Packet Capture, Filtering and Analysis

Today's Challenges with 20 Years Old Issues

Alexandre Dulaunoy

alexandre.dulaunoy@circl.lu

January 20, 2012

Promiscuous mode

Where can we capture the network data ? a layered approach

- *A network card can work in two modes, in non-promiscuous mode or in promiscuous mode :*
  - *In non-promiscuous mode, the network card only accept the frame targeted with is own MAC or broadcasted.*
  - *In promiscuous mode, the network card accept all the frame from the wire. This permits to capture every packets.*

```
ifconfig eth0 promisc
```

- *Other approaches possible to capture data (Bridge interception, dup-to of a packet filtering, ...)*

A side note regarding wireless network, promiscuous mode is only capturing packet for the associated AP. You'll need the monitor mode, to get capturing everything without being associated to an AP or in ad-hoc mode.

BPF History

How to get the data from the data link layers ?

- *BPF (Berkeley Packet Filter) sits between link-level driver and the user space. BPF is protocol independant and use a filter-before-buffering approach. (NIT on SunOS is using the opposite approach).*
- *BPF includes a machine abstraction to make the filtering (quite) efficient.*
- *BPF was part of the BSD4.4 but libpcap provide a portable BPF for various operating systems.*
- *The main application using libpcap (BPF) is tcpdump. Alternative exists to libpcap from wiretap library or Fairly Fast Packet Filter.*

Network data capture is a key component of a honeynet design.

BPF - Filter Syntax

- How to filter specific host :

```
host myhostname
dst host myhostname
src host myhostname
```

- How to filter specific ports :

```
port 111
dst port 111
src port 111
```

BPF - Filter Syntax

- How to filter specific net :

```
net 192.168
dst net 192.168
src host 192.168
```

- How to filter protocols :

```
ip proto \tcp
ether proto \ip
```

BPF - Filter Syntax

- Combining expression :

```
&& -> concatenation
not -> negation
|| -> alternation (or)
```

- Offset notation :

```
ip[8] Go the byte location 8 when not specified
      check 1 byte
tcp[2:2] Go the byte location 2 and read 2 bytes
tcp[2:2] = 25 (similar to dst port 25)
Matching (detailed after) is also working tcp[30:4] = 0xDEAD
```

BPF - Filter Syntax

- Offset notation and matching notation (what's the diff?):

```
ip[22:2]=80
tcp[2:2]=80
ip[22:2]=0x80
tcp[2:2]=0x80
```

BPF - Filter Syntax

- Using masks to access "bits" expressed information like TCP flags:

```
+-+-+-+-+-+-+-+-+
|C|E|U|A|P|R|S|F|
|W|C|R|C|S|S|Y|I|
|R|E|G|K|H|T|N|N|
+-+-+-+-+-+-+-+-+
```

```
tcp[13] = 2 (only SYN -> 00000010)
tcp[13] = 18 (only SYN, ACK -> 00010010)
tcp[13]&4 = 4 (matching RST ->00000100&00000100)
```

BPF - Filter Syntax

- If you don't want to match every bits, you have some variations.
- Matching only some bits that are set :

  `tcp[12] &9 != 0`
- If you want to match the exact value without the mask :

  `tcp[12] = 1`

BPF - Filter Syntax

- Using masks to access "bits" expressed information like IP version:

```
+-+-+-+-+-+-+-+-+
|Version|  IHL  |
+-+-+-+-+-+-+-+-+
```

```
ip[0] & 0xf0 = 64
ip[0] & 0xf0 = 96
```

BPF - Filter Syntax on Payload

- Matching content with a bpf filter. bpf matching is only possible on 1,2 or 4 bytes. If you want to match larger segment, you'll need to combine filter with &&.
- An example, you want to match "GE" string in a TCP payload :

```
echo -n "GE" | hexdump -C
00000000  47 45      |GE|
sudo tcpdump -s0 -n -i ath0 "tcp[20:2] = 0x4745"
```

Libpcap dev - a very quick introduction

- How to open the link-layer device to get packet :

```
pcap_t *pcap_open_live(char *device, int snaplen,
                       int promisc, int to_ms,
               char *ebuf)
```

- How to use the BPF filtering :

```
int pcap_compile(pcap_t *p, struct bpf_program *fp,
                 char *str, int optimize,
         bpf_u_int32 netmask)
int pcap_setfilter(pcap_t *p,
                   struct bpf_program *fp)
```

Libpcap - a very quick introduction 2/2

- How to capture some packets :

```
u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)
```

- How to read the result (simplified) from the inlined structs :

```
sniff_ethernet addr
sniff_ip addr + SIZE_ETHERNET
sniff_tcp addr + SIZE_ETHERNET
               + {IP header length}
payload addr + SIZE_ETHERNET
             + {IP header length}
             + {TCP header length}
```

Libpcap libraries

You don't like C and you'll want to code quickly for the workshop... Here is a non-exhaustive list of libcap (and related) binding for other languages :

- *Net::Pcap - Perl binding*
- *rubypcap - Ruby binding with a nice OO interface*
- *pylibpcap, pypcap - Python bindings*
- *plokami - Common Lisp pcap binding*

Libpcap tools

- tcpdump, tcpslice
- ngrep (you can pass regex search instead of offset search)
- tshark, wireshark
- tcpdstat
- tcptrace
- ipsumdump (relying on click router library)
- tcpflow
- ssldump

Digging in real packet captures

Practical session will be the analysis of a packet capture in a pcap format.

- Where to start? Focus on little events? big events?
- Can I find the attacker? the kind of attack?
- You can use any of the tools proposed but...
- ... you can build your own tools to ease your work.
- Time reference is a critical part in forensic analysis.
- Be imaginative.

Common issues at capture level

- Appropriate snaplen size (tcpdump -s0?)
- Network card/driver performance (pps versus bit/s)
- Size of stored packet capture (streaming versus storing)
- The pre-filter dilemma
- Capture after attacks (and not before)

- Total size of packet capture session can be very large
  - Disk access versus memory access
  - A multitude of small or large files
  - pcap format and the lack of metadata (e.g. usually metadata is the filename)
- Noise versus "interesting" traffic
  - Network baseline doesn't usually exist before the incident
  - Noise→malicious traffic classification dilemma
- Protocol detection
  - port number $\neq$ protocol
  - Detection of covert channels

- Packet capture and analysis are performed by software and software is **prone to attack**
  - Don't underestimate the attackers to compromise or divert your network capture/analysis
  - Parser and dissector are a common place for software bugs and vulnerabilities
- Passive detection of your network capture/forensic tools
  - Attackers don't like to be trapped or monitored
  - Indirect detection like the DNS resolving are not unusual

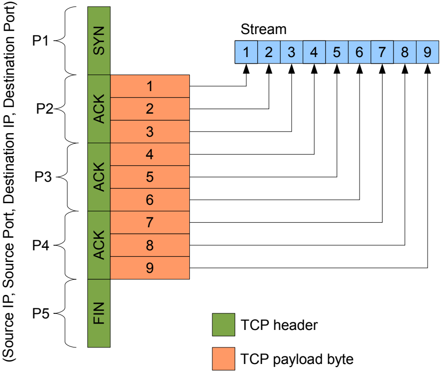
Attacking TCP reassembly

Definitions and terminology

- A PCAP file contains network packets
- Analyst is the person that is analyzing a PCAP file
- An attacker is the person that tries to lure the analyst
- A 4-tuple is (source IP, source port, destination IP, destination port)
- A TCP session
  - Starts with the TCP ESTABLISHED state
  - Ends with the TCP CLOSED state

Introduction

TCP reassembly



Related work

TCP reassembly is not new ... and some attacks still work ...

- TCP Reassembly Attacks for Network Intrusion Detection Systems
  - Tools
    - Fragrouter → NIDS benchmark
  - Attack countermeasures
    - Traffic Normalization → remove ambiguities
  - Reference
    - Nidsbench (1999) describes NIDS tests and attacks
    - SniffJoke (2011) *downgrade the sniffer technology from multi gigabits to multi kilobits*

Tools

Targeted tools

Tcpflow	Tcptrace
Wireshark	Tcpick

Used tools

Tcpdump	User Mode Linux	Fragrouter
Iptables	Socat	Nc

$\rightarrow$ Standard tools of network researchers and operators

Launching Valgrind on TCP reassembly tools

Error	Tcptrace	Tcpflow	Tcpick	
Invalid read s=4	5	0	0	occ.
Invalid read s=1	2	11	0	occ.
Definitely lost	345	0	16	bytes
Possibly lost	49152	0	0	bytes
Invalid fd	36196	0	0	occ.
Uninitialization	0	4	2	occ.

Attacking the TCP implementation

Definition

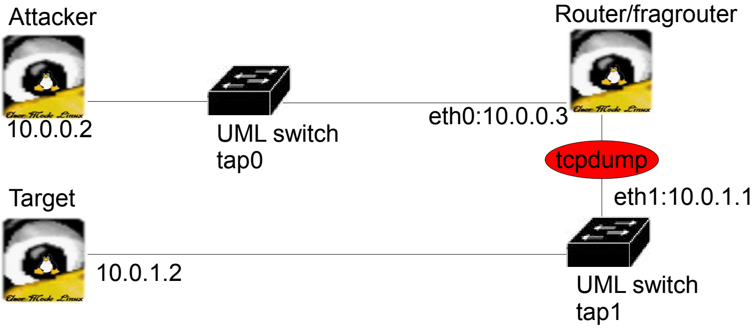
- Most of the forensics tools have their own TCP/IP implementation
- TCP/IP implementations are often incomplete or defective

Example

- IP fragmentation is not implemented
- The implementation is vulnerable to fragment attacks
- The TCP implementation does not completely respect the standard TCP state machine

Attacking the TCP implementation

Attacker setup



Note: All is software based on User Mode Linux

Attacking the TCP implementation

Constraints

- Attacker and target need to be on different subnets
  - Cause: Fragrouter eats ARP responses from the attacker
- On the router UML, `/proc/sys/net/ipv4/ip_forward` must be 0
  - Avoid race conditions between attacker TCP/IP stack and fragrouter
  - Routing is done by fragrouter (user space)

Attacking the TCP implementation

Methodology

- At the router UML
  - Launch fragrouter with an attack on eth0
  - Launch fragrouter with IP forwarding on eth1 → return packets
  - tcpdump -n -s0 -w packets.cap
- At the target UML
  - nc -l -p 2000 > receive.dat
- At the attacker UML
  - cat data.dat | nc target 2000
- Was the attack successful? → diff data.dat receive.dat
- Launch reassembly tool on packets.cap :-)

Attacking the TCP implementation

Fragrouter attacks

- Attacks are named after the command line switches
- Check capture process → B1 is regular IP forwarding
- Ordered 16-byte fragments, fwd-overwriting → F7
- 3-whs, bad TCP checksum FIN/RST, ordered 1-byte segments → T1
- 3-whs, ordered 2-byte segments, fwd-overwriting → T5

Attacking the TCP implementation

Results

Attack	Tcpflow	Wireshark	Tcptrace	Tcpick
B1	$\surd$	$\surd$	$\surd$	$\surd$
T1	$\times$	$\times$	$\times$	$\times$
T5	$\times$	$\times$	$\times$	$\times$
F7	$\times$	$\surd$	$\times$	$\times$
IPv6[1]	$\times$	$\surd$	$\surd$	$\times$

- In Wireshark was used the follow TCP stream feature
- $\surd$ packets were correctly reassembled
- $\times$ packets were not at all/wrongly reassembled

[1] Not really an attack

Attacking the TCP reassembly software design

PCAP bomb

Problem

A vulnerable reassembly tool assumes that:

- A TCP session is a 4-tuple

Consequences

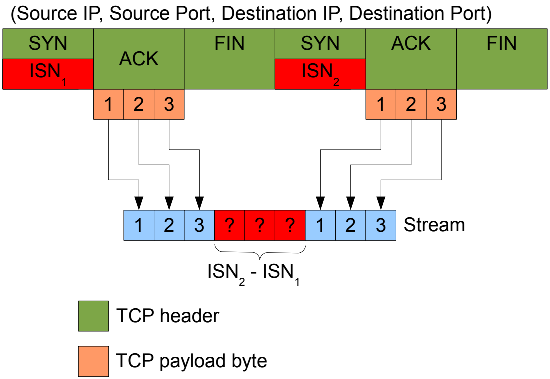
- Different streams are mixed in one file
- Offset between streams due to random ISN (Initial Sequence Number)

Target

- Fill analyst's hard disk
- Memory exhaustion $\rightarrow$ kill high-level stream analysis software

Attacking the TCP reassembly software design

PCAP bomb



Attacking the TCP reassembly software design

PCAP bomb

Proof of concept

Shell

```
tcpdump -i lo -s0 -w pcap-bomb.cap
i=1235
while [ 1 ]; do
j=0
while [ $j -lt 5 ]; do
cat req.txt | socat - tcp:localhost:80,
sourceport=$i,reuseaddr
sleep 1
let j=$j+1
done
let i=i$i+1
done
```

Attacking the TCP reassembly software design

PCAP bomb

- On average each flow has a size of 2GB.
- Tune attack: Write a small PCAP program that maximize ISN difference
- Vulnerable tool: Tcpflow

Hiding Streams 1/2

Problem

A vulnerable reassembly tool assumes that:

- A TCP session is identified by a 4-tuple

Target

- Hide intended web request i.e. rootkit download

How the attack works

- Send dummy data (or just establish a TCP connection)
- Download the real data using the same source port

Hiding Streams

Proof of Concept

Shell

```
$ tcpdump -i lo -s0 -w hidden-stream.cap
$ cat empty.txt | socat - tcp:localhost:80,sourceport=1235,
reuseaddr
$ cat req.txt | socat - tcp:localhost:80,sourceport=1235,
reuseaddr
```

Notes

- empty.txt is an empty file
- req.txt contains an HTTP request to download a file

Mitigating TCP reassembly errors

Countermeasures

- Choose the right capture location (e.g. TTL attack)
- Before analyzing a capture, know how the capture has been performed
- Filter out spoofed packed with a packet filter
- Traffic normalization/scrubing before the capture takes place
  - Reassemble fragments
  - Discard packets with wrong checksums
  - Discard packets with wrong TTL
- Compare results between different analysis tools

Q and A

- Thanks for listening.
- alexandre.dulaunoy@circl.lu