http://thisissecurity.net/2015/11/23/hackers-do-the-haka-part-1/
(http://thisissecurity.net/2015/11/23/hackers-do-the-haka-part-1/)

Haka是一种开源的以网络安全为导向的语言，可以用于编写安全规则和协议剥离器。在这一部分中，我们的重点是编写安全规则。

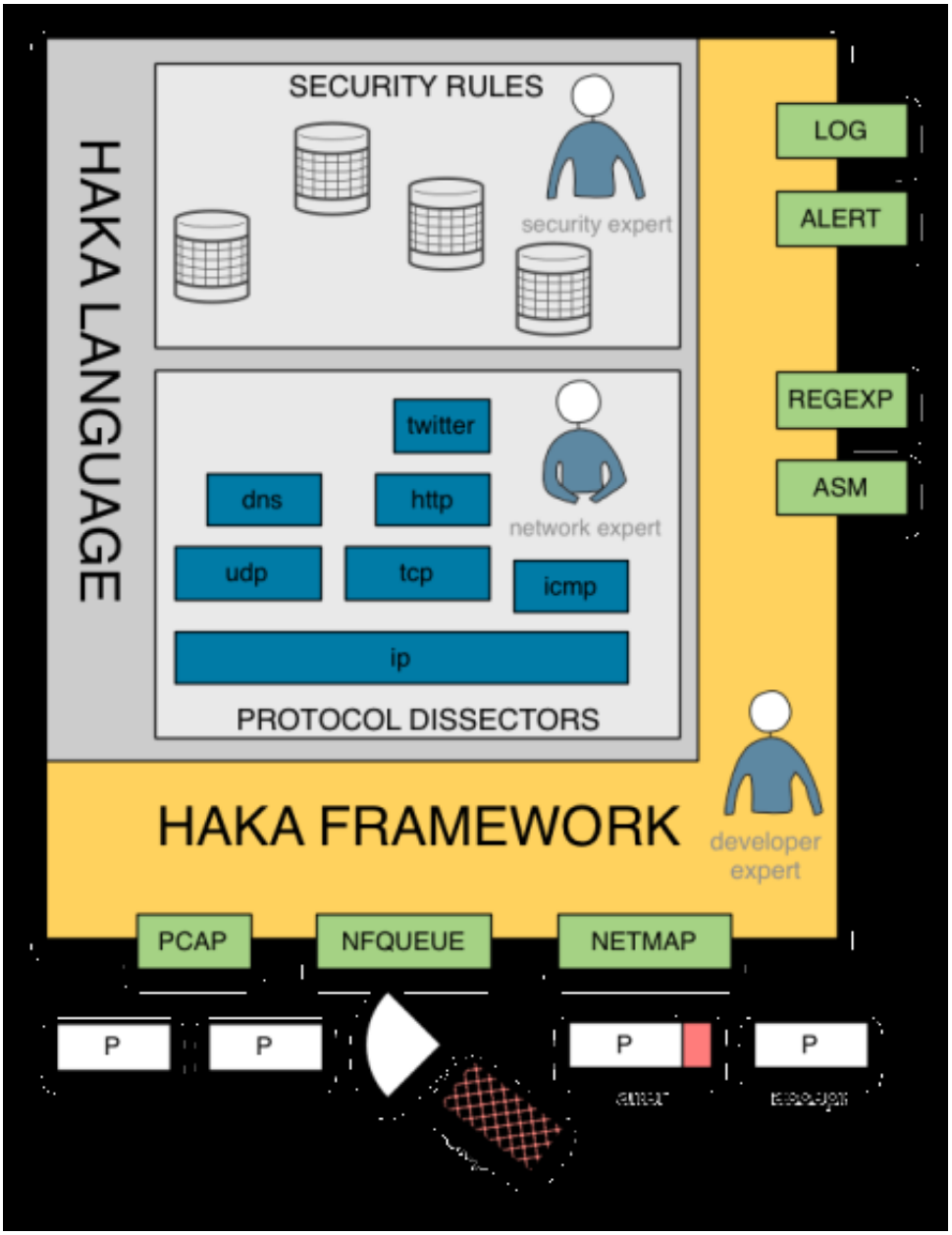
0x00 什么是Haka

Haka是一种开源的以网络安全为导向的语言，可以在捕捉到的活动流量上指定和应用安全策略。Haka基于Lua。这是一种简单、轻量（~200 kB）且快速的（有一个可用的JiT编译器）脚本语言。

