VTun UDP Packet Dissector for Wireshark/Ethereal

Jason But, Adam Black
Centre for Advanced Internet Architectures, Technical Report 080111A
Swinburne University of Technology
Melbourne, Australia
jbut@swin.edu.au, adamblack@swin.edu.au

***Abstract*— When testing new protocol implementations, VTun can be used to build large virtual networks with multiple hops and paths using a limited amount of hardware. However, it is often necessary to capture and analyse generated traffic to ensure that the protocol is functioning as designed. Capturing tunnelled network traffic results in capturing the tunneling IP/UDP packets and requires an effort to decode and extract the encapsulated IP packets. In this paper we present a patch for Wireshark version 0.99.7 that can automatically decode and further process VTun encapsulated packets to allow deeper inspection into the IP packets being carried within the tunnel.**

I. Introduction

When testing the implementation of network protocols, it is often convenient to run these protocols over a large network. In this manner it is possible to examine the protocol functionality under a variety of different network conditions including multiple hops, congested paths, bandwidth limited paths and various routing conditions. However, it can be inconvenient to establish and maintain a large testbed on a long-term basis. If performance is not an issue, one alternative is to tunnel traffic via multiple virtual interfaces on different computers. By correctly setting up routing tables and firewall rules, it is possible to establish a complex network environment using minimal hardware – as long as performance is not a key measure for testing purposes.

The problem with tunnelling traffic over virtual interfaces comes when we are capturing the network traffic for analysis purposes. If we capture traffic on the physical interface, the packets of interest are encapsulated within the tunnelling protocol packets. This means that we need to decode the tunnelling layer to examine the actual IP packets being transmitted through the tunnel.

VTun [1] is a simple Unix based tunnelling solution that can be used to establish non-encrypted IP tunnels over physical interfaces. For experimental purposes, not encrypting the data can be useful as it minimises processing load and simplifies further packet decoding.

Wireshark [2] (formerly Ethereal) is a graphical network protocol analyser for both traffic trace files and live network capture. In this Technical Report we present and discuss our Wireshark decoding module to decode and process VTun encapsulated packets to enable traffic analysis of IP network traffic transmitted over these tunnels. Our patch has been generated against Wireshark version 0.99.7 only but the decoder module source code has been successfully tested with version 0.99.6 and Ethereal version 0.99.0.

II. VTun in a Network Testbed

When using tunnels to establish a large virtual network testbed, we are typically not interested in securing traffic between two sites, but rather looking to establish multiple virtual network point-to-point links that can then be built into a network environment through the use of routing tables. Using IP tunnels in this manner allows us to examine the workings of IP-based protocols over a variety of different network conditions.

III. Decoding Encapsulated VTun Packets

The module discussed in this report was originally developed as part of the SONATA project [3], to allow us to examine the multi-homing functionality of SCTP [4] in a large network testbed. Since this project was to develop a NAT implementation for SCTP, it was imperative that we be able to capture and analyse the SCTP packets both before and after passing through the NAT. In this case we need to be able to capture SCTP traffic on all the virtual interfaces during each experimental run. Unfortunately, the traffic captured on the physical interface would consist of the VTun UDP packets, with the encapsulated IP packet contained within the UDP packet payload.

To simplify analysis and development of our software, we needed instant feedback on the encapsulated protocol, we decided to develop a decoding module for VTun within the framework of Wireshark [2]. When

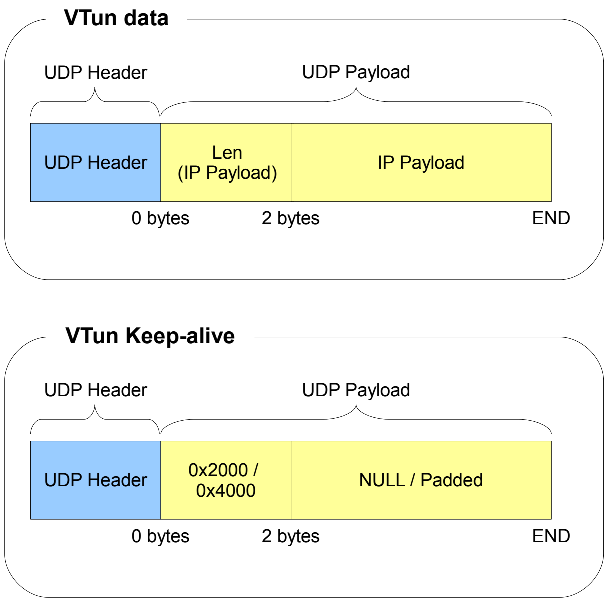


Fig. 1. VTun Packet Structure

Wireshark is compiled with the VTun decoder module, the Wireshark application will automatically treat UDP packets on port 5000 as VTun encoded packets and – if the packets are encapsulated without encryption – decode and process the VTun payload as an IP packet for further processing. This enables all tunnelled traffic to be captured and decoded as if no tunnels were in place.

