OpenAppID

Open Source Community Webinar

Costas Kleopa, OpenAppID Development Manager

Cliff Judge, OpenAppID Engineer

snort-openappid@lists.sourceforge.net

Released: February 4, 2015

Agenda

- Introduction to OpenAppID
- Use Cases
- Roadmap
- Open Source Community

Introduction

OpenAppID – First OSS Application & Control

- OpenAppID Language Documentation
  - Based on Lua programming language
- Part of Snort as of 2.9.7
  - Additional configurable option for functionality to be enabled
- Capabilities
  - Identify and Control Applications over network
  - Report Usage stats
  - Snort rule language supported
- Library of OpenAppID Detectors
  - Over 2600 new detectors to use
  - Open Sourced, extendable sample detectors

OpenAppID – Capabilities

- Lua JIT
  - High performance, low memory footprint, scripting language
  - Flexible, Multi-Platform, Used by multiple products/projects around the world
  - Also included in Snort 3.0 Alpha release
    - As part of the new configurations and Rule capabilities
- Applications Detections based on Patterns in traffic
  - HTTP, SSL, SIP, RTMP/RTSP
- Other capabilities
  - Future Flow support
  - IPv6 support
  - State tracking

Use Cases

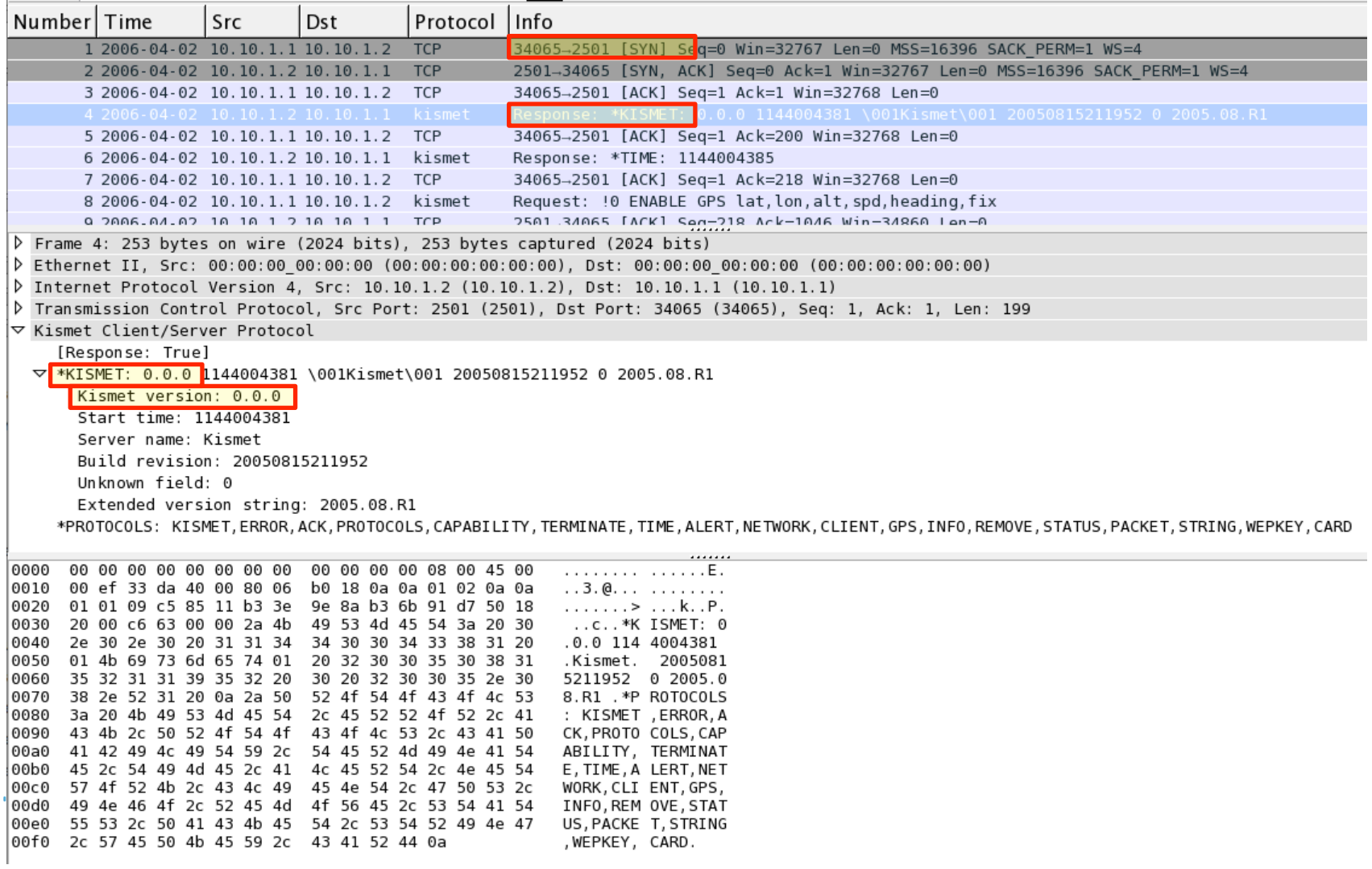
API Walkthrough

- https://www.snort.org/downloads
  - OpenAppID: OpenDetectorDeveloperGuide.pdf
- Pattern Based Detection
  - HTTP Protocol
    - void open_addUrlPattern (serviceAppId, clientAppId, payloadAppId, hostpattern, pathPattern, schemePattern)
    - `gDetector:open_addUrlPattern(0, 0, gAppId, "nbc.com", "/", "http:");`
  - SSL Protocol
    - void addSSLCertPattern (type, appId, pattern)
    - `gDetector:addSSLCertPattern("0", gAppId, "facebook.com");`
  - TCP/UDP Packet matching
    - int matchSimplePattern (pattern, patternSize, position)
    - `if (gDetector:matchSimplePattern(”\000\002\003", 3, 0))`

Detector Example:

- Detector Creation Plan
  - Capture Traffic of that application
  - Identify the patterns/behaviors that are unique
    - Create additional Fast Patterns to help performance
  - Make sure the traffic patterns are unique with multiple traffic scenarios
- Capture Traffic Examples
  - Kismet detector
  - Minecraft detector

Detector Case: Kismet


Number Time Src Dst Protocol Info
1 2006-04-02 10.10.1.1 10.10.1.2 TCP 34065→2501 [SYN] Seq=0 Win=32767 Len=0 MSS=16396 SACK_PERM=1 WS=4
2 2006-04-02 10.10.1.2 10.10.1.1 TCP 2501→34065 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MSS=16396 SACK_PERM=1 WS=4
