Debugging Tools

Matteo Virgilio

Politecnico di Torino

matteo.virgilio@polito.it

Valgrind

- Valgrind is an open source tool for memory debugging and it is available for Linux.
- Can be exploited to:
  - Find memory leaks;
  - Find Invalid Pointer Use
  - Detect The Use Of Uninitialized Variables
  - Etc etc…
- Complete documentation available at: **http://valgrind.org/docs/**

Valgrind brief HOWTO

1. Compile your code with **–g** option in GCC.
2. Run your program within the Valgrind environment. Assuming your program is executed with the following cmd line:

   ```
   ./server 1500
   ```

   You can simply invoke:

   ```
   valgrind ./server 1500
   ```

3. After quitting your program (e.g. Ctrl-C), you will get the Valgrind output.

Example: a simple calculator

```
#include <stdio.h>
#include <stdlib.h>
#define N 1024

int main(int argc, char* argv[])
{
      char* buffer;
      int op1, op2;
      while(1)
      {
          buffer = (char*)malloc(N*sizeof(char));
          printf ("Insert two integers: ");
          fgets (buffer, N, stdin);
          sscanf (buffer, "%d %d", &op1, &op2);
          printf ("Result=%d\n", op1+op2);
      }
}
```

```
gcc –g –o calc calc.c
```

Output with Valgrind: memory leak found!

```
mettiu@mettiu-virtual-machine:~/valgrind_example$ valgrind ./calc
==5857== Memcheck, a memory error detector
==5857== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==5857== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==5857== Command: ./calc
==5857==
Insert two integers: 10 10
Result=20
Insert two integers: ^C==5857==
==5857== HEAP SUMMARY:
==5857==     in use at exit: 2,048 bytes in 2 blocks
==5857==   total heap usage: 2 allocs, 0 frees, 2,048 bytes allocated
==5857==
==5857== LEAK SUMMARY:
==5857==    definitely lost: 1,024 bytes in 1 blocks
==5857==    indirectly lost: 0 bytes in 0 blocks
==5857==      possibly lost: 0 bytes in 0 blocks
==5857==    still reachable: 1,024 bytes in 1 blocks
==5857==         suppressed: 0 bytes in 0 blocks
==5857== Rerun with --leak-check=full to see details of leaked memory
==5857==
==5857== For counts of detected and suppressed errors, rerun with: -v
==5857== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
```

The revised code

```
#include <stdio.h>
#include <stdlib.h>
#define N 1024

int main(int argc, char* argv[])
{
        char* buffer;
        int op1, op2;
        while(1)
        {
             buffer = (char*)malloc(N*sizeof(char));
             printf ("Insert two integers: ");
             fgets (buffer, N, stdin);
             sscanf (buffer, "%d %d", &op1, &op2);
             printf ("Result=%d\n", op1+op2);
             free (buffer); // Release the allocated memory to the OS
        }
}
```

Output with Valgrind: everything ok!

```
mettiu@mettiu-virtual-machine:~/valgrind_example$ valgrind ./calc
==5884== Memcheck, a memory error detector
==5884== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==5884== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==5884== Command: ./calc
==5884==
Insert two integers: 10 10
Result=20
Insert two integers: ^C==5884==
==5884== HEAP SUMMARY:
==5884==     in use at exit: 1,024 bytes in 1 blocks
==5884==   total heap usage: 2 allocs, 1 frees, 2,048 bytes allocated
==5884==
==5884== LEAK SUMMARY:
==5884==    definitely lost: 0 bytes in 0 blocks
==5884==    indirectly lost: 0 bytes in 0 blocks
==5884==      possibly lost: 0 bytes in 0 blocks
==5884==    still reachable: 1,024 bytes in 1 blocks
==5884==         suppressed: 0 bytes in 0 blocks
==5884== Rerun with --leak-check=full to see details of leaked memory
==5884==
==5884== For counts of detected and suppressed errors, rerun with: -v
==5884== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
```