A. *VTun Packet Format*

VTun uses a client/server model where the client establishes a VTun connection with the server. Once any configured authentication takes place, a virtual interface is established at both ends of the tunnel and all IP packets destined for the tunnel are encapsulated within a UDP packet sent between the two physical hosts. At the server side the UDP port number is typically 5000 (configurable) while at the client side the UDP port number is randomly allocated.

We will further examine the structure of un-encrypted VTun packets which are sent via the UDP protocol. VTun data packets contain the encapsulated IP packet inside the UDP payload. The first two bytes of the payload contain the length of the IP packet. The IP packet itself begins at the 3rd byte of the payload field and continues until the end of the VTun packet.

Depending on your VTun configuration, there may also be VTun keep-alive packets sent between the client and server. These packets contain either 0x2000 or 0x4000 in the length field. See Figure 1 for an example.

B. *Wireshark Module Implementation*

All source code resides in "`packet-vtun.c`" – the majority of this is boiler-plate code to allow successful communication with the main Wireshark application. Key implementation aspects are:

1) VTunRegisterHandoff(): Here we register our VTunDissect() function to handle UDP traffic on port 5000. We also locate the IP dissector plugin which is later used to dissect VTun packets.

2) VTunDissect(): This function handles the dissection of VTun packets. The first step is to check the length field which appears in the first 2 bytes of the UDP payload. If the length matches the number of remaining bytes then the IP dissector is called on the payload of the UDP packet with the first 2 bytes stripped off. If the length field is invalid then we test the length value against 0x2000 and 0x4000. If one of these values is matched it means the VTun packet is a keep-alive/synchronisation message and shouldn't be dissected any further. If the length value does not meet any of the above criteria the packet is marked as unrecognisable.

More details about creating a Wireshark dissector plugin can be found in the Wireshark online manual [5].

IV. Installation

The module is available for download at `http://caia.swin.edu.au/urp/sonata/downloads.html`. The download file is a gzipped tarball consisting of this document and a patch to run against the Wireshark source code. Instructions for installation follow.

A. *Requirements*

To compile the VTun plugin you will require:

- UNIX-like operating system
- Wireshark source code 0.99.7 [2]
- VTun patch for wireshark

The patch must be manually applied to the source-code and cannot be applied to a previous or new packaged install, or to any pre-compiled binary installation. If you already have Wireshark installed you must uninstall your current version prior to installing the patched version.

The patch provided as part of this download will only cause the VTun decoder module to be compiled under a Unix-like operating system because the patch does not touch or create the required *nmake* files for compiling under Windows. If you need to compile the patch for Windows, please read the documentation file included with the Wireshark source (`wireshark-src/doc/README.plugins`)

The provided patch has been generated against version 0.99.7 of Wireshark and will only work if you download this version of the Wireshark source code. The actual VTun decoder module should compile correctly against any Wireshark/Ethereal version but you may need to manually modify the patch first. The VTun decoder module has been successfully compiled with Wireshark versions 0.99.6 and 0.99.7 and Ethereal version 0.99.0 on both FreeBSD and Linux platforms.

B. Compiling the VTun Module into Wireshark/Ethereal

As previously noted, all the source code required to patch Wireshark to support VTun decoding is contained within the file "`plugins/vtun/packet-vtun.c`". This code should compile against other versions of Wireshark. The provided patch is listed to only work against Wireshark version 0.99.7 because of the modifications it makes to existing files to ensure that the new VTun module gets compiled and installed.

The steps listed below assume that you have downloaded the Wireshark 0.99.7 source code.