3 2006-04-02 10.10.1.1 10.10.1.2 TCP 34065→2501 [ACK] Seq=1 Ack=1 Win=32768 Len=0
4 2006-04-02 10.10.1.2 10.10.1.1 kismet Response: *KISMET: 0.0.0 1144004381 \001Kismet\001 20050815211952 0 2005.08.R1
5 2006-04-02 10.10.1.1 10.10.1.2 TCP 34065→2501 [ACK] Seq=1 Ack=200 Win=32768 Len=0
6 2006-04-02 10.10.1.2 10.10.1.1 kismet Response: *TIME: 1144004385
7 2006-04-02 10.10.1.1 10.10.1.2 TCP 34065→2501 [ACK] Seq=1 Ack=218 Win=32768 Len=0
8 2006-04-02 10.10.1.1 10.10.1.2 kismet Request: !0 ENABLE GPS lat,lon,alt,spd,heading,fix
9 2006-04-02 10.10.1.2 10.10.1.1 TCP 2501→34065 [ACK] Seq=218 Ack=1046 Win=34860 Len=0
Frame 4: 253 bytes on wire (2024 bits), 253 bytes captured (2024 bits)
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 10.10.1.2 (10.10.1.2), Dst: 10.10.1.1 (10.10.1.1)
Transmission Control Protocol, Src Port: 2501 (2501), Dst Port: 34065 (34065), Seq: 1, Ack: 1, Len: 199
Kismet Client/Server Protocol
[Response: True]
*KISMET: 0.0.0 1144004381 \001Kismet\001 20050815211952 0 2005.08.R1
Kismet version: 0.0.0
Start time: 1144004381
Server name: Kismet
Build revision: 20050815211952
Unknown field: 0
Extended version string: 2005.08.R1
*PROTOCOLS: KISMET,ERROR,ACK,PROTOCOLS,CAPABILITY,TERMINATE,TIME,ALERT,NETWORK,CLIENT,GPS,INFO,REMOVE,STATUS,PACKET,STRING,WEPKEY,CARD
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 ....E.
0010 00 ef 33 da 40 00 80 06 b0 18 0a 0a 01 02 0a 0a ..3.@... .....
0020 01 01 09 c5 85 11 b3 3e 9e 8a b3 6b 91 d7 50 18 .....> ...k..P.
0030 20 00 c6 63 00 00 2a 4b 49 53 4d 45 54 3a 20 30 ..c..*K ISMET: 0
0040 2e 30 2e 30 20 31 31 34 34 30 30 34 33 38 31 20 .0.0 114 4004381
0050 01 4b 69 73 6d 65 74 01 20 32 30 30 35 30 38 31 .Kismet. 2005081
0060 35 32 31 31 39 35 32 20 30 20 32 30 30 35 2e 30 5211952 0 2005.0
0070 38 2e 52 31 20 0a 2a 50 52 4f 54 4f 43 4f 4c 53 8.R1 .*P ROTOCOLS
0080 3a 20 4b 49 53 4d 45 54 2c 45 52 52 4f 52 2c 41 : KISMET ,ERROR,A
0090 43 4b 2c 50 52 4f 54 4f 43 4f 4c 53 2c 43 41 50 CK,PROTO COLS,CAP
00a0 41 42 49 4c 49 54 59 2c 54 45 52 4d 49 4e 41 54 ABILITY, TERMINAT
00b0 45 2c 54 49 4d 45 2c 41 4c 45 52 54 2c 4e 45 54 E,TIME,A LERT,NET
00c0 57 4f 52 4b 2c 43 4c 49 45 4e 54 2c 47 50 53 2c WORK,CLI ENT,GPS,
00d0 49 4e 46 4f 2c 52 45 4d 4f 56 45 2c 53 54 41 54 INFO,REM OVE,STAT
00e0 55 53 2c 50 41 43 4b 45 54 2c 53 54 52 49 4e 47 US,PACKE T,STRING
00f0 2c 57 45 50 4b 45 59 2c 43 41 52 44 0a ,WEPKEY, CARD.

