IP Fragment Reassembly with scapy

带 scapy 的 IP 片段重组

GIAC (GCIA) Gold Certification

GIAC (GCIA)黄金认证

!"#$%&'()*&+(,* - .##/(0%1234415-6*7089%6 !:;7<%&'(=79+(>*??.&

！" #$%&'()*&+(，* -。##/(0%1234415-6*7089%6！:；7<%&'(=79+(>*？？。&

!99.@#.:'(A"?.1B#$(2412

！99.@#。:'(甲)? 。1B#$(2412

!C<#&*9#

！C<#&*9#

D;.&0*@@7?-(EF(G&*-6.?#<(9*?(C.("<.:(CH(*##*9+.&<(#%($7:.(#$.7&(?.G*&7%"<(7?#.?#7%?<(G&%6(7?#&"<7%?(:.#.9#7%?(<H<#.6(*?:(*?*0H<#<8(((D@.&*#7?-(<H<#.6<(-7;.@&.G.&.?9.(#%(%;.&0*@@7?-(G&*-6.?#<(C*<.:("@%?(.7#$.&(#$.@%<7#7%?(7?(#$.(@*9+.#(%&(#$.#76.(%G(*&&7;*08(((!<(*(&.<"0# G&*-6.?#.:(@*9+.#<(67-$#(C.(&.*<<.6C0.:(7?(%?.(%G G7;.(:7GG.&.?#(I*H<8((EG(#$.(EJK(%&(#$.(*?*0H<#(:%(?%#(&.*<<.6C0.(#$.(@*9+.#<(#$.(<*6.(I*H *<(#$.(#*&-.#($%<#/(*?(*##*9+(6*H(<"99..:(*?:(-%("?:.#.9#.:8(((>$70.(<%6.(7?#&"<7%?(:.#.9#7%?(<H<#.6<($*;.(#.9$?7L".<(G%&(:.*07?-(I7#$(#$.<.*##*9+<(#$.&.(*&.(;.&H G.I(#%%0<(*;*70*C0.(#%(#$.(*?*0H<#(#%(0%%+(7?<7:.(#$.(&.*<<.6C0H(@&%9.<<(*?:(#&H(#%("?:.&<#*?:(#$.(*##*9+.&M< 7?#.?#8((N$7<(@*@.&(I700(.O@0%&.($%I(*?(*?*0H<#(9*?("<.(<9*@H(#% &.*<<.6C0.(#$.(G&*-6.?#.:(*##*9+(@*9+.#<(7?(*(<7670*&(6*??.&(#% P7?"O/(>7?:%I</()*97?#%<$/(Q7<9%(&%"#.&< *?:(%#$.&(%@.&*#7?-(<H<#.6< #%(<..($%I(.*9$(%@.&*#7?-(<H<#.6(I%"0:(7?#.&@&.#(#$.(G&*-6.?#.:(@*9+.#<8(((

d .。&0*@@7? -(英孚(G&*-6)。? #<(9*? (c .(" <。:(CH(*##*9+)。&<(#%($7:。(#$.7&(? 。G*&7%"<(7? #。? #7%? <(G&%6(7? #&"<7%? (:。# 9 # 7%? (< H<#.6(*? :(*? *0H<#<8(((D)。&*#7? -(< H < # . 6 <(-7; 。@&.G.&。? 9.(# %(%; 。&0*@@7? -(G&*-6。? #<(C* <。:(" @%? (. 7#$。&(#$。@%<7#7%? (7? (#$。(@*9+。#(%&(#$)。#76。(% G(* & & 7; *08((! <((&)。< " 0# G&*-6。? #。:(@*9+。#<(67-$#(C.(&)。*<<.6C0.:(7? (%? 。(% G7; 。(:7GG。&。? #(I*H<8((例如($)。(EJK(%)&(# $)。(*? *0H<#(:%(? %#(&。* < < < . 6c 0 .(# $。(@*9+。#<(#$。(< *6。(工号*<(#$)。(#*&-。#($%<#/(*? (*##*9+(6*H(<"99..:(*? :(-% ("? :。#.9#。:8((> 70 美元。(< %6。(7? #&"<7%? (:。# 9 # 7%? (< H < # .6 <($ *; 。(#.9 美元? 7L”。<(G%&(:)。*07? -(I7#$(#$)。<。*##*9+<(#$。&。(*&。(; 。& H G . I(% % 0 <(*; *70*C0。(#%(#$)。(*? *0H<#(#%(0%%+(7? < 7:。(#$。(&。* < < < 6 C0H(@ & % 9。< <(*? :(# & H(%("? :。&<#*? :(#$。(*##*9+。&M< 7? #。? # 8($ 7 <(@ * @)。&(I700()。O@0%&。($ % 1(*? (*? *0H<#(9*? (" <。(< 9 * @ H(% &)。* < < < . 6c 0 .(# $。(G&*-6。? #。:(* # # # 9+(@ * 9+)。#<(7? (*(<7670*&(6*? ? 。&(#% P7? O/(>7? :%I</()*97? #%<$/(Q7<9%(& %"#)。&< *? :(%#$。&(%@。&*#7? -(<H<# ...($ % 1(.*9$(%@。&*#7? -(<H<# .#。&@&。#(#$。(G&*-6。? #。:(@*9+。#<8(()

Fragment Reassembly with Scapy 2

用 Scapy 2 重组片段

1.Introduction

1.介绍

1.1.The Problem

1.1 .问题是

Overlapping IP fragments can be used by attackers to hide their nefarious intentions from intrusion detection system and analysts.Packets fragmentation will be performed by a router when the size of a packet exceeds the link layers MTU of the upstream network.(Fall, Stevens, 2011) The receiving host is responsible for reassembling those fragmented packets and passing it up the TCP stack to the proper application.The RFC's are silent on the matter of what the receiving host is supposed to do when the fragments it receives are retransmitted or overlap one another.No guidance is given as to whether or not the host should favor the first "retransmitted" fragment it receives, the second "retransmitted" fragment or the last.Similarly, should it favor overlapping fragments with the lowest offset or the highest?As a result different operating systems handle overlapping fragments in different ways.This problem is illustrated by the paper "Active Mapping: Resisting NIDS Evasion Without Altering Traffic" by Umesh Shankar and Vern Paxson (Shankar & Paxson, 2003) and then further explained in "Target Based Fragmentation Assembly" by Judy Novak (Novak, 2005).

攻击者可以利用重叠的知识产权片段向入侵检测系统和分析师隐藏其邪恶意图。当数据包的大小超过上游网络的链路层 MTU 时，路由器将执行数据包分段。(史蒂文斯，2011 年秋季)接收主机负责重新组装这些碎片数据包，并将其向上传递到适当的应用程序。当接收到的片段被重传或相互重叠时，RFC 对接收主机应该做什么保持沉默。没有给出关于主机是否应该支持它接收的第一个“重传”片段、第二个“重传”片段还是最后一个片段的指导。同样，它应该倾向于偏移最小还是偏移最大的重叠片段？因此，不同的操作系统以不同的方式处理重叠的片段。这一问题在梅什·尚卡尔和弗恩·帕克森(尚卡尔和帕克森，2003 年)的论文“主动映射:在不改变流量的情况下抵制 NIDS 规避”中得到说明，然后在朱迪·诺瓦克(诺瓦克，2005 年)的“基于目标的碎片组装”中得到进一步解释。

Imagine that we send the following 6 IP fragments that overlap in the following ways.For the sake of this discussion each fragment is 8 bytes in length which is the minimum size of a fragment.Also, to help keep things straight we will set the payload to be an eight ASCII 1s for packet 1, 2s for packet 2 and so on.Fragment 1 has an offset of zero and has a payload length of 24 bytes so that that fills fragment positions 0, 1 (offset 8) and 2 (offset 16).Fragment 2 begins at offset 24 and has a length of 16 bytes so that it fills fragment positions 4 and 5.Fragment 3 has an offset of 48, length of 24 bytes and fills fragment positions 6, 7 and 8.Fragment 4 has an offset of 8, a length of 32 and fills fragment positions 1 (offset 8), 2 (offset 16), 3 (offset 24) and 4 (offset 32) causing it to