Haka的使用范围有两种。首先，Haka能允许指定安全规则来过滤掉不想要的流并报告恶意活动。Haka提供了一个简单的API用于操作高级数据包和流。用户可以投放数据包，或创建新的数据包，并注入数据包。Haka还支持修改运行中的数据包。这是Haka的主要功能之一，因为用户可以看到所有的复杂任务，比如调整数据包的大小，设置正确的序列号。这些都是现场完成的，根本不需要代理。

其次，Haka还附带有一种语法，允许协议规范及其底层的状态机。Haka支持两种类型的协议：基于二进制的协议（如，dns）和基于文本的协议（如，http）。这种规范包括了基于数据包的协议（如ip）以及基于流的协议（如http）。

Haka内嵌到了一个模块化框架中，其中包括几个数据包捕捉模块（pcap, nfqueue），能允许终端用户在实时捕捉到的流量上应用自己的安全策略，或者是在一个数据包跟踪文件上重现这个安全策略。这个框架还提供了记录（syslog）和报警模块（syslog，elasticsearch）。报警遵循了一种类似IDMED的格式。最后，这个框架还具有附属模块，比如模式匹配引擎和一个指令反汇编模块。这些模块允许编写精密的安全规则来检测经过混淆的木马。因为Haka是按照模块方式设计的，所以用户可以通过额外的模块来扩展其功能。



0x01 Haka工具套件

Haka提供了一个包含有4个工具的套件：

- haka.这是工具集中的主程序。这是一个守护进程，用于监控后台中的数据包。Haka会根据指定的安全策略文件来分析和过滤这些数据包。例如，下面的配置样本会要求Haka使用nfqueue模块从接口eth0中捕捉数据包，并使用策略文件myrules.lua过滤这些数据包。这个脚本通常会加载用户定义或内置的协议剥离器，并定义一系列的安全规则。另外，用户可以选择报警和报告模块并设置一些特定的模块选项：

```
[general]
# Select the haka configuration file to use
configuration = "myrules.lua"

# Optionally select the number of thread to use
# By default, all system thread will be used
#thread = 4

[packet]
# Select the capture model, nfqueue or pcap
module = "packet/nfqueue"

# Select the interfaces to listen to
#interfaces = "any"
interfaces = "eth0"

# Select packet dumping for nfqueue
#dump = yes
#dump_input = "/tmp/input.pcap"
#dump_output = "/tmp/output.pcap"

[log]
# Select the log module
module = "log/syslog"

# Set the default logging level
#level = "info,packet=debug"

[alert]
# Select the alert module
module = "alert/syslog"
#module = "alert/file"
#module = "alert/elasticsearch"

# Disable alert on standard output
#alert_on_stdout = no
```

- hakactl. 这个工具允许控制一个运行的Haka守护进程。用户可以实时分析捕捉到的数据包，检查日志或简单地关停/重启守护进程。

- hakapcap. 这个工具允许使用pcap模块在数据包捕捉跟踪上离线重现一个策略文件。例如，这样对执行网络分析非常有用。

- hakabana.这个工具允许使用Kibana 和 Elasticsearch实时地可视化和监控网络流量。Hakabana由一系列的自定义安全规则组成，这些规则会把流量信息通过Haka推送到一个elastisserach服务器，并让这些流量能通过Kibana控制面板获取到。另外一个控制板也可以可视化Haka警报。

0x02 编写安全规则

Haka提供了一种简单的方式来编写安全规则，从而过滤，修改，创建，注入数据包和流。当检测到一个恶意流后，用户可以报告一个警报或放弃这个流。用户可以定义更多复杂的方案来应对某次攻击所造成的影响。例如，用户可以报警http请求，强制旧版浏览器更新或伪造特定的数据包来欺骗端口扫描工具。

0x03 数据包过滤

下面的这个规则是一个基础的数据包过滤规则，能够拦截所有到某个网络地址的连接。

```
local ipv4 = require("protocol/ipv4")
local tcp = require("protocol/tcp_connection")

local net = ipv4.network("192.168.101.0/24")

haka.rule{
    hook = tcp.events.new_connection,
    eval = function (flow, pkt)
        haka.log("tcp connection %s:%i -> %s:%i",
            flow.srcip, flow.srcport,
            flow.dstip, flow.dstport)

        if net:contains(flow.dstip) then
            haka.alert{
                severity = "low",
                description = "connection refused",
                start_time = pkt.ip.raw.timestamp
            }
            flow:drop()
        end
    end
}
```