Lua Implementation

```
local DC = DetectorCommon

DetectorPackageInfo = {
    name =  "Kismet",
    proto =  DC.ipproto.tcp,
    server = {
        init =  'DetectorInit',
        validate =  'DetectorValidator',
    }
}

gServiceId = 20140
gServiceName = 'Kismet'
gSfAppIdKismet = 1451

gPorts = {
    {DC.ipproto.tcp, 2501},
}

gPatterns = {
    server_resp = {'KISMET:', 1, gSfAppIdKismet},
}

gFastPatterns = {
    {DC.ipproto.tcp, gPatterns.server_resp},
}

gAppRegistry = {
      --AppIdValue          Extracts Info
      ----------------------------------------
      {gSfAppIdKismet,                    0}
}
```

Lua Implementation

```
function serviceInProcess(context)

    local flowFlag = context.detectorFlow:getFlowFlag(DC.flowFlags.serviceDetected)

    if ((not flowFlag) or (flowFlag == 0)) then
        gDetector:inProcessService()
    end

    DC.printf('%s: Inprocess, packetCount: %d\n', gServiceName, context.packetCount);
    return DC.serviceStatus.inProcess
end

function serviceSuccess(context)
    local flowFlag = context.detectorFlow:getFlowFlag(DC.flowFlags.serviceDetected)

    if ((not flowFlag) or (flowFlag == 0)) then
        gDetector:addService(gServiceId, "", context.version, gSfAppIdKismet)
    end

    DC.printf('%s: Detected, packetCount: %d\n', gServiceName, context.packetCount);
    return DC.serviceStatus.success
end

function serviceFail(context)
    local flowFlag = context.detectorFlow:getFlowFlag(DC.flowFlags.serviceDetected)

    if ((not flowFlag) or (flowFlag == 0)) then
        gDetector:failService()
    end

    context.detectorFlow:clearFlowFlag(DC.flowFlags.continue)
    DC.printf('%s: Failed, packetCount: %d\n', gServiceName, context.packetCount);
    return DC.serviceStatus.nomatch
end
```

