We’re Going to the Lua

The Shift to a New Programming Language

Identifying and analyzing application protocols with Parsers written in the Lua scripting language.

RSA NetWitness
USER CONFERENCE

Introduction

- Lua Overview
- Decoder Overview
- Lua Parser API Overview
- Examples
- Roadmap

What is Lua?

- Lua is a powerful, fast, lightweight, embeddable scripting language.
- Features
  - Dynamically typed
  - Garbage collected
- Active community
- http://www.lua.org

What is and What isn’t Included

- Lua 5.1.5
- Standard Libraries
  - *string*, *table*, *math*, *coroutine*
  - But not *debug*, *io*, *os*, *package*
  - http://www.lua.org/manual/5.1
- Lua BitOp 1.0.2
  - Bitwise operations library
  - http://bitop.luajit.org

A Quick Decoder Process Overview

- Packet Capture
- Session Assembly
- Session Parse
- Packet, Meta and Session Persistence

Packet Capture

- Per Packet Network/Transport layer analysis
  - Determine source/destination addresses and ports
  - Identify application payload offset and size
- Network Rules
  - Executed for each packet
  - Can truncate/filter packets
- Packets that are not filtered are sent to the Session Assembly

Session Assembly

- Session State
  - Packets arriving from Capture Process are added to existing Sessions or create new Sessions
- Accumulate Packets until:
  - Session size exceeded (32MB)
  - Packet timeout reached (60 seconds)
  - Decoder resources exceeded
- Send Session to Session Parse

Session Parse

- Identify application protocols and extract meta information
  - Parsers
    System, Search, Snort Rules, Flex, Lua
  - Feeds
  - Application Rules
- Multiple Sessions independently parsed in parallel
- Send Packets, Session and Meta to Persistence Process

Packet, Session and Meta Persistence

- Packets, Session and Meta written to disk
- Sessions and Meta available to external processes
  - Concentrator Aggregation
  - SDK/REST API calls
  - Applications:
    Informer, Investigator, Security Analytics, Spectrum, Visualize, etc...

Identify DNS on port 5300

```
local name = "DnsAlt"
local description = "DNS Alternate Port Identification"

-- create the parser object
local dnsParser = nw:createParser(name, description)

-- define an event callback function
function dnsParser:onPort5300(portNumber)
        -- set the application type for this session to 53 (DNS)
        nw:setAppType(53)
end

-- define a table of event callbacks to functions
local callbacksTable = {
        -- integer keys indicate the associated function will be called for a
        -- matching port value
        [5300] = dnsParser.onPort5300
}

-- set the callbacks for this parser
dnsParser:setCallbacks(callbacksTable)
```

Parser Structure

- Defined in a single file (e.g. dnsalt.lua)
- Initialization
  - Create the parser object
- Define Event Handlers
  - Lua functions associated with the parser object
  - Implement parser specific logic
- Event Handler Registration
  - Maps parser object functions to specific events

Lua Parser API

- *nw*
  - Parser definition
  - Logging
  - Access to session and stream properties (e.g. source/destination, packet counts, payload bytes, etc…)
  - Meta creation
  - Application payload access via nwpayload objects
- *nwpayload*
  - Interface to the application payload of a stream
  - Implements a subset of the Lua *string* library (e.g. *byte*, *find*, *sub*, *equal*)
  - Numeric conversion functions
  - Packet payload scoping and iteration

Parser Execution

- Initialization
  - Occurs at system startup and parser reload
  - Lua file is executed
  - OnInit event is fired
- Capture start/stop
  - OnStart, OnStop
- Session/Stream Callbacks
  - OnReset, OnSessionBegin/End, OnStreamBegin/End
- Content Callbacks
  - Ports, tokens and meta callbacks

Parser State

- Each parser executes in its own environment
  - No direct references to state of other parsers
  - No interference with environment of other parsers
- Variable state maintained across sessions
  - Parser is responsible for initializing necessary values before session parsing (OnReset event)
  - nwpayload objects invalidated
  - A given parser instance will not see every session so using parser state to track statistics

Registering Event Callbacks

```
local myParser = nw:createParser("myParser", "Event Callbacks Example")

function myParser:onPort80(portNumber)          nw:logDebug("Found port 80!") end

function myParser:onToken(tokenId, first, last)  nw:logDebug("Found token!") end

function myParser:onSessionBegin()               nw:logDebug("Found session begin!") end

function myParser:onAlert(metaId, value)         nw:logDebug("Found alert!”) end

local callbacksTable = {
        [80]                          = myParser.onPort80,          -- port event
        ["GET /"]                     = myParser.onToken,           -- token event
        [nwevents.OnSessionBegin]     = myParser.onSessionBegin,    -- session event
        [nw:LanguageKey("alert”)]     = myParser.onAlert            -- meta callback
}

-- set the callbacks for this parser
myParser:setCallbacks(callbacksTable)
```

Creating Meta

```
local clientParser = nw:createParser("ClientParser", "Create client meta for User-Agent string.")

-- define the client language key
local lkClient = nw:LanguageKey("client")

-- set the language keys that this parser can create
clientParser:setKeys({lkClient})

-- define an event callback function
function clientParser:onUserAgent(token, first, last)
		-- create client meta of the first 10 bytes following the user agent header
		nw:createMeta(self.keys.client, last + 1, last + 10)
end

local callbacksTable = {
		["\r\nUser-Agent"] = clientParser.onUserAgent
}
clientParser:setCallbacks(callbacksTable)
```

Payloads and Parser State

```
-- token callback for "\n\rContent-type: "
function httpParser:onContentType(token, first, last)
        if not self.parsingHeaders then
                -- currently not parsing an HTTP header
                return
        end
        -- get a reference to the 50 bytes following the content type field name
        local payload = nw:getPayload(last + 1, last + 50)
        if self.foundResponse then
                -- an HTTP response header was encountered, this is response content
                local semi = payload:find(";")
                if semi then
                        payload = payload:sub(1, semi - 1)
                end
                nw:createMeta(self.keys.content, payload)
        else
                -- this is a request content type, check if the content is a post
                if self.request.foundPostQuery and
                  not payload:find("application/x-www-form-urlencoded") then
                        self.request.foundPostQuery = false
                end
        end
end
```

Roadmap

- Available with the release of 9.8, Service Pack 1
- Future Work
  - Performance!
  - Parser development tools
  - Parser performance metrics
  - Content
    - Migrate native parsers where applicable
    - Implement new parsers for availability via Live
  - Expand current API
    - Packet level analysis
    - Suggestions?

NetWitness Community

- Please visit the relaunch of the NetWitness Community
  - https://community.emc.com/go/netwitness
  - Ask questions and get answers straight from the NetWitness developers
  - Post ideas for new features
    - Help contribute to the future direction of the product!

RSA NetWitness
USER CONFERENCE

Thank you.