第一行会加载需要的协议剥离器，也就是ipv4和tcp连接剥离器。前者是处理ipv4数据包。后者是一个状态性的tcp剥离器，这个剥离器会维持一个连接表并管理tcp流。下一行，定义了必须要拦截的网络地址。

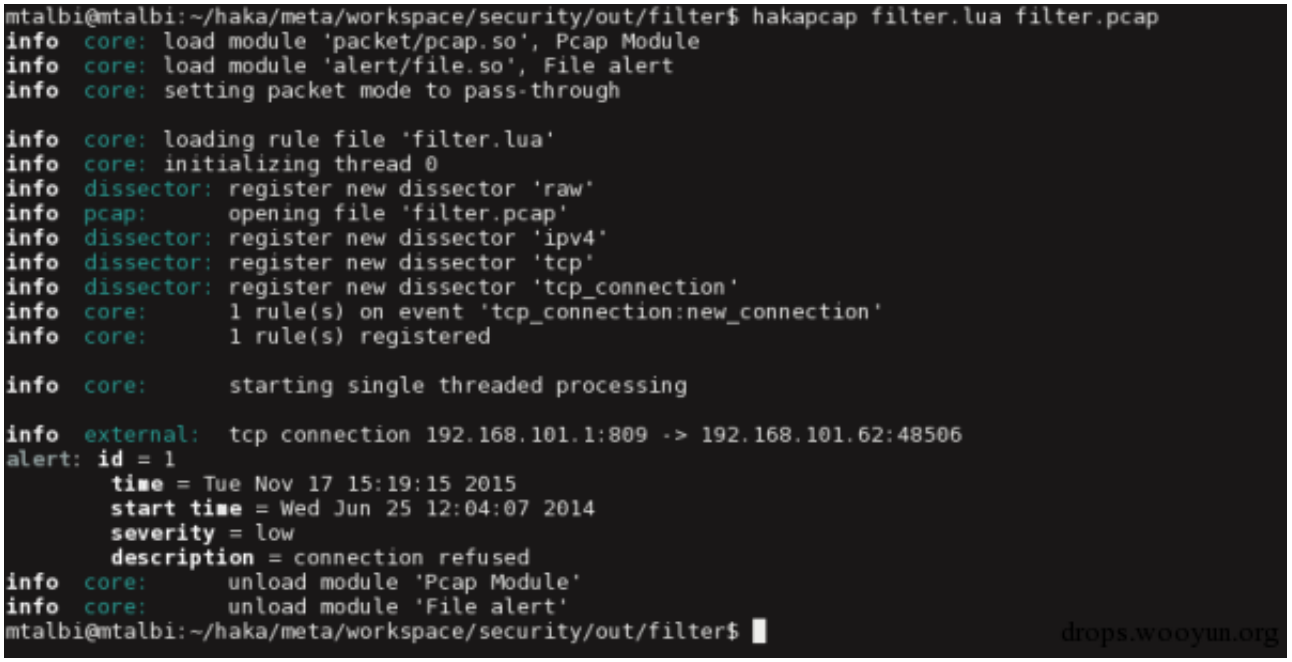
这种安全规则是通过haka.rule关键字定义的。一个安全规则由一个hook和一个评估函数eval组成。这个hook是一个会触发安全规则评估的事件。在这个例子中，在每次尝试创建一个tcp连接时，这个安全规则就会被评估。传递到评估函数的参数会根据事件来确定。以new_connection事件为例，eval会获取两个参数：flow和pkt。第一个参数中会有关于连接的详细信息，另一个参数一个包含有所有tcp（以及下层）数据包字段的表。

在安全规则的核心中，我们首先记录（haka.log）了一些关于当前连接的信息。然后，我们检查了源地址是否属于先前定义的非授权IP地址范围。如果测试成功，我们就会发出警报（haka.alert）并放弃连接。注意，我们只在警报中报告了少量的一些细节。用户可以添加更多信息，比如来源和目标服务。

我们使用hakapcap工具在一个pcap追踪文件filter.pcap上，测试了我们的规则filter.lua (https://thisiscybersec.files.wordpress.com/2015/11/filter.zip)：

```
$ hakapcap filter.lua filter.pcap
```

从这里往下就是Haka的输出，Haka转储了一些关于已加载剥离器和已注册规则的信息。这个输出显示Haka成功拦截了目标地址192.168.101.62的连接：



在上面的例子中，我们已经定义了一个单独的规则来拦截连接。用户可以用关键字haka.group写一个类似于防火墙一样的完整规则集。以这个配置为例，如果没有任何安全规则能认证流量，用户可以选择一种默认行为（如拦截所有的连接）。