假设我们发送以下 6 个以下列方式重叠的 IP 片段。为了便于讨论，每个片段的长度为 8 字节，这是片段的最小大小。此外，为了帮助保持正常，我们将数据包 1 的有效载荷设置为 8 个 ASCII 1s，数据包 2 的有效载荷设置为 2，依此类推。片段 1 的偏移量为 0，有效载荷长度为 24 字节，从而填充片段位置 0、1(偏移量 8)和 2(偏移量 16)。片段 2 从偏移量 24 开始，长度为 16 字节，以便填充片段位置 4 和 5。片段 3 的偏移量为 48，长度为 24 字节，并填充片段位置 6、7 和 8。片段 4 的偏移量为 8，长度为 32，并填充片段位置 1(偏移量 8)、2(偏移量 16)、3(偏移量 24)和 4(偏移量 32)，使其

overlapping part of fragment positions 1 and 2.Fragment 5 has an offset of 48 and fills

片段位置 1 和 2 的重叠部分。片段 5 的偏移量为 48，并填充

positions 6, 7 and 8 so that it perfectly overlaps fragment 3.Fragment 6 has an offset of

位置 6、7 和 8，使得它完全重叠片段 3。片段 6 的偏移量为

72 and fills fragment positions 9, 0xa and 0xb.Visually it would look like this:

72 并填充片段位置 9、0xa 和 0xb。从视觉上看，应该是这样的:

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

Fragment Reassembly with Scapy 3

用 Scapy 3 重组片段

Figure 1: 6 Fragmented Packets (Shankar & Paxson, 2003)(Novak, 2005) Depending upon whether the reassembling host wants to favor the packets that arrive first or last or favor the packets with the lowest offset the fragments may end up in one of the 5 possible combinations.These combinations have been named First, Last, Linux, BSD and BSDRight.Reassembled using policy: First (Windows, SUN, MacOS, HPUX) Reassembled using policy: Last/RFC791 (Cisco) Reassembled using policy: Linux (Linux) Reassembled using policy: BSD (AIX, FreeBSD, HPUX, VMS) Reassembled using policy: BSD-Right (HP Jet Direct) Figure 2: 5 Reassembly Methods (Shankar & Paxson, 2003)(Novak, 2005) These inconsistencies allow attackers to put a malicious payload in an overlapped fragment.If the IDS and the host reassemble the packets differently the IDS will not see the packets, but the reassembling host will.Although many IDS's attempt to mitigate this risk by reassembling the packets in multiple ways, such as SNORT's frag3

2012 ansinstituteauthorRetainsfullrights。图 1: 6 分段数据包(Shankar & Paxson，2003)(Novak，2005)根据重组主机是希望优先还是最后到达的数据包，还是希望偏移最小的数据包，分段可能最终会出现在 5 种可能的组合中的一种。这些组合被命名为第一、最后、Linux、BSD 和 BSDRight。使用策略重新组装:首先(视窗、太阳、苹果机、高性能操作系统)使用策略重新组装:最后/射频 791(思科)使用策略重新组装:Linux (Linux)使用策略重新组装:BSD (AIX、免费软件、高性能操作系统、虚拟机)使用策略重新组装:BSD-Right(惠普 Jet Direct)图 2: 5 重新组装方法(尚卡尔和帕克森，2003)(诺瓦克，2005)这些不一致允许攻击者在重叠片段中放置恶意负载。如果入侵检测系统和主机重组数据包的方式不同，入侵检测系统将看不到数据包，但重组主机会看到。尽管许多入侵检测系统试图通过多种方式重组数据包来降低这一风险，例如 SNORT 的 frag3

preprocessor, the analyst is given very little insight to what happens inside the reassembly

预处理器，分析员对重组过程中发生的情况了解甚少

engine.This can lead to the analyst incorrectly dismissing an attack as an IDS false

引擎。这可能导致分析师错误地将攻击视为入侵检测错误

negative.Consider the following scenario.The attacker sends a crafted packet that

否定的。考虑以下场景。攻击者发送一个精心制作的数据包，该数据包

contains both a Linux Exploit and a Windows Exploit to a vulnerable Windows target.

包含针对易受攻击的视窗目标的 Linux 漏洞和视窗漏洞。

The IDS has an intelligent reassembly engine and successfully detect that exploitation of

入侵检测系统具有智能重组引擎，并成功检测到

the host.The Analyst then sees the alert and examines the full packet capture where he

主人。分析师随后会看到警报，并检查完整的数据包捕获

finds a Linux exploit targeting a Windows host and incorrectly dismisses it as a false

发现一个针对视窗主机的 Linux 漏洞，并错误地将其视为错误

positive.

积极。

sets and find

设置和查找

To avoid this situation an a

为了避免这种情况

or create new tools to properly analyze these

或者创建新的工具来正确分析它们

nalyst has to be aware of the limitations of his tool

分析学家必须意识到他的工具的局限性

lrights.attacks.

艾瑞莱特。攻击。

Authorretainsful !"#$%&%' ("")*+#,%-,)."/% 01234%)25%612578/% 94$:71"%.,);<#2"/% "),;#"12;%)%612578/% =7/" !"#$%>%' ?@!%*7,,#*":A%)//#<B:#/% $)*+#"/%)/%"=# %"),;#"%=7/"% 873:5%)25%):#,"/%"=)"%"=#%)"")*+%=)/%7**3,,#5 !"#$%C%' (2):A/"%94)<12#/% "=#%.3::%$)*+#"%*)$"3,#D% /##/%)%01234%#4$:71"% "),;#"12;%612578/%)25% 51/<1//#/%"=#%.):/#% $7/1"1E# !"#$%&'()*+,%"- ."#/*(()*+,%"- !"#$ %&'()*+,%"- ."#/*(()*+,%"- (FF(-G9H%I?96 ?@!%%I?96 (J(0K!F%I?96 612578/%72%612578/L % /3**#//.3:%)"")*+%(09HFMMM 01234%72%612578/% .)1:#5%)"")*+L%%!"3$15% ?@!L% %J#4"%$)*+#"MM

作者保留！" #$%&%' ("")*+#，%-，)。"/% 01234%)25%612578/% 94$:71"%。)；< #2"/% "，；# " 12；%)%612578/% =7/"！ " #$%>% '? @！ %*7、，# *:A %)//# < B:#/% $)*+# "/%)/% " = # % "，；#"%=7/"% 873:5%)25%):#，"/% " =)" = # %)" ")*+% =)/% 7 * * 3，，#5！ " #$%C%' (2):A/"%94)<12#/% "=#% .# " 12；%612578/%)25% 51/<1//#/%"=#%。):/#% $7/1"1E#！ "#$ %&'()*+，%"-，"#/*(()*+，%"-！ "#$%&'()*+，%"-，"#/*((()*+，% "-(FF(-G9H % 1? 96? @！ % % 1? 96(焦耳(0K！ F%I? 96 612578/% 72% 612578/L %/3 * * #//. 3:%)*+%(09HFMMM 01234% 72% 612578/%。)1:#5%)"")*+L%%！ "3 美元 15%? @！ L%%J#4"%$)*+# "毫米

20SANS

20SANS

Figure 3: Views of the attacker, IDS and analyst

图 3:攻击者、入侵检测系统和分析师的视图

InstituteFor example, if an analyst uses Wireshark to extract the payload of fragmented packets he will see neither the Linux payload NOR the Windows payload.Instead he will see a combination of the two.Wireshark uses the BSD reassembly policy when

例如，如果分析师使用 Wireshark 来提取分段数据包的有效负载，他既看不到 Linux 有效负载，也看不到视窗有效负载。相反，他会看到两者的结合。Wireshark 在以下情况下使用 BSD 重组策略

putting fragments back together.So, how then does an analyst know exactly what was launched against their system?

把碎片放回一起。那么，分析师如何确切知道针对他们的系统启动了什么?