1) Extract the wireshark-0.99.7 source into a directory in which you have write privileges
2) Copy the VTun wireshark patch (`wireshark_vtun_patch.sh`) into the `wireshark-0.99.7/` base directory
3) Enter the wireshark-0.99.7 directory and execute:
   ```
   sh wireshark_vtun_patch.sh
   ```
   Check the patch output carefully, all the files should have been patched successfully. If you see any messages saying 'Hunk Failed' you will need to skip to Section IV-C before proceeding with these instructions.
4) Now execute:
   ```
   ./autogen.sh
   ./configure
   make
   ```
   `autogen.sh` must be run to ensure that the `configure` script is regenerated. Whilst running either `autogen.sh` or `configure`, you may be prompted to install some required dependencies for Wireshark. If so, please install the required packages and then re-run `autogen.sh/configure`
5) To install the Wireshark/Ethereal binaries in your system run as the root user:
   ```
   make install
   ```

Alternatively, if you are using FreeBSD and wish to install Wireshark from the ports tree as a registered package, you can use the following steps:

1) Change into the proper port directory
   ```
   /usr/ports/net/wireshark
   ```
2) Download the Wireshark source code and patch with FreeBSD specific patches
   ```
   make patch
   ```
3) Change into the compile directory (where XXX is the current Wireshark version number)
   ```
   cd work/wireshark-XXX
   ```
4) Follow steps 2, 3 and 4 above to install the patch and modify the source code. For step 4 you need only execute the `autogen.sh` script since the FreeBSD ports build will run `configure` for you. This step is required otherwise Ports will use the original `configure` script. Dependencies should not be an issue since the ports dependency tree will have ensured that everything is already installed
5) Change back to the port directory and build/install Wireshark
   ```
   cd ../..
   make install
   ```
   The FreeBSD Ports system will automatically regenerate and execute the "`configure`" script prior to building and installing Wireshark

You should now have a working installation of Wireshark/Ethereal with the required plugin for decoding unencrypted UDP VTun packets. To run Wireshark or Ethereal simply type '`wireshark`' or '`ethereal`' at the command prompt. The VTun module should be automatically loaded upon launch, to check if the VTun module compiled successfully click the '**Help**' menu, '**About**' and select the '**Plugins**' tab. Scroll down the list to check for the **vtun.so** dissector.

If Wireshark detects UDP traffic on port 5000 it will call the VTun packet dissector which will recursively dissect the packet. Under Ethereal the VTun payload must be manually dissected by traversing the Protocol Hierarchy tree.

C. Manually Patching Ethereal or Wireshark Source

The provided patch is designed to work against version 0.99.7 of the Wireshark source code. If you are compiling against a different version of Wireshark or Ethereal – or your received a 'Hunk Failed' error message during `wireshark_vtun_patch.sh` execution – you will need to manually patch the source code.

When executing `wireshark_vtun_patch.sh`, the first files created/modified are the the VTun module source code files, these files should be generated regardless of the version of Wireshark/Ethereal you are patching. Once the source code has been added, the remaining patches are to "`Makefile.am`", "`configure.in`" and "`plugins/Makefile.am`". These changes are required to ensure that running `configure`/`make` will build Wireshark/Ethereal with the VTun module.

Patching the remaining three files may not work for other versions of Wireshark/Ethereal. The are two possible means of patching these files:

1) Read the `doc/README.plugins` file in the Wireshark/Ethereal base directory and locate the section outlining the necessary modifications to "`Makefile.am`", "`configure.in`" and "`plugins/Makefile.am`". This should only be a simple one line addition to each file
2) Alternatively, a 'cleaner' solution would require editing the `wireshark_vtun_patch.sh` file and modifying the line numbers that describe where to patch the specified files. This technique is for sufficiently advanced users and will not be covered in any more detail

Acknowledgements

The development of the Wireshark VTun decoding module was completed as part of the SONATA [3] project and was made possible through the funding of Cisco.