Lua Implementation

```
function registerPortsPatterns()

    --register port based detection
    for i,v in ipairs(gPorts) do
        gDetector:addPort(v[1], v[2])
        DC.printf('%s: registering port %d\n',gServiceName,v[2])
    end

    --register pattern based detection
    for i,v in ipairs(gFastPatterns) do
        if ( gDetector:registerPattern(v[1], v[2][1], #v[2][1], v[2][2], v[2][3]) ~= 0) then
            DC.printf ('%s: register pattern failed for %s\n', gServiceName,v[2][1])
        else
            DC.printf ('%s: register pattern successful for %s\n', gServiceName,v[2][1])
        end
    end

       for i,v in ipairs(gAppRegistry) do
             pcall(function () gDetector:registerAppId(v[1],v[2]) end)
       end
end

function DetectorInit( detectorInstance)

    --print (gServiceName .. ': DetectorInit()')

    gDetector = detectorInstance
    gDetector:init(gServiceName, 'DetectorValidator', 'DetectorFini')
    registerPortsPatterns()

    return gDetector
end
```

Lua Implementation

```
function DetectorValidator()
    local context = {}
    context.detectorFlow = gDetector:getFlow()
    context.packetDataLen = gDetector:getPacketSize()
    context.packetDir = gDetector:getPacketDir()
    context.srcIp = gDetector:getPktSrcAddr()
    context.dstIp = gDetector:getPktDstAddr()
    context.srcPort = gDetector:getPktSrcPort()
    context.dstPort = gDetector:getPktDstPort()
    context.flowKey = context.detectorFlow:getFlowKey()
    context.packetCount = gDetector:getPktCount()
    local size = context.packetDataLen
    local dir = context.packetDir
    local srcPort = context.srcPort
    local dstPort = context.dstPort
    local flowKey = context.flowKey

    DC.printf('%s:DetectorValidator(): packetCount %d, dir %d, size %d, srcPort %d\n', gServiceName, context.packetCount, dir,
size, srcPort)

    if (size == 0) then
        return serviceInProcess(context)
    end

    if (size >= 14 and (gDetector:matchSimplePattern (gPatterns.server_resp[1], #gPatterns.server_resp[1],
gPatterns.server_resp[2]) == 0)) then
        DC.printf('%s: got a server_resp packet\n', gServiceName)
        matched, ver = gDetector:getPcreGroups('KISMET: (.*) ', 1)
        if (matched) then
            context.version = ver
        end
        return serviceSuccess(context)
    end
```

Detector Case: Minecraft

Number	Time	Src	Dst	Protocol	Info
1	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=105679782 TSecr=
2	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1018
3	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=105679783 TSecr=101889070
4	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=20 TSval=105679783 TSecr=10188907
5	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [ACK] Seq=1 Ack=21 Win=29056 Len=0 TSval=101889070 TSecr=105679783
6	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [PSH, ACK] Seq=21 Ack=1 Win=29312 Len=9 TSval=105679783 TSecr=10188907
7	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [ACK] Seq=1 Ack=30 Win=29056 Len=0 TSval=101889070 TSecr=105679783
8	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [PSH, ACK] Seq=1 Ack=30 Win=29056 Len=173 TSval=101889070 TSecr=105679
9	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [ACK] Seq=30 Ack=174 Win=30336 Len=0 TSval=105679783 TSecr=101889070
10	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [PSH, ACK] Seq=30 Ack=174 Win=30336 Len=263 TSval=105679844 TSecr=1018

```
▷ Frame 8: 239 bytes on wire (1912 bits), 239 bytes captured (1912 bits)
▷ Ethernet II, Src: Vmware_98:e8:99 (00:0c:29:98:e8:99), Dst: Vmware_ec:91:1a (00:0c:29:ec:91:1a)
▷ Internet Protocol Version 4, Src: 192.168.1.120 (192.168.1.120), Dst: 192.168.1.119 (192.168.1.119)
▷ Transmission Control Protocol, Src Port: 12345 (12345), Dst Port: 54050 (54050), Seq: 1, Ack: 30, Len: 173
▷ Data (173 bytes)
```

```
0000  00 0c 29 ec 91 1a 00 0c  29 98 e8 99 08 00 45 00   ..)..... ).....E.
0010  00 e1 ca 27 40 00 40 06  eb af c0 a8 01 78 c0 a8   ...'@.@. .....x..
0020  01 77 30 39 d3 22 36 ff  7e 0b 1f 35 c1 d5 80 18   .w09."6. ~..5....
0030  00 e3 73 35 00 00 01 01  08 0a 06 12 b4 2e 06 4c   ..s5.... .......L
0040  8b a7 ab 01 01 00 a2 01  30 81 9f 30 0d 06 09 2a   ........ 0..0...*
0050  86 48 86 f7 0d 01 01 01  05 00 03 81 8d 00 30 81   .H...... ......0.
0060  89 02 81 81 00 85 51 d8  f6 cb ba 58 52 d4 76 ef   ......Q. ...XR.v.
0070  ea b7 5f 15 0e 7a 89 8c  5b cd dc 76 bb 5f 76 b7   .._..z.. [..v._v.
0080  ec a7 56 37 d2 13 03 4b  eb 6a ab 67 02 c3 22 32   ..V7...K .j.g.."2
0090  fa 70 45 0a 1e 2b 54 7b  a3 05 b6 cc bf f3 7d a2   .pE..+T{ ......}.
00a0  07 2b a2 74 65 c7 eb da  5c 12 74 75 ad 7c d8 4a   .+.te... \.tu.|.J
00b0  5c 17 0c 6a ae 4a b0 72  dc c0 db f9 f8 77 c2 1f   \..j.J.r .....w..
00c0  01 5f 35 52 81 5a 80 95  14 f8 a3 7f 9f 25 4b 90   ._5R.Z.. .....%K.
00d0  05 76 31 77 41 27 c9 94  20 76 7a f3 7a bc 5f 67   .v1wA'.. vz.z._g
00e0  d2 8c 30 ed 39 02 03 01  00 01 04 8f 79 21 65      ..0.9... ....y!e
```

```
\171\001\001\000\162\001\048\129\159\048\013\006\
009\042\134\072
```