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

2012SANSInstituteAuthorretainsfullrights.Figure 4: Wireshark uses BSD reassembly technique 1.2.One Possible Solution Tools such as scapy and Python can be used to quickly reassemble packets in each of the differently combinations used by modern operating systems to get a better understanding of what the attacker may have intended to exploit.Over the next few pages we will examine how to recreate the reassembly engines as they are implemented by various operating systems.Then we as analysts, can use these techniques to peer behind the curtain and see how our reassembly engines would see that attackers packets.Understanding the techniques being used by the attacker will give us greater insight to the skill of our adversary and perhaps even help us identify attacks that our automated reassembly engines might overlook.(Shankar & Paxson, 2003)(Novak, 2005) 2.Writing a fragment reassembly engine Writing the IP fragmentation engines in our TCP stack is no easy task.

2012 ansinstituteauthorRetainsfullrights。图 4: Wireshark 使用 BSD 重组技术 1.2。一种可能的解决方案工具，例如 scapy 和 Python，可以用来快速重组现代操作系统使用的每种不同组合中的数据包，以便更好地理解攻击者可能想要利用的内容。在接下来的几页中，我们将研究如何重新创建重组引擎，因为它们是由各种操作系统实现的。然后，作为分析师，我们可以使用这些技术在幕后窥视，看看我们的重组引擎会如何发现攻击者的数据包。了解攻击者使用的技术将使我们更深入地了解对手的技能，甚至可能帮助我们识别自动化重组引擎可能忽略的攻击。(尚卡尔和帕克森，2003 年)(诺瓦克，2005 年)2。编写碎片重组引擎在我们的 TCP 堆栈中编写 IP 碎片引擎不是一件容易的事情。

Fortunately for us, we do not have to deal with many of the difficulties the authors of

对我们来说幸运的是，我们不必处理许多作者提出的困难

those programs do.We don't have to worry about reassembly timeout, TTLs, memory

那些程序有。我们不必担心重组超时、TTLs、内存

management and other issues associated with the live transmission of data.We are

与数据实时传输相关的管理和其他问题。我们是

Fragment Reassembly with Scapy 6

用 Scapy 6 重组片段

reading our packets from stored packet captures.This gives us another advantage over

从存储的数据包捕获中读取我们的数据包。这给了我们另一个优势

reassembling live packets in that we have all of the packets in our possession and can

重组实时数据包，因为我们拥有所有数据包，并且可以

reorder them as needed before processing them.To reassemble the packets we will

在处理它们之前，根据需要对它们重新排序。为了重组数据包，我们将

allocate a buffer in memory and then write each fragment to t fragments to overwrite existing data

在内存中分配一个缓冲区，然后将每个片段写入测试片段以覆盖现有数据

.Using this method

。使用这种方法

,

,

the last fragments that we

我们最后的碎片

ta

谢谢

he buffer based

基于缓冲区的

insfull

insfull

allowing

允许

rights.process will overwrite any existing data in the buffer.To reassemble packets as each of the different reassembly policies we will just have to reorder our packet before we

权利。进程将覆盖缓冲区中的任何现有数据。要将数据包重组为不同的重组策略，我们只需在

process them.For example, to process packets according to the "LAST/RFC791" policy we would just process packets from the first one we received until the last in chronological order.Since subsequent overlapping fragments will overwrite the previous packets we are favoring the LAST packet to arrive.To process packets according to the "FIRST" policy we process the same packets in reverse order filling the buffer with the last packet to arrive, then the 2nd to last etc.The first packet to arrive will overwrite any data that was written in the buffer by later packets thus favoring the FIRST packets.

处理它们。例如，要根据“最后/射频 791”策略处理数据包，我们只需按时间顺序处理从收到的第一个数据包到最后一个数据包。由于后续的重叠片段将覆盖先前的数据包，我们倾向于最后一个到达的数据包。为了根据“第一”策略处理数据包，我们以相反的顺序处理相同的数据包，用最后一个到达的数据包填充缓冲区，然后是第二个到达的数据包，等等。第一个到达的数据包将覆盖由后面的数据包写入缓冲区的任何数据，从而有利于第一个数据包。

2.1.Python and scapy data structures

2.1 .Python 和 scapy 数据结构

Python's StringIO module provides us with a good data structure to use as our buffer for the reassembled fragments.We can use StringIO's seek() method to set the location in the buffer to the fragments offset.Then we use the write() method to put our data in the buffer.After we have processed all the fragments we can use the getvalue() method to retrieve the contents of our completed fragment payload.

Python 的 StringIO 模块为我们提供了一个良好的数据结构，作为我们重新组装片段的缓冲区。我们可以使用 StringIO 的 seek()方法将缓冲区中的位置设置为片段偏移量。然后我们使用 write()方法将数据放入缓冲区。处理完所有片段后，我们可以使用 getvalue()方法来检索已完成片段有效负载的内容。

Scapy allows you to quicky and easily tear apart packets and get to the fields you are interested in.By following a variable containing a packet with "[protocol]" and ".field" you can pull the contents of various fields from each packet.For example, to examine the IP ID field of a given packet we would simply address the variable containing the packet as variablename[IP].id. This tells scapy you want the value

Scapy 允许您快速轻松地拆开数据包，并到达您感兴趣的领域。通过跟随包含具有“[协议”的分组的变量“和”。字段”您可以从每个数据包中提取各个字段的内容。例如，为了检查给定数据包的 IP 标识字段，我们只需将包含该数据包的变量作为变量名称“[IP”]。这就告诉了我们您想要该值的原因

assigned to the 'id' field in the 'IP' layer of the packet.The field we are interested in is

分配给数据包“IP”层的“id”字段。我们感兴趣的领域是

the [IP].frag which contains the fragment offset of the current fragment and the payload

[知识产权]。片段，包含当前片段和有效载荷的片段偏移量

of each of the fragmented packets.

每一个分段的分组。

The fragment offset will be the number of bytes into

片段偏移量将是进入的字节数

fragment chain that the payload bytes should be written.(Kozierok, 2005)

有效负载字节应该写入的片段链。(Kozierok，2005 年)

The scapy

替罪羊

frag field is a fragment position not the byte offset.To get the byte offset you need to

frag 字段是一个片段位置，而不是字节偏移量。要获得字节偏移量，您需要

multiplying that number by 8 (8 bytes in the smallest fragment).Then using the

将该数字乘以 8(最小片段中有 8 个字节)。然后使用

StringIO.seek() method we place the pointer into the buffer at the location where the

方法，我们将指针放入缓冲区中

20SANS

20SANS

payload should be written.Using a FOR loop to step through each packet we simple reassembly engine.

有效载荷应该被写入。使用 FOR 循环遍历每个数据包，我们简单重组引擎。

have a

有

ights.2.2.The "Last/RFC791" policy Let's look at the simplest reassembly policies "Last/RFC791".This reassembly

ights。2.2 .“最后/射频 791”策略让我们看看最简单的重组策略“最后/射频 791”。这次重组

policy gives preference to fragment that appear later in a packet capture.Assume we have a list of all of the fragments that need to be reassembled.By processing the list of packets from the first to the last allowing the later to overwrite the earlier we follow the "Last" policy.When combined with scapy's ability to easily parse packets and extract fields like the fragment offset and payload we can write a very basic packet reassembly engine in just a few lines of Python code.The following code will take a list of fragments and assemble the payload according to the Last/RFC791 policy.