0x04 数据包注入

在Haka中，用户可以创建新的数据包并注入这些数据包。下面的这个规则会制作一个RST数据包，用于欺骗Xmas nmap扫描。最终，nmap会认为目标端上的所有端口都是关闭的。

```
raw = require("protocol/raw")
ipv4 = require("protocol/ipv4")
tcp = require("protocol/tcp")

haka.rule {
    hook = tcp.events.receive_packet,
    eval = function(pkt)
        local flags = pkt.flags
        -- test for xmas nmap scans
        if flags.fin and flags.psh and flags.urg then
            -- raw packet
            local rstpkt = raw.create()

            -- ip packet
            rstpkt = ipv4.create(rstpkt)
            rstpkt.ttl = pkt.ip.ttl
            rstpkt.dst = pkt.ip.src
            rstpkt.src = pkt.ip.dst

            -- tcp packet
            rstpkt = tcp.create(rstpkt)
            rstpkt.srcport = pkt.dstport
            rstpkt.dstport = pkt.srcport
            rstpkt.flags.rst = true
            rstpkt.flags.ack = true
            rstpkt.ack_seq = pkt.seq + 1

            -- inject forged packet and
            -- drop malicious scanning packet
            rstpkt:send()
            pkt:drop()
        end
    end
}
```

0x05 数据包警报

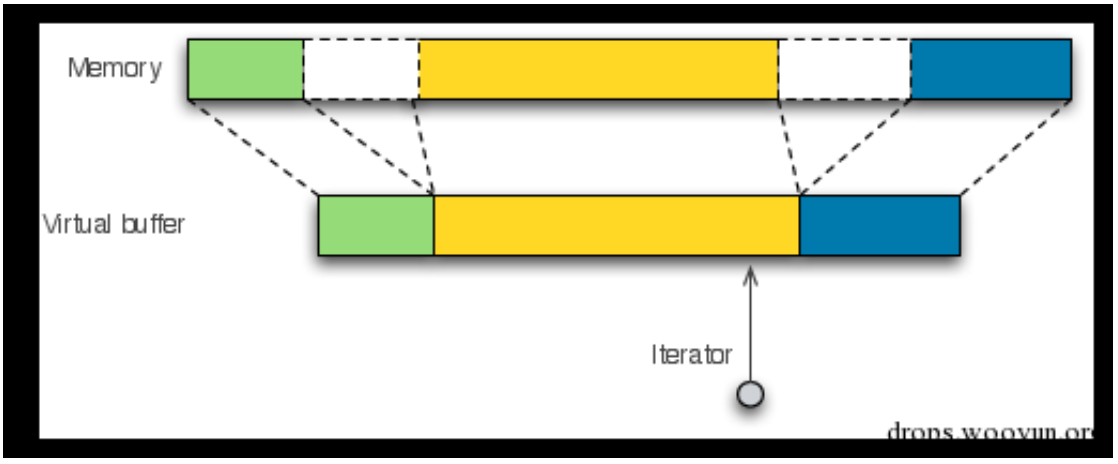
数据包修改是Haka中最高级的功能之一。Haka会在流和数据包层级上自动处理所有的内部修改：重新设置数据包大小和分段，重置序列号等。下面的例子证明了访问和修改协议字段有多么容易。这个规则会修改一些http协议的标头。更准确的说， user-agent标头会被修改（如果没有设置，添加至标头列表），并移除accept-encoding标头。

```
local http = require("protocol/http")

http.install_tcp_rule(80)

haka.rule{
    hook = http.events.request,
    eval = function (flow, request)
        request.headers["User-Agent"] = "HAKA User Agent"
        request.headers["Accept-Encoding"] = nil
    end
}
```

blurring-the-web (https://www.paulfariello.fr/assets/blurring-the-web.lua)和inject_ponies (https://thisiscybersec.files.wordpress.com/2015/11/inject-ponies.zip)是非常有意思的脚本，会通过修改http响应流量来分别模糊和污染（注入垃圾）被请求的web页面：

0x06 流过滤

在讲解流过滤前，我们首先要说明的是Haka如何在内部管理数据包和流。在Haka中，所有的数据包和流都会用虚拟缓冲区（见下图）来表示。虚拟缓冲区是一种非相邻内存块的统一视图。这些虚拟缓冲区能允许简单和有效的修改内存数据。虚拟缓冲区使用分散列表来表示非相邻数据块，避免分配和复制不必要的内存块。Haka提供了迭代器来导航通过这些内存块。这些迭代器可以通过拦截让停用一些函数，然后，比如当流上有更多可用的数据时，透明地恢复这些函数的执行。



