate the most efficient route for collecting materials in WoW.
  http://svn.wowace.com/wowace/trunk/Routes/TSP.lua

Community Code: Examples continued...

- Compress - LZW and Huffman codec implementation. There are a few different compression implementation used when an addon needs to access small amounts of data on demand, for better storage.
  http://svn.wowace.com/wowace/trunk/LibCompress/
- LibXML - Converts a Lua table to an XML data entity.
  http://svn.wowace.com/wowace/trunk/LibXML/
- Roman - Converts numbers into strings of roman numerals.
  http://svn.wowace.com/wowace/trunk/LibRoman-1.0/
- LibStub - Simple global stub that handles versioning of registered components. Allows two versions of the same library to exist alongside each other, and facilitates the upgrading of minor versions of the same library. Allowed us to create our own DLL-hell within World of Warcraft.
  http://svn.wowace.com/wowace/trunk/LibStub/

Community Code: Examples continued...

- TEA - Implementation of the "Tiny Encryption Algorithm" [1] in Lua.
  http://svn.wowace.com/wowace/trunk/TEALib/
- RC4 - Implementation of the RC4 algorithm [2] in Lua.
  http://svn.wowace.com/wowace/trunk/RC4Lib/
- MD5 - Implementation of the MD5 algorithm [3] in Lua.
  http://svn.wowace.com/wowace/trunk/MD5Lib/
- Panda - Rudimentary infobot [4] implementation in Lua, designed to work over the World of Warcraft communication channels. Could be altered to work in another medium.
  http://www.curse.com/downloads/details/2056/

---

[1] http://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm
[2] http://en.wikipedia.org/wiki/RC4
[3] http://en.wikipedia.org/wiki/MD5
[4] http://en.wikipedia.org/wiki/Infobot

Questions?