策略优先考虑数据包捕获中稍后出现片段。假设我们有一个需要重组的所有片段的列表。通过从第一个到最后一个处理数据包列表，允许后一个覆盖前一个，我们遵循“最后一个”策略。结合 scapy 轻松解析数据包和提取片段偏移量和有效负载等字段的能力，我们可以用几行 Python 代码编写一个非常基本的数据包重组引擎。下面的代码将根据 Last/RFC791 策略获取片段列表并组装有效负载。

def rfc791(listoffragments):

def rfc791(listoffragments):

buffer=StringIO.StringIO() for pkt in listoffragments:

buffer=StringIO。列表框中 pkt 的 StringIO()。

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue()

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效负载)返回缓冲区. getvalue()

Let's look at this code line by line.The first line uses the keyword "def" to define a new function called "rfc791" which will be passed a single parameter.The parameter will be stored in a variable called "listoffragments".As you might guess from the name the parameter will be a Python data structure called a list, and it will contain all of the fragments in a given fragment train.Notice that after the first line we begin indenting the code by 4 spaces.The indention is very important to Python.It tells Python that each of those indented lines is part of the "code block" that makes up the "rfc791" function we are defining.The second line will create a variable in memory

让我们一行一行地看这段代码。第一行使用关键字“def”来定义一个名为“rfc791”的新函数，该函数将被传递一个参数。该参数将存储在一个名为“listoffragments”的变量中。您可能从名称中猜到，该参数将是一个名为列表的 Python 数据结构，它将包含给定片段序列中的所有片段。请注意，在第一行之后，我们开始将代码缩进 4 个空格。缩进对 Python 非常重要。它告诉 Python，每一行缩进都是组成我们定义的“rfc791”函数的“代码块”的一部分。第二行将在内存中创建一个变量

called "buffer" which is of type StringIO.The variable "buffer" will be used to store all

称为“缓冲区”，属于 StringIO 类型。变量“缓冲区”将用于存储所有

of the pieces of the fragment train.Next we start a "FOR" loop to step through each

碎片列车的碎片。接下来，我们开始一个“FOR”循环来遍历每一个

individual fragment inside of the fragment train.The for loop is followed by another

碎片序列中的单个碎片。for 循环后面跟着另一个循环

group of indented lines.Again, the indention is used to group lines of code into a "code

一组缩进行。同样，缩进被用来将代码行分组为“代码”

block".The two lines that follow the for loop wi

阻止”。for 循环 wi 后面的两行

ll be executed repeatedly as part of the

将作为

for loop for each individual fragment in our list "listoffragments".The first time

对于我们列表“列表片段”中每个片段的循环。第一次

through the loop the variable "pkt" will contain the first fragmented packet in

通过循环，变量“pkt”将包含

20SANS

20SANS

“

"

fragmented packet in

碎片包在

listoffragment".The second time through the

listoffragment "。第二次通过

In

在

"listoffragments

“名单

stitute

研究所

"

"

A

A

.This will repeat for every packet in

。这将对中的每个数据包重复

uth

脐带缆终端接头

loop it will contain the second

循环它将包含第二个

orretainsfullrights."listofpackets".So, for every fragment in "listoffragments" we will execute these next two lines.The first one, "buffer.seek(pkt[IP].frag*8)" sets the pointer that will be used to write data in the buffer to the value that is contained in the scapy fragment position field of the current packet multiplied by eight.To convert a scapy fragment position number we multiply by 8 because each of these fragments will contain 8 bytes (64 bits).(Kozierok, 2005) Now that the pointer is set, the next line will write the payload of the fragment into the buffer at the location that was just set by the seek method.Once we have done that for all of the fragments we simply retrieve the contents of the buffer with the getvalue() method and return that from our function.(Python Software Foundation,

orretainsfullrights。“包裹列表”。因此，对于“列表片段”中的每个片段，我们将执行下面两行。第一个是“缓冲搜索”。frag*8)”将用于将缓冲区中的数据写入的指针设置为当前数据包的 scapy 片段位置字段中包含的值乘以 8。为了转换一个 scapy 片段的位置号，我们乘以 8，因为每个片段都包含 8 个字节(64 位)。(Kozierok，2005)现在指针设置好了，下一行将把片段的有效载荷写入缓冲区中刚才由 seek 方法设置的位置。一旦我们对所有片段都这样做了，我们只需用 getvalue()方法检索缓冲区的内容，然后从我们的函数中返回。(蟒蛇软件基金会，

2012)

2012 年)

2.3.The "First" policy

2.3 .“第一”政策

To write the FIRST reassembly engine we can follow the exact same process we followed to favor the LAST packet, but process our packets in reverse order.In doing so the first shall be last and our packets will be assembled properly.Python lists make it very easy to process a list in reverse order.By simply adding "[::1]" to the end of our list of fragments we reverse the list.(Lutz, 2012) Now writing our "FIRST" reassembly engine is almost identical to rfc791.

为了编写第一个重组引擎，我们可以遵循与支持最后一个包完全相同的过程，但是以相反的顺序处理我们的包。这样，第一个将是最后一个，我们的包将被正确地组装。Python 列表使得以相反的顺序处理列表变得非常容易。通过简单地在片段列表的末尾加上“[::1”，我们颠倒了列表。(卢茨，2012)现在编写我们的“第一”重组引擎几乎与 rfc791 相同。

def first(listoffragments):

def 优先(列表):

buffer=StringIO.StringIO() for pkt in listoffragments[::1]:

buffer=StringIO。[::1 列表中 pkt 的字符串()。

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue()

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效负载)返回缓冲区. getvalue()

2.4.

2.4 .

The "BSDRight" policy

“公平竞争”政策

Our remaining 3 reassembly policies look at more than just the chronological

我们剩下的 3 个重组策略不仅仅着眼于时间顺序

order the fragments arrived in.They also take the fragment offset into consideration

订购到达的碎片。他们还考虑了片段偏移

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

when deciding which fragment takes precedent.We need to reorder the packets so we

当决定哪个片段成为先例时。我们需要重新排序数据包，所以我们

process them based upon both the time they arrived and their offset according to the

根据它们到达的时间和它们的偏移量来处理它们

different reassembly engines.For the BSDRight policy we need to process fragments in

不同的重组引擎。对于 BSDRight 策略，我们需要在

20SANS

order by their fragment offset from lowest to highest.If two packets have the same offset then we allow the last one

按片段偏移量从低到高排序。如果两个包具有相同的偏移量，那么我们允许最后一个

Institut

协会

to arrive chronologically

按时间顺序到达

e, Authorre

e，Authorre

to overwrite the existing data.

覆盖现有数据。

tainsfullrights.Since our fragments are already in chronological order, sorting the packets based on their fragment offset will line the packets up for the BSD policy.We can use the sorted() function to put the fragments in to order by fragment offset then by chronological order.We pass the sorted function two parameters.We will pass it the list we want to sort and a "key" function to sorted() and it returns a list that is sorted based on the key.In this case our key function is "lambda x:x[IP].frag" which tells sorted() to put them in fragment offset order.

tainsfullrights。由于我们的片段已经按时间顺序排列，根据它们的片段偏移量对数据包进行排序将使数据包符合 BSD 策略。我们可以使用 sorted()函数按片段偏移量然后按时间顺序排列片段。我们传递排序函数的两个参数。我们将把我们想要排序的列表和一个“key”函数传递给 sorted()，它将返回一个基于 key 排序的列表。在这种情况下，我们的关键功能是“λx:x[知识产权”。frag ",它告诉 sorted()将它们按片段偏移顺序排列。

def bsdright(listoffragments):

def bsdright(列表选项):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(listoffragments, key=lambda x:x[IP].frag): buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload)

对于排序后的 pkt(列表项，关键字=λx:x[知识产权)。缓冲区。搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue()

返回 buffer.getvalue()

2.5.The "BSD" policy

2.5。“平衡计分卡”政策

BSD is simply BSDRight in reverse.Processing the BSDRight sorted fragments from last to first will cause the early fragments to overwrite the latter ones.We can take the same approach we used with FIRST and process the packets backwards by adding a '[::1]' to the end of our list of fragments.Because we want to process them in reverse order after they have been sorted, we add the [::1] to the end of the sorted function.Now we will processing the packet the same way we did for BSDRight but in reverse.