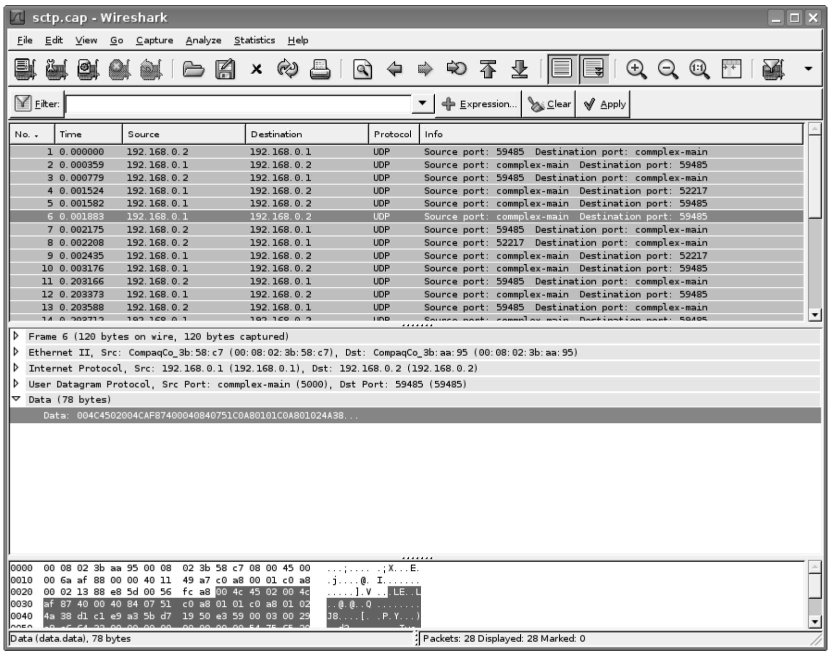
References

[1] Maxim Krasnyansky, "VTun – Virtual Tunnels over TCP/IP networks," Viewed 02 January 2008, http://vtun.sourceforge.net/.
[2] Wireshark, "Wireshark," Viewed 02 January 2008, http://www.wireshark.org/download/src/all-versions/.
[3] CAIA, "SONATA – SCTP Over NAT Adaptation," Viewed 02 January 2008, http://caia.swin.edu.au/urp/sonata.
[4] Randall Stewart, "Stream Control Transmission Protocol (SCTP)," Viewed 02 January 2008, http://www.sctp.org.
[5] Wireshark, "Wireshark Developers Documentation - Adding a basic dissector," Viewed 15 January 2008, http://www.wireshark.org/docs/wsdg_html_chunked/ChDissectAdd.html.

Appendix A

Figures 2 and 3 show screenshots of Wireshark 0.99.7 after opening a traffic trace containing SCTP traffic over an unencrypted VTun tunnel. Figure 2 shows the default Wireshark output without the VTun module loaded. In this case we see only the original captured traffic. Wireshark decodes the IP/UDP header and displays the VTun encapsulating packet information but does not decode the encapsulated IP/SCTP packets.

Figure 3 show Wireshark output with the the VTun module loaded. In this case the protocol heirarchy tree automatically shows the UDP payload as a VTun unencrypted tunnel and further examination of the payload shows the details of the encapsulated IP/SCTP packets. Further in the summary window showing packet information, the encapsulated IP/SCTP packet information is summarised as opposed to the encapsulating IP/UDP VTun packet.



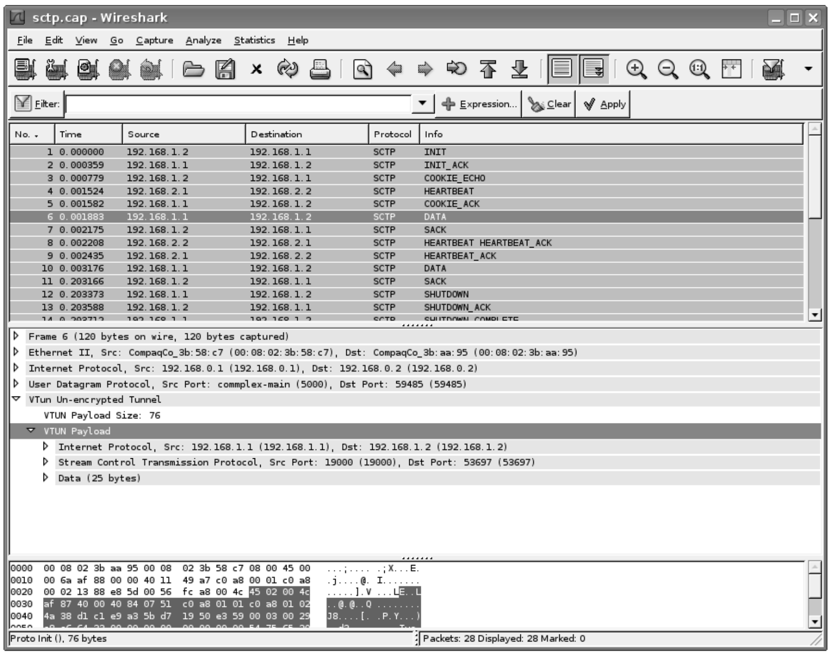
```
No. .  Time      Source         Destination    Protocol  Info
     1 0.000000  192.168.0.2    192.168.0.1    UDP       Source port: 59485  Destination port: commplex-main
     2 0.000359  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 59485
     3 0.000779  192.168.0.2    192.168.0.1    UDP       Source port: 59485  Destination port: commplex-main
     4 0.001524  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 52217
     5 0.001582  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 59485
     6 0.001883  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 59485
     7 0.002175  192.168.0.2    192.168.0.1    UDP       Source port: 59485  Destination port: commplex-main
     8 0.002208  192.168.0.2    192.168.0.1    UDP       Source port: 52217  Destination port: commplex-main
     9 0.002435  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 52217
    10 0.003176  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 59485
    11 0.203166  192.168.0.2    192.168.0.1    UDP       Source port: 59485  Destination port: commplex-main
    12 0.203373  192.168.0.1    192.168.0.2    UDP       Source port: commplex-main  Destination port: 59485
    13 0.203588  192.168.0.2    192.168.0.1    UDP       Source port: 59485  Destination port: commplex-main
```

```
▷ Frame 6 (120 bytes on wire, 120 bytes captured)
▷ Ethernet II, Src: CompaqCo_3b:58:c7 (00:08:02:3b:58:c7), Dst: CompaqCo_3b:aa:95 (00:08:02:3b:aa:95)
▷ Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
▷ User Datagram Protocol, Src Port: commplex-main (5000), Dst Port: 59485 (59485)
▽ Data (78 bytes)
    Data: 004C4502004CAF87400040840751C0A80101C0A801024A38...
```

```
0000  00 08 02 3b aa 95 00 08  02 3b 58 c7 08 00 45 00   ...;.... .;X...E.
0010  00 6a af 88 00 00 40 11  49 a7 c0 a8 00 01 c0 a8   .j....@. I.......
0020  00 02 13 88 e8 5d 00 56  fc a8 00 4c 45 02 00 4c   .....].V ...LE..L
0030  af 87 40 00 40 84 07 51  c0 a8 01 01 c0 a8 01 02   ..@.@..Q ........
0040  4a 38 d1 c1 e9 a3 5b d7  19 50 e3 59 00 03 00 29   J8....[. .P.Y...)
```

```
Data (data.data), 78 bytes          Packets: 28 Displayed: 28 Marked: 0
```