Number	Time	Src	Dst	Protocol	Info
1	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=105679782 TSecr=0
2	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=101889
3	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=105679783 TSecr=101889070
4	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=20 TSval=105679783 TSecr=101889070
5	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [ACK] Seq=1 Ack=21 Win=29056 Len=0 TSval=101889070 TSecr=105679783
6	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [PSH, ACK] Seq=21 Ack=1 Win=29312 Len=9 TSval=105679783 TSecr=101889070
7	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [ACK] Seq=1 Ack=30 Win=29056 Len=0 TSval=101889070 TSecr=105679783
8	2014-10-28	192.168.1.120	192.168.1.119	TCP	12345→54050 [PSH, ACK] Seq=1 Ack=30 Win=29056 Len=173 TSval=101889070 TSecr=10567978
9	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [ACK] Seq=30 Ack=174 Win=30336 Len=0 TSval=105679783 TSecr=101889070
10	2014-10-28	192.168.1.119	192.168.1.120	TCP	54050→12345 [PSH, ACK] Seq=30 Ack=174 Win=30336 Len=263 TSval=105679844 TSecr=101889

- Frame 10: 329 bytes on wire (2632 bits), 329 bytes captured (2632 bits)
- Ethernet II, Src: Vmware_ec:91:1a (00:0c:29:ec:91:1a), Dst: Vmware_98:e8:99 (00:0c:29:98:e8:99)
- Internet Protocol Version 4, Src: 192.168.1.119 (192.168.1.119), Dst: 192.168.1.120 (192.168.1.120)
- Transmission Control Protocol, Src Port: 54050 (54050), Dst Port: 12345 (12345), Seq: 30, Ack: 174, Len: 263
- Data (263 bytes)

```
0040  b4 2e 85 02 01 80 01 2b  c7 06 54 be aa 63 d0 40   ......+ ..T..c.@
0050  07 c5 d8 27 94 75 01 b6  47 a9 de 65 75 57 4a 3f   ...'.u.. G..euWJ?
0060  09 dd 0d 0f 0a 33 a9 05  c1 a8 58 73 0e 76 6b 34   .....3.. ..Xs.vk4
0070  9c 59 fa c1 b8 67 85 55  35 22 eb 45 11 56 0b f3   .Y...g.U 5".E.V..
0080  52 2b 7b b5 79 70 f6 0c  2d 79 4c 06 73 c0 19 b2   R+{.yp.. -yL.s...
0090  4f 57 e3 f4 1d 55 85 6b  e3 83 3a 45 fd a8 5c f1   OW...U.k ..:E..\.
00a0  1e e2 19 77 d4 dd 1b f3  b4 de 28 a8 76 b6 03 96   ...w.... ..(.v...
00b0  ad 7e 0d 42 6f 2d d3 ba  84 6b c1 53 14 f6 ec e3   .~.Bo-.. .k.S....
00c0  f1 5b 9d ee ff e6 d2 80  01 0a 8f 99 46 5c 33 10   .[...... ....F\3.
00d0  ba d6 db 02 b0 14 8a 6d  96 c0 fb 99 4d 36 aa ef   .......m ....M6..
00e0  38 34 68 02 06 7c 7f be  49 11 65 4c 91 d0 c0 3a   84h..|.. I.eL...:
00f0  58 be 35 5c bb a8 06 b5  07 f2 c5 a5 11 ac 0d a6   X.5\.... ........
0100  17 94 6e 44 d4 c0 ce 0d  65 ae 47 92 36 6e 7b 5e   ..nD.... e.G.6n{^
0110  13 46 7b 70 82 19 0c f3  37 75 8e a3 6b 00 d3 18   .F{p.... 7u..k...
0120  0e d8 fd bc f8 c7 80 c2  0a e0 6a dc 94 f5 11 bc   ........ ..j.....
0130  de b4 cb 81 af 10 9a e4  25 60 57 c2 5f bb 93 44   ........ %`W._..D
```

\133\002\001\128\001

Minecraft Lua Detector: Header/Initialization

```
--patterns used in DetectorInit()
gPatterns = {
    --patternName        Pattern                                        offset
    ------------------------------------------------------------------------
    srv      = {'\171\001\001\000\162\001\048\129\159\048\013\006\009\042\134\072', 0, gSfAppIdMinecraft},
    cli      = {'\133\002\001\128\001', 0, gSfAppIdMinecraft},
}

gFastPatterns = {
    --protocol       patternName
    ------------------------------------
    {DC.ipproto.tcp, gPatterns.srv},
}


function registerPortsPatterns()

    --register pattern based detection
    for i,v in ipairs(gFastPatterns) do
        if ( gDetector:registerPattern(v[1], v[2][1], #v[2][1], v[2][2], v[2][3]) ~= 0) then
            --print (gServiceName .. ': register pattern failed for ' .. v[2])
        else
            --print (gServiceName .. ': register pattern successful for ' .. i)
        end
    end

      for i,v in ipairs(gAppRegistry) do
            pcall(function () gDetector:registerAppId(v[1],v[2]) end)
      end

end
```

Minecraft Lua Detector: Validator

```
function DetectorValidator()
…
 local dir = context.packetDir
 local rft = FT.getFlowTracker(flowKey)
 if (size == 0) then
        return serviceInProcess(context)
    end

    DC.printf ('%s:DetectorValidator(): packetCount %d, dir %d\n', gServiceName, context.packetCount, dir);

    if (not rft) then
        rft = FT.addFlowTracker(flowKey, {server_received=0})
    end

    if (dir == DC.flowDirection.fromResponder and
        rft.server_received == 0 and
        gDetector:matchSimplePattern (gPatterns.srv[1], #gPatterns.srv[1], gPatterns.srv[2]) == 0) then
            rft.server_received = 1
        return serviceInProcess(context)
    end

    if (dir == DC.flowDirection.fromInitiator and
        rft.server_received == 1 and
        gDetector:matchSimplePattern(gPatterns.cli[1], #gPatterns.cli[1], gPatterns.cli[2]) == 0) then
            FT.delFlowTracker(flowKey)
        return serviceSuccess(context)
    end

    FT.delFlowTracker(flowKey)
    return serviceFail(context)
end
```

Roadmap

OpenAppID Roadmap

- OpenAppID Refactoring
  - Plan to open more APIs for broader application identification
  - DNS based pattern matching
  - IP Address, Port Numbers, Protocols, Packet Length & Direction based detectors patterns
  - Ability to combine multiple patterns and combinations of other AppIDs to form a Detector
- Adding appid keyword in other snort log formats like cmg
- Adding more fields in AppStats for better connection tracking
- appMapping.data appid column will be deprecated
  - 638 Firefox 0 32 0 ~ firefox
  - appid will be the full name of the detector
  - spaces will be allowed as part of the appid name
  - appid's name will not be case sensitive

Open Source Community

Open Source Community

- Monthly Open Source releases of Detectors
- Open to New Detector requests/features
- Get involved in creating Application Detectors
  - snort-openappid@lists.sourceforge.net
- All new updates will be available in the next major release of Snort.

TOMORROW starts here.