BSD 就是反向的 BSDRight。从最后到最先处理 BSDRight 排序的片段将导致早期片段覆盖后一个片段。我们可以采用与 FIRST 相同的方法，通过在片段列表的末尾添加“[::1”来向后处理数据包。因为我们希望在排序后以相反的顺序处理它们，所以我们将[::1]添加到排序函数的末尾。现在，我们将像处理 BSDRight 一样处理数据包，但方向相反。

def bsd(listoffragments):

def bsd(列表分割):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(listoffragments,key=lambda \ x:x[IP].frag)[::1]:

对于排序后的 pkt(列表项，关键字=λ\ x:x[知识产权)。frag)[::1]:

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue()

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效负载)返回缓冲区. getvalue()

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

2.6.

2.6 .

The "Linux" policy

“Linux”策略

The Linux Policy also takes the fragment offset into consideration.It favors

Linux 策略也考虑了片段偏移。它喜欢

whatever packet has the lowest offset.By processing the packets in reverse fragment offset order we allow the lowest fragments to overwrite the highest.So we need to sort our packets with the highest fragment offset appearing first in the list.To reverse a sort is as simple as passing the parameter "reverse=True" to our sorted command.By applying a reverse sort to our fragments before processing them first to last we get a Linux reassembly policy.

任何具有最低偏移的数据包。通过以反向片段偏移顺序处理数据包，我们允许最低的片段覆盖最高的片段。因此，我们需要对列表中最先出现的片段偏移量最高的数据包进行排序。反转排序就像将参数“reverse=True”传递给我们的排序命令一样简单。通过在处理片段之前对片段进行反向排序，我们得到了一个 Linux 重组策略。

def linux(listoffragments):

def Linux(list off fragments):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(listoffragments, key= lambda x:x[IP].frag,\ reverse=True):

对于排序后的 pkt(列表项，关键字=λx:x[知识产权)。frag，\ reverse=True):

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue()

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效负载)返回缓冲区. getvalue()

2.7.Testing the code

2.7 .测试代码

To test the code we can reassemble various fragmented packets samples from internet.We can craft our own fragmented packets using tools such as fragroute and fragrouter.We can also craft our own packets using scapy.The following section of code will generate the six packet fragments outlined in the introduction with the offsets specified in the Shankar/Paxson and Novak papers.

为了测试代码，我们可以从互联网上重新组装各种碎片数据包样本。我们可以使用 fragroute 和 fragrouter 之类的工具来制作我们自己的分段数据包。我们也可以用 scapy 制作我们自己的包。下面的代码部分将生成导言中概述的六个数据包片段，偏移量在 Shankar/Paxon 和 Novak 文件中指定。

def genfragments():

def genfragments():

pkts=scapy.plist.PacketList()

pkts=scapy.plist.PacketList()

pkts.append(IP(flags="MF",frag=0)/("1"*24)) pkts.append(IP(flags="MF",frag=4)/("2"*16)) pkts.append(IP(flags="MF",frag=6)/("3"*24)) pkts.append(IP(flags="MF",frag=1)/("4"*32)) pkts.append(IP(flags="MF",frag=6)/("5"*24)) pkts.append(IP(frag=9)/("6"*24)) return pkts

附加(2"*16)) pkts .附加(IP(flags="MF "，frag = 0)/(" 1 " * 24 ")pkts .附加(IP(flags="MF "，frag=4)/("2"*16)) pkts .附加(IP(flags="MF "，frag=6)/("3"*24)) pkts .附加(IP(flags="MF "，frag=1)/("4"*32)) pkts .附加(IP(flags="MF "，frag=6)/("5 ")

Now we can pass that fragment train off to each of our functions to see how it "puts humpty back together".Passing the results of genfragments() to the first() function will generate our fragment test pattern, then reassemble the packets using the

现在我们可以把这个片断序列传递给我们的每一个功能，看看它是如何“把驼峰放回到一起”的。将genfragments()的结果传递给第一个()函数将生成我们的片段测试模式，然后使用

first policy.By calling each of the reassembly engines we can see if our results match

第一项政策。通过调用每个重组引擎，我们可以看到结果是否匹配

those outlines in paper.

纸上的轮廓。

print "Reassembled using policy: First" print first(genfragments())

打印“使用策略重新组装:第一个”先打印(genfragments())

print "Reassembled using policy: Last/RFC791" print rfc791(genfragments()) print "Reassembled using policy: Linux" print linux(genfragments()) print "Reassembled using policy: BSD" print bsd(genfragments())

打印“使用策略重新组装:最后/RFC791”打印 rfc791(genfragments())打印“使用策略重新组装:Linux”打印 linux(genfragments())打印“使用策略重新组装:BSD”打印 bsd(genfragments())

print "Reassembled using policy: BSDRight" print bsdright(genfragments())

打印“使用策略重新组装:BSDRight”打印 bsdright(genfragments())

Running our script we would get the following result and we can see that indeed our reassembled packets do match what is expected from each of the reassembly engines.

运行我们的脚本，我们会得到下面的结果，我们可以看到我们重组的数据包确实与每个重组引擎的预期相匹配。

Figure 5: Output of running a simple fragment generator and reassembly engine

图 5:运行简单片段生成器和重组引擎的输出

2.8.Extending the code

2.8 .扩展代码

With the basic reassembly engines completed we can turn our attention to making the code user friendly and useful.One possible use would be to place each of the reassembly routines into a modularize script so you can "import" it into your existing scapy sessions to reassemble payloads as needed.Another application of this code would be to add functionality that extracts fragmented packets from pcap files then reassembles them using each of the 5 different policies so the analyst can see how the

随着基本重组引擎的完成，我们可以把注意力转向让代码变得用户友好和有用。一种可能的用途是将每个重组例程放入模块化脚本中，以便您可以将它“导入”到现有的 scapy 会话中，根据需要重组有效负载。此代码的另一个应用是添加从 pcap 文件中提取分段数据包的功能，然后使用 5 种不同策略中的每一种重新组装它们，这样分析师就可以看到

packets would be interpreted by different operating systems.The extended application

数据包将由不同的操作系统解释。扩展应用程序

could also allow the analyst to write reconstructed packets to disk.Although the review

还可以允许分析师将重构的数据包写入磁盘。尽管审查

of that source code is beyond the scope of this paper, I have produced that tool for you

源代码的内容超出了本文的范围，我已经为您制作了这个工具

and provided it in the appendix of this paper.Let's look at how we can use that

并在本文附录中提供。让我们看看如何使用它

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

application to analyze fragmented packets generated by fragroute and scapy.The

分析 fragroute 和 scapy 生成的分段数据包的应用程序。这

extended version of this script will read the packets from disk, find all fragmented

该脚本的扩展版本将从磁盘读取数据包，找到所有碎片

packets, prompt the user and ask if they want to process a given fragment train and

数据包，提示用户并询问他们是否想要处理给定的片段序列

display the reassembled packet on the screen or write the payload to disk.is updated the latest source code will be available for download at the

在屏幕上显示重新组装的数据包或将有效负载写入磁盘。最新的源代码将在

address

地址

As this script

就像这个剧本

rights.http://baggettscripts.googlecode.com/svn/trunk/reassembler/ .The version that was used for testing in this paper is included in the Appendix.First let's take a look at the

权利。http://baggettscripts.googlecode.com/svn/trunk/reassembler/。本文中用于测试的版本包含在附录中。首先让我们来看看

options that are available to the program.We can see the options by passing "—help" as an option to the script.

程序可用的选项。我们可以通过将“帮助”作为脚本的选项来查看选项。

Figure 6 : Help for reassembler.py

图 6:重组的帮助

Here you can see we can pass it r to read a pcap from disk and process it.We can use w if we want it to write the payloads to disk instead of printing to the screen.Finally,p is specified if we want to specify the filename prefix to use when using the w option to create payload files on disk.There is also the d option which will generate a fragmented packet stream then decode it.The d option is used to quickly gain an understanding of what each of the fragmentation engines does.If we really want to test the application we need some fragmented packets to test with.For that we can use a tool like fragroute or fragrouter to generate our packets.