Fig. 2. Wireshark without VTun plugin enabled



```
No. .  Time      Source         Destination    Protocol  Info
     1 0.000000  192.168.1.2    192.168.1.1    SCTP      INIT
     2 0.000359  192.168.1.1    192.168.1.2    SCTP      INIT_ACK
     3 0.000779  192.168.1.2    192.168.1.1    SCTP      COOKIE_ECHO
     4 0.001524  192.168.2.1    192.168.2.2    SCTP      HEARTBEAT
     5 0.001582  192.168.1.1    192.168.1.2    SCTP      COOKIE_ACK
     6 0.001883  192.168.1.1    192.168.1.2    SCTP      DATA
     7 0.002175  192.168.1.2    192.168.1.1    SCTP      SACK
     8 0.002208  192.168.2.2    192.168.2.1    SCTP      HEARTBEAT HEARTBEAT_ACK
     9 0.002435  192.168.2.1    192.168.2.2    SCTP      HEARTBEAT_ACK
    10 0.003176  192.168.1.1    192.168.1.2    SCTP      DATA
    11 0.203166  192.168.1.2    192.168.1.1    SCTP      SACK
    12 0.203373  192.168.1.1    192.168.1.2    SCTP      SHUTDOWN
    13 0.203588  192.168.1.2    192.168.1.1    SCTP      SHUTDOWN_ACK
```

```
▷ Frame 6 (120 bytes on wire, 120 bytes captured)
▷ Ethernet II, Src: CompaqCo_3b:58:c7 (00:08:02:3b:58:c7), Dst: CompaqCo_3b:aa:95 (00:08:02:3b:aa:95)
▷ Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
▷ User Datagram Protocol, Src Port: commplex-main (5000), Dst Port: 59485 (59485)
▽ VTun Un-encrypted Tunnel
    VTUN Payload Size: 76
  ▽ VTUN Payload
    ▷ Internet Protocol, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.2 (192.168.1.2)
    ▷ Stream Control Transmission Protocol, Src Port: 19000 (19000), Dst Port: 53697 (53697)
    ▷ Data (25 bytes)
```

```
0000  00 08 02 3b aa 95 00 08  02 3b 58 c7 08 00 45 00   ...;.... .;X...E.
0010  00 6a af 88 00 00 40 11  49 a7 c0 a8 00 01 c0 a8   .j....@. I.......
0020  00 02 13 88 e8 5d 00 56  fc a8 00 4c 45 02 00 4c   .....].V ...LE..L
0030  af 87 40 00 40 84 07 51  c0 a8 01 01 c0 a8 01 02   ..@.@..Q ........
0040  4a 38 d1 c1 e9 a3 5b d7  19 50 e3 59 00 03 00 29   J8....[. .P.Y...)
```

```
Proto Init (), 76 bytes          Packets: 28 Displayed: 28 Marked: 0
```

Fig. 3. Wireshark with VTun plugin enabled