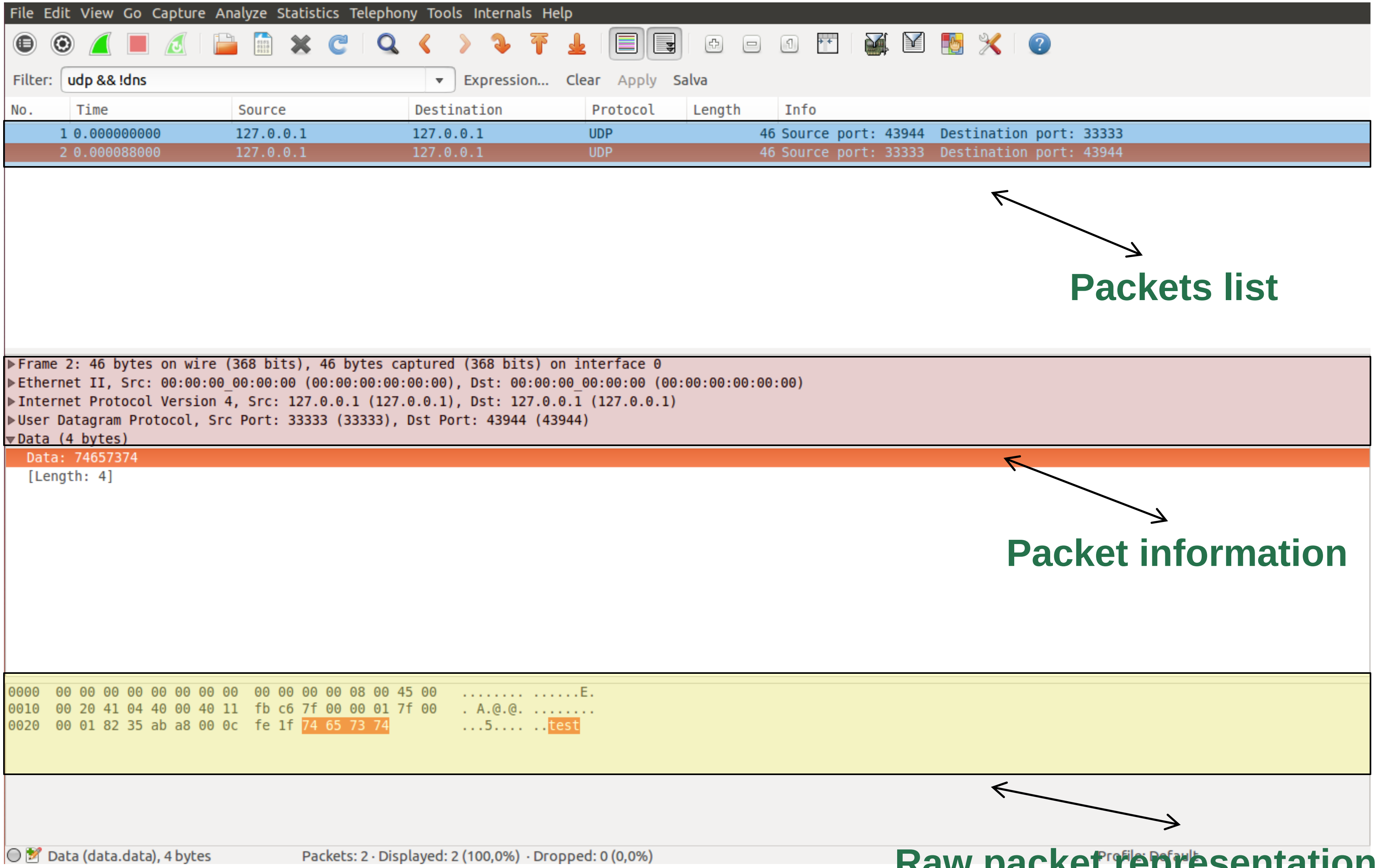
Wireshark

- Wireshark is one of the most famous packet analyzers.
- Cross-platform
- Simple interaction thanks to the advanced GUI provided
- Use wireshark to look at the actual data sent to/by your applications! ☺
- **http://www.wireshark.org/**

Wireshark brief HOWTO

1. Start wireshark (may require root privileges)
2. Select the correct interface on which to capture (loopback if you are working on your local machine)
3. Start the capture
4. Run your data transfer/program to debug
5. Stop the capture
6. Analyze wireshark output

An example


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Salva

Filter: udp && !dns

```
No.   Time          Source      Destination   Protocol   Length  Info
    1 0.000000000   127.0.0.1   127.0.0.1     UDP            46  Source port: 43944  Destination port: 33333
    2 0.000088000   127.0.0.1   127.0.0.1     UDP            46  Source port: 33333  Destination port: 43944
```

Packets list

```
▶Frame 2: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0
▶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: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶User Datagram Protocol, Src Port: 33333 (33333), Dst Port: 43944 (43944)
▼Data (4 bytes)
   Data: 74657374
   [Length: 4]
```

Packet information

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 20 41 04 40 00 40 11  fb c6 7f 00 00 01 7f 00   . A.@.@. ........
0020  00 01 82 35 ab a8 00 0c  fe 1f 74 65 73 74         ...5.... ..test
```

Data (data.data), 4 bytes   Packets: 2 · Displayed: 2 (100,0%) · Dropped: 0 (0,0%)   Profile: Default

Raw packet representation