在这里，您可以看到我们可以通过 r 从磁盘读取 pcap 并进行处理。如果我们想让 w 把有效载荷写到磁盘上，而不是打印到屏幕上，我们可以使用 w。最后，如果我们想指定在使用 w 选项在磁盘上创建有效负载文件时要使用的文件名前缀，请指定 p。还有 d 选项，它将生成一个分段的数据包流，然后对其进行解码。d 选项用于快速了解每个碎片引擎的作用。如果我们真的想测试应用程序，我们需要一些碎片包来测试。为此，我们可以使用 fragroute 或 fragrouter 这样的工具来生成数据包。

Although several fragmentation combinations were tested, here is one example of how we can create fragmented packets to test the software.First we create a fragroute configuration file that tells fragroute how to break our packets down.Here is an

虽然测试了几种碎片组合，但下面是我们如何创建碎片包来测试软件的一个例子。首先，我们创建一个 fragroute 配置文件，告诉 fragroute 如何分解数据包。这里有一个

example of a fragroute configuration file.

fragroute 配置文件示例。

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

作者保留版权。

Fragment Reassembly with Scapy 13

用 Scapy 13 重组片段

20SANS

20SANS

Figure 7: Fragroute Configuration File

图 7:碎片路由配置文件

This configuration file tells fragroute to br

该配置文件告诉 fragroute 将 br

InstituteAutho

InstituteAutho

eak the data into 8 by

用…把数据分成 8 份

rretainsf

rretainsf

t

t

u

u

e fragments then

那么 e 片段

llrights.insert duplicate packets with random payloads.Then transmit the fragments in random order.Note that the "ip_chaff dup" creates duplicate overlapping fragments.There are no partially overlapping fragments.The fragments perfectly overlap other fragments.

llrights。插入带有随机有效载荷的重复数据包。然后以随机顺序传输片段。请注意，“ip _箔条 dup”会创建重复的重叠片段。没有部分重叠的片段。这些碎片与其他碎片完全重叠。

All of the engines will reassemble the payloads as they to packets 3 and 5 from our overlapped packet test pattern.Therefore, we expect the "FIRST" and "BSD" patterns to reassemble packet one way and the other engines to assemble them the other way.

所有的引擎都将有效载荷重组为来自重叠包测试模式的包 3 和包 5。因此，我们期望“第一”和“BSD”模式以一种方式重组数据包，而其他引擎以另一种方式组装它们。

Figure 8: Results of assembling fragroute generated packets

图 8:组装碎片路由生成的数据包的结果

As expected First and BSD reassemble the packets one way and the other engines see a totally different payload.In this case First and BSD see the real payload and the other engines all see random garbage that was created by fragroute.Now, let's test to see if our code that writes the payloads to disk with an optional prefix works properly.

正如预期的那样，First 和 BSD 单向重组数据包，而其他引擎看到的是完全不同的有效负载。在这种情况下，First 和 BSD 看到真实的有效负载，而其他引擎都看到 fragroute 创建的随机垃圾。现在，让我们测试一下，我们用可选前缀将有效负载写入磁盘的代码是否工作正常。

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

2012SANSInstituteAuthorretainsfullrights.Figure 9: Writing reassembled payloads to disk It works!!This should be suitable for extracting binary payloads such as shell code and other exploits which may not behave well when displayed to the screen.Now, we as analysts can get the same insight to fragmented packets that our IDS engine may already have and make better informed decisions about the threat posed by a given attack 3.Conclusions The use of overlapping IP fragments for IDS evasion has been around since the 20th century.While some IDSs reduce the risk of false negatives through various reassembly mechanisms, the IDS Analyst is often blind to these attacks and left to trust the technology is not overlooking the threat.However, with tools such as Python, scapy and a little elbow grease the analyst can see exactly what malicious activities are being launched against their organization.By making use of these tools and techniques analysts are less likely to incorrectly dismiss an IDS generated true positive as a false positive.

2012 ansinstituteauthorRetainsfullrights。图 9:将重组后的有效载荷写入磁盘，这是可行的！！这应该适用于提取二进制有效载荷，如外壳代码和其他在屏幕上显示时可能表现不佳的漏洞。现在，作为分析师，我们可以对我们的入侵检测引擎可能已经拥有的碎片数据包有相同的见解，并对给定攻击 3 所造成的威胁做出更明智的决策。结论自 20 世纪以来，重叠知识产权片段在入侵检测系统规避中的应用一直存在。虽然一些入侵检测系统通过各种重组机制降低了误报的风险，但入侵检测系统分析师通常对这些攻击视而不见，只能相信技术不会忽视威胁。然而，有了像 Python、scapy 这样的工具和一点小小的工具，

分析师就可以准确地看到针对他们组织的恶意活动。通过使用这些工具和技术，分析师不太可能错误地将生成的真阳性的入侵检测系统视为假阳性。

4.References

4.参考

Shankar, U., & Paxson, V. (2003).Active mapping: Resisting nids evasion without altering traffic.Retrieved

Shankar，u .，& Paxson，V. (2003 年)。主动映射:在不改变流量的情况下抵抗入侵防御系统的逃避。恢复

脐带缆终端接头

from

从

orretainsfullrights.http://www.icir.org/vern/papers/activemapoak03.pdf Novak, J. (2005, April).Targetbased fragmentation reassembly.Retrieved April 29, from http://www.snort.org/assets/165/target_based_frag.pdf Fall, K., & Stevens, W. R. (2011).TCP/IP illustrated .(2nd ed., Vol. 1, p. 148).Ann Arbor, Michigan: Pearson Education Inc. Lutz, M. (2012).Python pocket reference.(4th ed., p. 16).North Sebastopol, CA:

orretainsfullrights。http://www.icir.org/vern/papers/activemapoak03.pdf·诺瓦克(2005 年 4 月)。基于目标的碎片重组。检索于 4 月 29 日，来自英国 http://www.snort.org/assets/165/target_based_frag.pdf 法尔和斯蒂文斯(2011)。图示为 TCP/IP。(第二版。，第 1 卷，第 148 页)。密歇根安阿伯:皮尔森教育公司，麻省鲁兹(2012)。蟒蛇皮口袋参考。(第四版。第 16 页)。加利福尼亚州北塞瓦斯托波尔:

O'Reilly Media Inc.

奥赖利媒体公司

Seitz, J. (2009).Grey Hat Python.San Francisco, CA: No Starch Press.

塞茨，J. (2009 年)。灰帽蟒蛇。加利福尼亚州旧金山:没有淀粉印刷机。

Python Software Foundation (2012).Python Online Documentation.Retrieved from http://www.python.org/doc/

Python 软件基金会(2012)。Python 在线文档。从 http://www.python.org/doc/取回

Kozierok, C. (2005).The TCP Guide.(p. 374).San Francisco, CA: No Starch Press.

Kozierok，C. (2005 年)。TCP 指南。(临 374)。加利福尼亚州旧金山:没有淀粉印刷机。

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

Fragment Reassembly with Scapy 16

用 Scapy 16 重组片段

5.Appendix

5.附录

P*#.<#(<%"&9.(9%:.(*?:(<*6@0.(G&*-6.?#.:(@*9+.#<(*&.(*;*70*C0.(G%&(:%I?0%*:(*#(#$.(G%00%I7?-(R=P8((N$.(;.&<7%?(%G(9%:.("<.:(G%&(#.<#7?-(7?(#$7<(@*@.&(7<(7?90":.:(C.0%I8 $##@'SSC* - .##T<9&7@#<8-%%-0.9%:.89%6S<;?S#&"?+S&.*<<.6C0.&S Program 1 Simple fragment generator & reassembler

P*#。< #(<%"&9。(9%)。(*? :(< *6@0。(G&*-6。? #。:(@*9+。#<(*&。(*; *70*C0。(G % &(:% 1? 0%*:(*#(#$)。(G%00%I7? -(R=P8((N$)。(; 。&<7%? (%G(9%)。(" <。:(G%&(#)。< #7? -(7? (#$7<(@*@)。&(7<(7? 90 ":。:(C.0%I8 $##@'SSC* -。# # T9 & 7 @ @ # < 8-% %- 0.9% . . 89% 6S <; ? S#& "? +S&。*<<.6C0.&S 程序 1 简单片段生成器和重组器

from scapy.all import * import StringIO

从 scapy.all import * import StringIO

def rfc791(fragmentsin):

def rfc791(碎片):

buffer=StringIO.StringIO() for pkt in fragmentsin:

buffer=StringIO。StringIO()用于碎片中的 pkt:

