GSE Broadband Arsenal Lab (Blackhat Europe 2021)

Overview

GSE (Generic Stream Encapsulation) is one of the most commonly used protocols for transmitting internet traffic through Geostationary Earth Orbit (GEO). Like MPEG-TS in the first lab, GSE is a transport layer written on top of the DVB-S protol.

In this lab, you will learn how to begin to reverse engineer GSE traffic. You will use the open source format description languge `kaitai` to build a basic GSE parser and eavesdrop on simulated satellite internet traffic.

Launching Your Satellite

Open a command terminal with `CTRL+ALT+T` and navigate to the lab directory by running the following command:

```
cd ~/satcoms_demo/
```

Next, connect to your virtual SATCOMs workstation by running the following command:

```
./reset_exercise.sh
```

After a few seconds, a command line should open within your lab machine.

We're going to work inside the `/lab_resources/` directory for this exercise, so go ahead and navigate there now with the following command:

```
cd /lab_resources/
```

Exploring Your Traffic

Since we're indoors (and can't see any satellites), we're going to use a simulated satellite signal. Unlike in the MPEG-TS lab, we're going to abstract away the lower physical layers of the signal and work direclty with demodulated DVB-S.

Note: In real-world practice, this can be a difficult step as GSE feeds tend to use complex modulation schemes that require larger dishes and fast procssing hardware. I recommend the `TBS-6983` PCI tuner card (rather than a general purpose SDR) if you're interested in exploring these signals in the wild.

Let's go ahead and connect to our simulated satellite and make a quick recording with the following command.

Let it run for 30 or so seconds, then stop it with `CTRL+C`:

```
nc localhost 50000 > my_recording.gse
```

Spend some time familiarizing yourself with this data. You can use the hex editor `okteta` (installed on your laptop), or the command line tool `xxd` to look at the file you've captured. Try and answer the following questions for yourself. You can find your recording in the `~/satcoms_demo/lab_volume` directory.

- What sort of traffic do you think you've captured here?
- Can you see the start and end of any payloads or application packets?

- Can you identify a full, valid, IP packet? Try copy and pasting data from your hex editor into HPD (https://hpd.gasmi.net/) or Paketor (http://packetor.com/) and deleting bytes until it parses logically.

Open Kaitai

Identifying the relevant protocol and building a complete parser can take a long time, so we're going to abridge the process a bit here by assuming that you've gotten a tip that it's using the open GSE standard, as described in `~/satcoms_demo/lab_volume/gse_packet_spec.pdf`.

We're going to use to the open-source tool `kaitai` to build our packet parser. Open up your web browser and navigate to `localhost:8081` to open up the `kaitai` web development environment.

Create a new file with the name `gse` using the button in the bottom left corner of the screen. This is where we will write our description of the GSE format.

To get you started, go ahead and copy and paste the contents of `~/satcoms_demo/lab_volume/gse_template.ksy` into the Kaitai IDE. You'll see that most of this file starts commented out, and there are a few places where you'll need to add information yourself to the code.

Writing Your Protocol Spec

Look at the GSE header specification in `~/sat_carving_lab/templates/gse_header_format.pdf` and use it to fill out portions of the .ksy template which read "#CONFIGURE THIS#". Your task is to refer to the specification and set the correct sizes for each of the header elements so that the parser can interpret packets correctly.

To do this, you will need to uncomment parts of your Kaitai code as they become relevant. Detailed instructions appear below, but if you get stumped feel free to ask questions or play around with things.

Parser Steps

1. Uncomment the YAML object beginning on 20 (`end_indicator`) and replace the "#CONFIGURE THIS#" (on line 21) with the correct size from the standard.
2. Do the same for the `label_type_indicator` element (begins on line 22).
3. Do the same for the `padding_bits` element but also uncomment the `instances` array on line 42 and the first element of that array, the `is_padding_packet` instance on lines 43 & 44.
4. Continue this process for the remaining elements of the GSE header, being sure to set the appropriate value for the speicification and uncomment the corresponding `instances` elements wherever you se a line beging with an `if:` statement. *At this point, you have a full parser for the GSE header and should now be able to recognize where GSE payloads start and stop in the GUI display on the right side of the IDE.*
5. Go ahead and uncomment the `gse_payload` object begining on line 57.
6. Finally, uncomment lines 12-15. If you correctly specified the GSE header, you should now see the first GSE packet highlighted in the kaitai IDE.

Generating a Parser from Your Spec

Right click on the `gse.ksy` file you created on the left side of the Kaitai IDE and select `Generate parser -> python`.

Copy and paste the python code which Kaitai gives you and save it in the `~/satcoms_demo/lab_volume/` directory with the name `gse.py`

Running the Parser

From inside the lab machine console, navigate to `/lab_resources/` (`cd /lab_resources`) and run the following command to use your parser and try to make a valid pcap file.

```
python3 convert_gse.py test.gse output.pcap
```

Note: The convert_gse.py file is a wrapper which adds some additional logic over the packet spec your wrote to deal with fragmented packets in the GSE format and to convert your extracted payloads to pcaps. Feel free to check out the code if you're curious!

Exploring the Traffic

If everything worked, you should be able to open the packets in Wireshark from your main operating system (the `pcap` file will be in `~~/satcoms_demo/lab_volume`).

Congratulations! You now understand some of the most important parts of carving IP packets out of one of the most popular satellite broadband protocols used today.

If you're interested in learning more, see if you can implement `convert_gse.py` yourself based on the GSE protocol specification.

For a more complex variation of the same basic problem - which deals with corrupted data that you get in some real-world recordings - check out my open source tool `GSExtract` (https://github.com/ssloxford/gsextract) and the corresponding Blackhat USA 2020 briefing on the sensitive information it helped us intercept (https://youtu.be/d5Sbwlu6f8o).