下面的规则会收集http流，并在stdout上转储这些流。这个规则等同于Wireshark的follow tcp stream”功能。

```
local ipv4 = require('protocol/ipv4')
local tcp_connection = require('protocol/tcp_connection')

haka.rule{
    hook = tcp_connection.events.receive_data,
        options = {
            streamed = true
        },
    eval = function (flow, iter, dir)
        local data = flow.ccdata or {}
        flow.ccdata = data

        while iter:wait() do
            data[#data+1] = iter:sub('available'):asstring()
        end
        haka.log("%s -> %s:\n", flow.srcip, flow.dstip)
        io.write(table.concat(data))
     end
}
```

0x07 交互式数据包过滤

这是我最喜欢的Haka功能。这个功能允许检查每个数据包的流量包。下面的规则会提示每个HTTP POST请求。

```
local http = require("protocol/http")

http.install_tcp_rule(80)

haka.rule {
    hook = http.events.request_data,
    eval = function (http, data)
        haka.interactive_rule("interactive mode")(http, data)
    end
}

haka.rule {
    hook = http.events.request,
    eval = function (http, request)
        http:enable_data_modification()
    end
}
```

这个shell能够提供到完整Haka API的访问，以便处理数据包内容：读写和修改数据包字段，删除数据包，记录可疑的事件，警告等。Lua控制板支持自动补齐，因此，是深入Haka API的一个好的开始。

在下面的数据中，Haka会破解第一个POST请求。HTTP数据可以通过可用的输入来获取。在这个例子中，我们在运行中修改了用户凭据。

```
root@mtalbi:/home/mtalbi/haka/meta/workspace/security/out# haka -c haka.conf --no-daemon
info  core: load module 'log/syslog.so', Syslog logger
info  core: load module 'alert/syslog.so', Syslog alert
info  core: load module 'alert/file.so', File alert
info  core: load module 'packet/nfqueue.so', nfqueue
info  nfqueue: installing iptables rules for device(s) lo
info  core:    loading rule file 'post.lua'
info  core:    initializing thread 0
info  dissector: register new dissector 'raw'
info  dissector: register new dissector 'ipv4'
info  dissector: register new dissector 'tcp'
info  dissector: register new dissector 'tcp_connection'
info  dissector: register new dissector 'http'
info  core:      1 rule(s) on event 'http:request_data'
info  core:      1 rule(s) on event 'http:request'
info  core:      1 rule(s) on event 'tcp_connection:new_connection'
info  core:      3 rule(s) registered

info  core:      initializing thread 1
info  dissector: register new dissector 'raw'
info  dissector: register new dissector 'ipv4'
info  dissector: register new dissector 'tcp'
info  dissector: register new dissector 'tcp_connection'
info  dissector: register new dissector 'http'
info  core:      1 rule(s) on event 'tcp_connection:new_connection'
info  core:      1 rule(s) on event 'http:request_data'
info  core:      1 rule(s) on event 'http:request'
info  core:      3 rule(s) registered

info  core:      starting multi-threaded processing on 2 threads

interactive rule:
inputs = table {
  1 : class http {
    flow : class tcp_connection
    state : class StateMachineInstance
  }
  2 : userdata vbuffer_sub
}

Hit ^D to end the interactive session
interactive mode>  inputs[2]:asstring()
  #1    "username=user&password=pass"
interactive mode>  data = haka.vbuffer_from("username=admin&password=qwerty")
interactive mode>  inputs[2]:replace(data)
interactive mode>  ^D
```

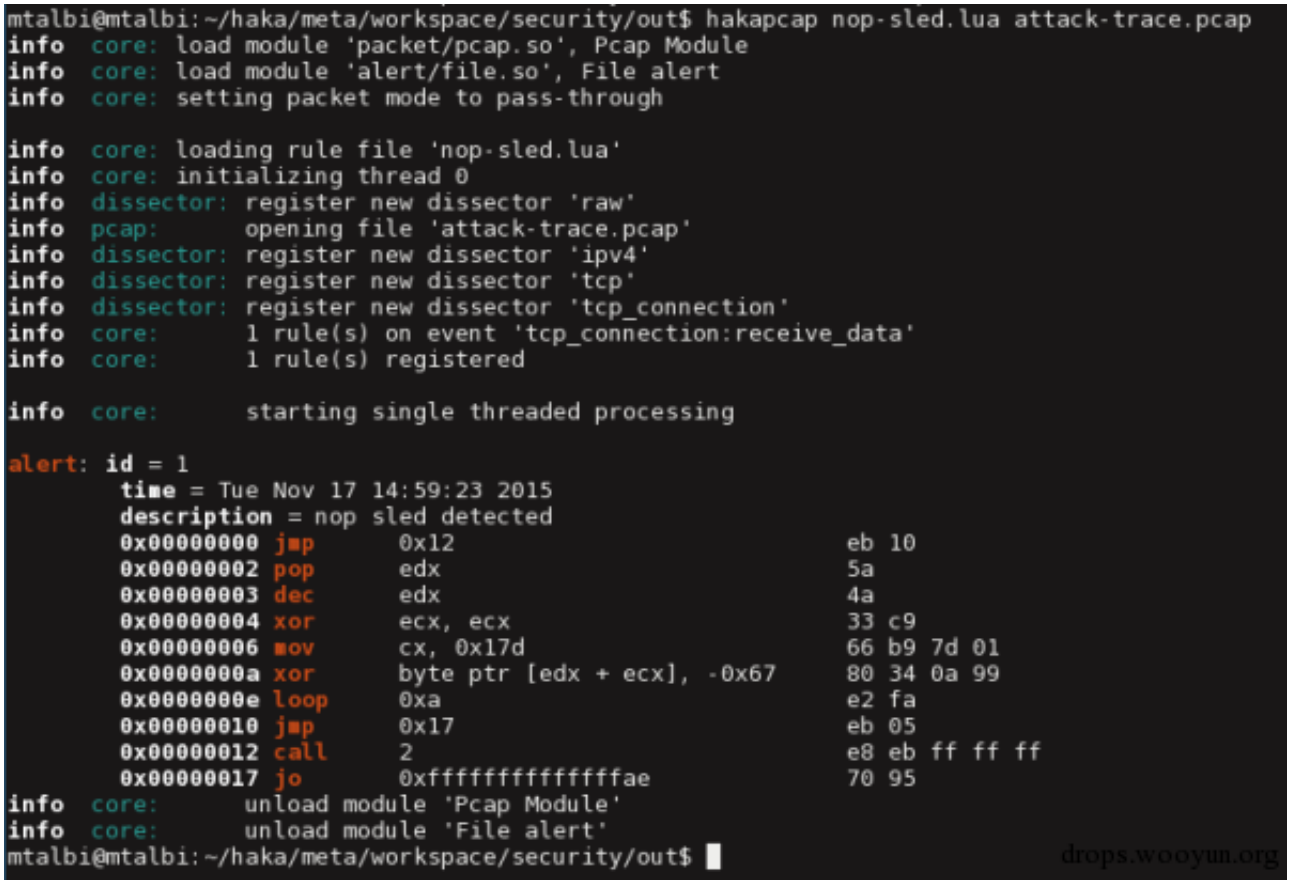

注意，最好在pcap文件上使用交互规则，因为修改会造成额外的延迟。

0x08 高级流过滤

Haka还有一个模式匹配引擎和一些反汇编模块。这两个模块都是基于流的模块，能让我们检测到分散在多个数据包中的恶意有效载荷。下面的规则使用了一个常规表达来检测nop sled。我们启用了流选项，也就是说，匹配函数会拦截和等待可用的数据来进行匹配。如果检测到nop sled，我们就会发出警告，并转储shellcode指令。注意，模式匹配函数会更新迭代器的位置，之后会指向shellcode。

```
local tcp = require("protocol/tcp_connection")

local rem = require("regexp/pcre")
local re = rem.re:compile("%x90{100,}")

local asm = require("misc/asm")
local dasm = asm.new_disassembler("x86", "32")

haka.rule{
    hook = tcp.events.receive_data,
        options = {
            streamed = true,
    },
    eval = function (flow, iter, direction)
        if re:match(iter, false) then
            -- raise an alert
            haka.alert{
                description = "nop sled detected",
            }
            -- dump instructions following nop sled
            dasm:dump_instructions(iter)
        end
    end
}
```

我们在著名的网络分析挑战上重播了这条规则，并得到了下面的输出。关于网络流量反汇编为指令的详细信息都可以在这里获得。



未完待续...

0x09 链接

- Haka 网站. haka-security.org (http://haka-security.org/)
- Haka 源代码. https://github.com/haka-security/haka (https://github.com/haka-security/haka)
- Haka twitter. @hakasecurity