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue() def first(fragmentsin):

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效负载)返回缓冲区。getvalue() def 优先(分段):

buffer=StringIO.StringIO() for pkt in fragmentsin[::1]:

buffer=StringIO。StringIO()用于[碎片中的 PKT::1]:

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue() def bsdright(fragmentsin):

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)返回缓冲区. getvalue() def bsdright(碎片信息):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(fragmentsin, key= lambda x:x[IP].frag): buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload)

对于已排序的 pkt(片段，关键字=λx:x[知识产权)。缓冲区。搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue() def bsd(fragmentsin):

返回 buffer.getvalue() def bsd(片段):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(fragmentsin, key=lambda x:x[IP].frag)[::1]: buffer.seek(pkt[IP].frag*8)
buffer.write(pkt[IP].payload)

对于已排序的 pkt(片段，关键字=λx:x[知识产权)。[::1]:缓冲区。搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue() def linux(fragmentsin):

返回 buffer.getvalue() def linux(片段):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(fragmentsin, key= lambda x:x[IP].frag): buffer.seek(pkt[IP].frag*8)
buffer.write(pkt[IP].payload)

对于已排序的 pkt(片段，关键字=λx:x[知识产权)。缓冲区。搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue() def genjudyfrags():

返回 buffer . getvalue()def genjudyfrags():

pkts=scapy.plist.PacketList()

pkts=scapy.plist.PacketList()

pkts.append(IP(flags="MF",frag=0)/("1"*24)) pkts.append(IP(flags="MF",frag=4)/("2"*16))
pkts.append(IP(flags="MF",frag=6)/("3"*24)) pkts.append(IP(flags="MF",frag=1)/("4"*32))
pkts.append(IP(flags="MF",frag=6)/("5"*24)) pkts.append(IP(frag=9)/("6"*24)) return pkts

附加(2"*16)) pkts .附加(IP(flags="MF "，frag = 0)/(" 1 " * 24 ")pkts .附加(IP(flags="MF "，frag=4)/("2"*16)) pkts .附加(IP(flags="MF "，frag=6)/("3"*24)) pkts .附加(IP(flags="MF "，frag=1)/("4"*32)) pkts .附加(IP(flags="MF "，frag=6)/("5 ")

print "Reassembling the following packets:"

打印“重新组装以下数据包:”

print "111111111111111111111111 2222222222222222333333333333333333333" print " 44444444444444444444444444444444
5555555555555555555555566666666666666666666666"

打印“111111111111111111112222222222333333333333333333333333”打印”44444444441111111111122222222223333333333333”3333”33333333333333333333”333333333”33333333333333333”3333333333”的“3333334)的

print "Reassembled using policy: First (Windows, SUN, MacOS, HPUX)" print first(genjudyfrags())

打印“使用策略重新组装:第一个(视窗、太阳、苹果操作系统、高性能操作系统)”打印第一个(genjudyfragment())

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

Fragment Reassembly with Scapy 17

用 Scapy 17 重组片段

print "Reassembled using policy: Last/RFC791 (Cisco)" print rfc791(genjudyfrags())

打印“使用策略重新组装:最后/RFC791(思科)”打印 rfc791(genjudyfrags())

print "Reassembled using policy: Linux (Umm..Linux)" print linux(genjudyfrags())

打印“使用策略重新组装:Linux(嗯..Linux)"打印 Linux(genjudyfrag())

print "Reassembled using policy: BSD (AIX, FreeBSD, HPUX, VMS" print bsd(genjudyfrags())

打印“使用策略重新组装:BSD (AIX，FreeBSD，HPUX，VMS)”打印 BSD(genjudyfrag())

print "Reassembled using policy: BSDRight (HP Jet Direct)" print bsdright(genjudyfrags())

打印“使用策略重新组装:直接打印(惠普喷墨打印机)”打印直接打印(通用打印机())

PROGRAM 2 Extending the code to reassemble fragments from disk

程序 2 扩展代码从磁盘重组片段

from scapy.all import * import StringIO

从 scapy.all import * import StringIO

from optparse import OptionParser import os import sys

从 optparse 导入选项分析器导入操作系统导入系统

def rfc791(fragmentsin):

def rfc791(碎片):

buffer=StringIO.StringIO() for pkt in fragmentsin:

buffer=StringIO。StringIO()用于碎片中的 pkt:

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue() def first(fragmentsin):

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效负载)返回缓冲区。getvalue() def 优先(分段):

buffer=StringIO.StringIO() for pkt in fragmentsin[::1]:

buffer=StringIO。StringIO()用于[碎片中的 PKT::1]:

buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload) return buffer.getvalue() def bsdright(fragmentsin):

缓冲搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)返回缓冲区. getvalue() def bsdright(碎片信息):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(fragmentsin, key= lambda x:x[IP].frag): buffer.seek(pkt[IP].frag*8) buffer.write(pkt[IP].payload)

对于已排序的 pkt(片段，关键字=λx:x[知识产权)。缓冲区。搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue() def bsd(fragmentsin):

返回 buffer.getvalue() def bsd(片段):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(fragmentsin, key=lambda x:x[IP].frag)[::1]: buffer.seek(pkt[IP].frag*8)
buffer.write(pkt[IP].payload)

对于已排序的 pkt(片段，关键字=λx:x[知识产权)。[::1]:缓冲区。搜索(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue()

返回 buffer.getvalue()

def linux(fragmentsin):

def linux(碎片):

buffer=StringIO.StringIO()

buffer=StringIO。StringIO()

for pkt in sorted(fragmentsin, key= lambda x:x[IP].frag, reverse=True): buffer.seek(pkt[IP].frag*8)
buffer.write(pkt[IP].payload)

对于已排序的 pkt(片段，关键字=λx:x[知识产权)。碎片，反向=真):缓冲。寻找(pkt 知识产权)。frag*8)缓冲区。写(pkt 知识产权)。有效载荷)

return buffer.getvalue() def genjudyfrags():

返回 buffer . getvalue()def genjudyfrags():

pkts=scapy.plist.PacketList()

pkts=scapy.plist.PacketList()

pkts.append(IP(flags="MF",frag=0)/("1"*24)) pkts.append(IP(flags="MF",frag=4)/("2"*16))
pkts.append(IP(flags="MF",frag=6)/("3"*24)) pkts.append(IP(flags="MF",frag=1)/("4"*32))
pkts.append(IP(flags="MF",frag=6)/("5"*24)) pkts.append(IP(frag=9)/("6"*24)) return pkts

附加(2"*16)) pkts .附加(IP(flags="MF "，frag = 0)/(" 1 " * 24 ")pkts .附加(IP(flags="MF "，frag=4)/("2"*16)) pkts .附加(IP(flags="MF "，frag=6)/("3"*24)) pkts .附加(IP(flags="MF "，frag=1)/("4"*32)) pkts .附加(IP(flags="MF "，frag=6)/("5 ")

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

Fragment Reassembly with Scapy 18

用 Scapy 18 重组片段

def processfrags(fragmenttrain):

def processfrags(fragmenttrain):

print "Reassembled using policy: First (Windows, SUN, MacOS, HPUX)" print first(fragmenttrain)

打印“使用策略重新组装:第一个(窗口、太阳、苹果操作系统、高性能操作系统)”打印第一个(碎片序列)

print "Reassembled using policy: Last/RFC791 (Cisco)" print rfc791(fragmenttrain)

打印“使用策略重新组装:最后/RFC791(思科)”打印 RFC791(碎片序列)

print "Reassembled using policy: Linux (Umm..Linux)" print linux(fragmenttrain)

打印“使用策略重新组装:Linux(嗯..linux)"打印 Linux(碎片训练)

print "Reassembled using policy: BSD (AIX, FreeBSD, HPUX, VMS)" print bsd(fragmenttrain)

