“Grasshopper always wrong in argument with Chicken.”
- Book of Chan
Functional Fuzzing with Funk

and further explorations into the use of functional languages for network scripting

Benjamin Kurtz
Funk is a framework for the scripted generation of network traffic, written using the Chicken Scheme-to-C compiler.
Funk is...

- Simple
- Tiny
- Powerful
- Extensible
- Platform Independent and Protocol Agnostic
- Easily described by random adjectives
Most Important Idea

Funk creates a generic interface to every network protocol!

This lets you keep your fuzzing logic separate from your protocol logic!
Ok, but can it do?

• Fuzzing
• Flooding
• Spoofing
• Traffic Generation
Long Term Goals

• Query-Response
• Arbitrary Network Scripting
• Rapid Prototyping
• Virtual Servers
• Firewall and IDS
Previous Design

• XML-based scripts in flat file DB
• C/++ parser generator engine
• Domain-Specific Language, limited by regular grammars
• Imperfect, but still made some money
Protocol logic and fuzzing logic were necessarily intertwined...

• Checksums

• Internet Header Length

• Type-Length Value Fields

• ICMP, DHCP, ASN.1
Cue the music...
Scheme FAQ

• What the hell is Scheme anyway?
• Seriously, what’s up with all the parentheses?
• Why are LISP programmers so smug?
• Why can’t you just use C like normal people?
Leave
In
Stupid
Parentheses
Why Scheme?

• Programming metaphor better suited to problem (lambda calc vs. Turing machine)
• Easily extensible
• Well established, widely used
• Portable
• No Bit Rot!
Why Chicken?

• Actively developed
• Highly optimized (fast even in interpreter)
• Extends with Eggs or SWIG
• Compiles to straight C
• Functional language makes dealing with network protocols easy
# Chicken vs. Python

<table>
<thead>
<tr>
<th>Feature</th>
<th>Chicken</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Compiles?</td>
<td>to C</td>
<td>to Java</td>
</tr>
<tr>
<td>Lambdas?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Painfully Slow?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Stupid?</td>
<td>Parentheses</td>
<td>Whitespace</td>
</tr>
<tr>
<td>Tastes Like?</td>
<td>Chicken</td>
<td>Chicken</td>
</tr>
</tbody>
</table>
Implementation
Packet Scripting

- Abstract Operations
- Flexibility
- Extensibility
Protocols

• Protocol Operations:
  • Generate
  • Serialize
  • Validate
  • Query
(define (install-ethernet-protocol)
  ;; Fields (list of lists with values: name, bitlength, validator, serializer)
  (define fields (list
    (list 'destmac  48 mac-validator mac-serializer)
    (list 'srcmac   48 mac-validator mac-serializer)
    (list 'pkt-type 16 (hex-validator 16) (hex-serializer 16))
  )

  (define (generate packet aggregator) (default-generator packet fields aggregator))
  (define (validate packet) (default-validator packet fields))

  ;; Public Interface
  (put-op 'generate '(ethernet) generate)
  (put-op 'validate '(ethernet) validate)

  "ethernet done")
(define (install-ip4-protocol)

;; Fields ( list of lists with values: name, bitlength, validator, serializer )
(define fields (list
    (list 'version  4 (hex-validator 4) (hex-serializer 4))
    (list 'internet-header-length 4 (hex-validator 4) (hex-serializer 4))
    (list 'type-of-service 8 (hex-validator 8) (hex-serializer 8))
    (list 'total-length 16 (hex-validator 16) (hex-serializer 16))
    (list 'identification 16 (hex-validator 16) (hex-serializer 16))
    (list 'CE 1 (hex-validator 1) (hex-serializer 1))
    (list 'DF 1 (hex-validator 1) (hex-serializer 1))
    (list 'MF 1 (hex-validator 1) (hex-serializer 1))
    (list 'fragment-offset 13 (hex-validator 13) (hex-serializer 13))
    (list 'time-to-live 8 (hex-validator 8) (hex-serializer 8))
    (list 'protocol 8 (hex-validator 8) (hex-serializer 8))
    (list 'header-checksum 16 (hex-validator 16) (hex-serializer 16))
    (list 'source-ip 32 ip-validator ip-serializer)
    (list 'dest-ip 32 ip-validator ip-serializer)
    (list 'options 0 (hex-validator 32) (hex-serializer 32))
))
Generate/Validate Operations on Packets and Protocols

(define (generate-layer packet) ( (get-op 'generate (car packet)) (cdr packet) u8vector-cat) )
(define (validate-layer packet) ( (get-op 'validate (car packet)) (cdr packet)) )

(define (validate packet) (cond ((null? packet) '())
      (else
       (cons (validate-layer (car packet))
             (validate (cdr packet)))))

(define (generate packet) (cond ((null? packet) '())
      (else
       (u8vector-cat (generate-layer (car packet))
                    (generate (cdr packet)))))

;; Generate/Validate
Generating a Packet

`Ethernet`

`IP`

`TCP`
(define my-ip-packet (attach-tag '(ip4)
  (list
    "4" "5" "10" "0020"
    "0030" "0" "4" "0"
    "0755" "01" "04"
    "A123" "192.168.1.1"
    "192.168.1.2" ""192.168.1.2"
    )))

(define my-eth-packet (attach-tag '(ethernet)
  (list
    "12:34:56:78:90:12"
    "AA:BB:CC:DD:EE:FF"
    "0800"))

(define my-packet (list my-eth-packet my-ip-packet ))

; send packet out
(require 'raw-sockets)
(raw-open "en0")
(define raw-packet (generate my-packet))
(raw-send raw-packet (u8vector-length raw-packet))
(raw-close)
Chicken Eggs

- bit-cat
- crc16
- raw-sockets
Future Work

• Filter/Receive/Inject Support
• Binary and File Format Fuzzing
• Visual Script Design
• Support for Additional Protocols
Funk Source Code

Current Funk Source is available at:

http://www.memescape.com/funk/funk_current.tgz
Recommended Reading

• Structure and Interpretation of Computer Programs ("The Wizard Book")
  - Abelson & Sussman
  http://mitpress.mit.edu/sicp/

• The Scheme Programming Language
  - R. Kent Dybvig
Stump the chump!
Extras

The following slides have all the information you need to set up a Funk/Chicken Scheme development environment on any platform.

Turn “Show Presenter Notes” on for more information.
Funk Development

• Chicken Scheme - http://www.callcc.org
• Eclipse - http://www.eclipse.org
• SchemeScript plugin for Eclipse
• REPL
• Funk Source Code
Install SchemeScript

• Install SchemeScript plugin
• Help > Software Updates > Find & Install
• Search for new features
• New Update Site:
  SchemeWay
  http://schemeway.sourceforge.net/update-site/
REPL

Compile with Chicken
and put resulting binary
in your project directory

csc -o remote_chicken remote_chicken.scm
Configuring Eclipse

- Add remote_chicken to External Tools
- Set SchemeScript to use Remote Interpreter
- Run remote_chicken from Run > External Tools
- Start Interpreter from Scheme > Start Interpreter
SchemeScript

Hotkeys

• Ctrl - Enter
  - Executes the preceding S-expression

• Ctrl - Shift - Enter
  - Executes the enclosing S-expression

• Ctrl - Shift - L
  - Loads current file in interpreter