打印“使用策略重新组装:BSD (AIX，FreeBSD，HPUX，VMS)”打印 BSD(碎片序列)

print "Reassembled using policy: BSDRight (HP Jet Direct)" print bsdright(fragmenttrain)

打印“使用策略重新组装:直接打印(惠普喷射直接)”打印直接打印(碎片序列)

def writefrags(fragmenttrain):

def writefrags(fragmenttrain):

fileobj=open(options.prefix+"first","w") fileobj.write(first(fragmenttrain)) fileobj.close()

file obj = open(options . prefix+" first "、" w ")file obj . write(first(fragmenttrain))file obj . close()

fileobj=open(options.prefix+"rfc791","w") fileobj.write(rfc791(fragmenttrain)) fileobj.close()

file obj = open(options . prefix+" RFC 791 "、" w ")file obj . write(RFC 791(fragmenttrain))file obj . close()

fileobj=open(options.prefix+"bsd","w") fileobj.write(bsd(fragmenttrain)) fileobj.close()

file obj = open(options . prefix+" BSD "、" w ")file obj . write(BSD(fragmenttrain))file obj . close()

fileobj=open(options.prefix+"bsdright","w") fileobj.write(bsdright(fragmenttrain)) fileobj.close()

file obj = open(options . prefix+" bsdright "、" w ")file obj . write(bsdright(fragmenttrain))file obj . close()

fileobj=open(options.prefix+"linux","w") fileobj.write(linux(fragmenttrain)) fileobj.close()

file obj = open(options . prefix+" Linux "、" w ")file obj . write(Linux(fragmenttrain))file obj . close()

def main():

def main():

parser=OptionParser(usage='%prog [OPTIONS]')

解析器=选项解析器(用法=“% Prog[选项]”)

parser.add_option('r','read',default="",help='Read the specified packet capture',dest='pcap')

parser.add_option('r '，' read '，default= " '，help='Read 指定的数据包捕获'，dest='pcap ')

parser.add_option('d','demo',action='store_true', help='Generate classic fragment test patter and reassemble it.')

parser.add_option('d '，' demo '，action='store_true '，help= '生成经典片段测试模式并重新组装它')

parser.add_option('w','write',action='store_true', help='Write 5 files to disk with the payloads.')

parser.add_option('w '，' write '，action='store_true '，help= '将 5 个文件与有效负载一起写入磁盘')

parser.add_option('p','prefix',default='reassembled', help='Specify the prefix for file names')

解析器. add_option('p '，'前缀'，默认值= '重组'，帮助= '为文件名指定前缀')

if (len(sys.argv)==1):

if (len(sys.argv)==1):

parser.print_help() sys.exit()

parser.print_help() sys.exit()

global options, args

全局选项，args

(options,args)=parser.parse_args() if options.demo:

(选项，参数)=parser.parse_args() if 选项. demo:

processfrags(genjudyfrags()) if not os.path.exists(options.pcap):

processfrags(genjudyfrags())如果不是 os.path.exists(options.pcap):

print "Packet capture file not found."sys.exit(2)

打印“找不到数据包捕获文件”系统退出(2)

packets=rdpcap(options.pcap)

packets=rdpcap(options.pcap)

filtered=[a for a in packets if a[IP].flags==1 or a[IP].frag > 0] if len(filtered)==0:

过滤=如果是[协议，则为数据包中的[协议。标志==1 或[知识产权]。frag > 0]如果 len(过滤)=0:

print "No fragments in packet capture file."sys.exit(2)

打印“数据包捕获文件中没有碎片”系统退出(2)

uniqipids={}

uniqipids={}

for a in filtered:

对于已过滤的中:

uniqipids[a[IP].id]='we are here'

[一[知识产权。id]=“我们在这里”

for ipid in uniqipids.keys():

对于 uniqipid 中的 ipid，key():

print "Packet fragments found.Collecting fragments now."fragmenttrain = [a for a in filtered if a[IP].id == ipid] processit = raw_input("Reassemble packets between hosts "+str(a[0][IP].src)+" and

打印”找到的数据包片段。现在收集碎片。”碎片序列= [a 代表过滤的[知识产权]。id == ipid] processit = raw_input(“重新组装主机之间的数据包”+字符串([0)[IP)。src)+"和

"+str(a[0][IP].dst)+"?[Y/N]")

"+字符串([0][知识产权]。dst)+"？[是/否]”)

)*&+(,* - .##/(0%1234415-6*7089%6

)*&+(，* -。##/(0%1234415-6*7089%6

作者保留版权。

if str(processit).lower()=="y": if options.write:

如果字符串(processit)。lower()=="y": if 选项。写:

writefrags(fragmenttrain) else:

writefrags(fragmenttrain)其他:

processfrags(fragmenttrain)

processfrags(fragmenttrain)

```
if __name__ == '__main__':
    main()
```

Upcoming Training

SANS Security Operations London 2019	London, United Kingdom	Dec 02, 2019 - Dec 07, 2019	Live Event
SANS Frankfurt December 2019	Frankfurt, Germany	Dec 09, 2019 - Dec 14, 2019	Live Event
SANS Cyber Defense Initiative 2019	Washington, DC	Dec 10, 2019 - Dec 17, 2019	Live Event
SANS vLive - SEC503: Intrusion Detection In-Depth	SEC503 - 202001,	Jan 13, 2020 - Mar 04, 2020	vLive
Mentor Session - SEC503	Falls Church, VA	Jan 22, 2020 - Mar 25, 2020	Mentor
SANS Security East 2020	New Orleans, LA	Feb 01, 2020 - Feb 08, 2020	Live Event
Security East 2020 - SEC503: Intrusion Detection In-Depth	New Orleans, LA	Feb 03, 2020 - Feb 08, 2020	vLive
SANS Jacksonville 2020	Jacksonville, FL	Feb 24, 2020 - Feb 29, 2020	Live Event
SANS Zurich February 2020	Zurich, Switzerland	Feb 24, 2020 - Feb 29, 2020	Live Event
Blue Team Summit & Training 2020	Louisville, KY	Mar 02, 2020 - Mar 09, 2020	Live Event
Dallas 2020 - SEC503: Intrusion Detection In-Depth	Dallas, TX	Mar 09, 2020 - Mar 14, 2020	vLive
SANS Dallas 2020	Dallas, TX	Mar 09, 2020 - Mar 14, 2020	Live Event
Community SANS Columbia SEC503 @UKI	Columbia, MD	Mar 09, 2020 - Mar 14, 2020	Community SANS
SANS Norfolk 2020	Norfolk, VA	Mar 16, 2020 - Mar 21, 2020	Live Event
SANS San Francisco Spring 2020	San Francisco, CA	Mar 16, 2020 - Mar 27, 2020	Live Event
SANS 2020	Orlando, FL	Apr 03, 2020 - Apr 10, 2020	Live Event
Mentor Session - SEC503	Houston, TX	Apr 04, 2020 - May 02, 2020	Mentor
SANS 2020 - SEC503: Intrusion Detection In-Depth	Orlando, FL	Apr 05, 2020 - Apr 10, 2020	vLive
SANS London April 2020	London, United Kingdom	Apr 20, 2020 - Apr 25, 2020	Live Event
SANS Baltimore Spring 2020	Baltimore, MD	Apr 27, 2020 - May 02, 2020	Live Event
SANS Security West 2020	San Diego, CA	May 06, 2020 - May 13, 2020	Live Event
SANS Amsterdam May 2020	Amsterdam, Netherlands	May 11, 2020 - May 18, 2020	Live Event
SANS San Antonio 2020	San Antonio, TX	May 17, 2020 - May 22, 2020	Live Event
SANS Paris June 2020	Paris, France	Jun 08, 2020 - Jun 13, 2020	Live Event
SANS Las Vegas Summer 2020	Las Vegas, NV	Jun 08, 2020 - Jun 13, 2020	Live Event
SANSFIRE 2020	Washington, DC	Jun 13, 2020 - Jun 20, 2020	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced