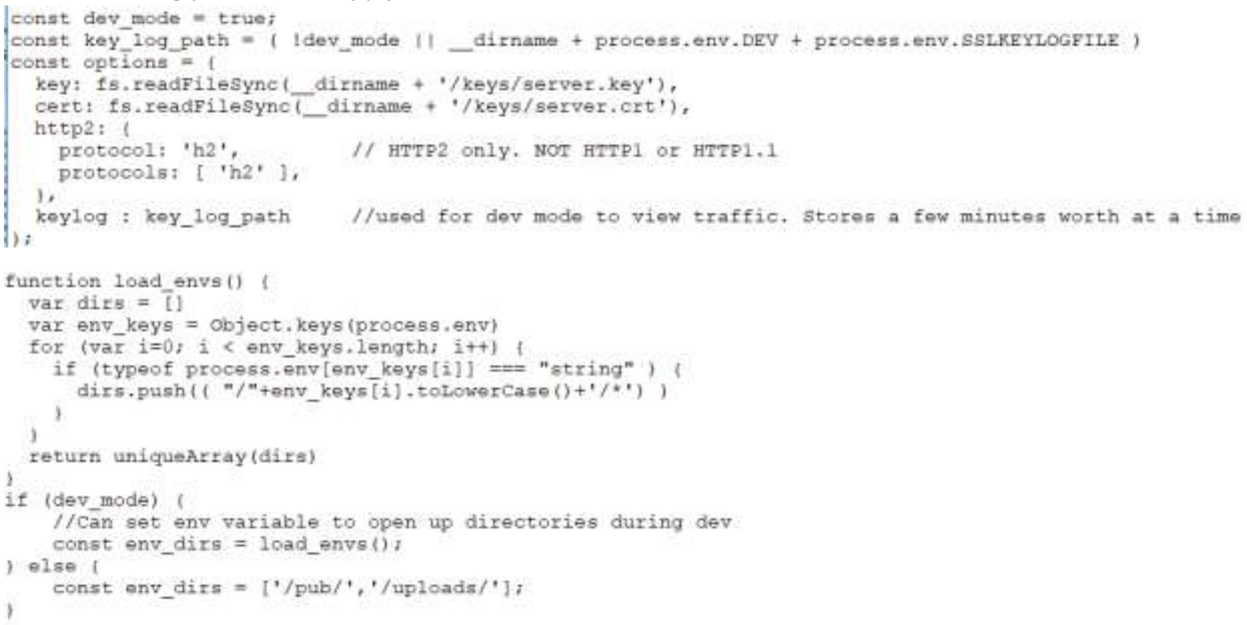
Objective--Network Traffic Forensics (Part 3)

Solution (or part of it)

The interesting parts of the app.js file are here.



```
const dev_mode = true;
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
const options = {
  key: fs.readFileSync(__dirname + '/keys/server.key'),
  cert: fs.readFileSync(__dirname + '/keys/server.crt'),
  http2: {
    protocol: 'h2',         // HTTP2 only. NOT HTTP1 or HTTP1.1
    protocols: [ 'h2' ],
  },
  keylog : key_log_path     //used for dev mode to view traffic. Stores a few minutes worth at a time
};

function load_envs() {
  var dirs = []
  var env_keys = Object.keys(process.env)
  for (var i=0; i < env_keys.length; i++) {
    if (typeof process.env[env_keys[i]] === "string" ) {
      dirs.push(( "/"+env_keys[i].toLowerCase()+'/*') )
    }
  }
  return uniqueArray(dirs)
}
if (dev_mode) {
    //Can set env variable to open up directories during dev
    const env_dirs = load_envs();
} else {
    const env_dirs = ['/pub/','/uploads/'];
}
```

We are looking for the SSLKEYLOGFILE, according to HTTP/2 Decryption and Analysis in Wireshark. Sure enough, there is a line with exactly what we are looking for.

```
const key_log_path = ( !dev_mode || __dirname + process.env.DEV +
process.env.SSLKEYLOGFILE )
```

The environment variable is `SSLKEYLOGFILE`.

The function `load_envs()` takes all the environment variables, converts them to lower case and pushes them into a list. That is strange, but maybe it is trying to make the code scalable as the environment variables article suggests. You had better be careful with your environment variables if you do that.

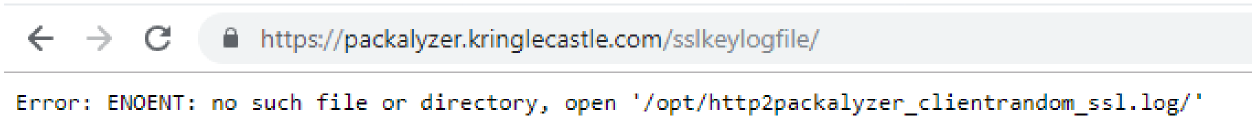
The `if` statement opens directories to all environment variables if `dev_mode` is True. If `dev_mode` is False it opens the directories `put` and `uploads`.

When we look back up to the constants, we find this, so the application is in `dev_mode`.

```
const dev_mode = true;
```

It appears our developer was not careful with the environment variables.

Therefore, the server should be opening a directory or file like the value stored in `sslkeylogfile`. Browsing to that directory gives us this, so it appears the file name is `http2packalyzer_clientrandom_ssl.log.`



https://packalyzer.kringlecastle.com/sslkeylogfile/

```
Error: ENOENT: no such file or directory, open '/opt/http2packalyzer_clientrandom_ssl.log/'
```

What a weird and wonderful (for attackers) that error message is!

However, `/opt/http2packalyzer_clientrandom_ssl.log/` looks strange. Let's go back to the constant that created that string.

```
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
```

We know that `dev_mode` is `True`, so `!dev_mode` is `False`. The OR (||) is using short-cut execution. If the first part of the OR is True, the entire statement is True so the second part does not need to be executed. If the first part is False, the second part must be evaluated to determine if the statement is True or False. The second part is only executed when the first part is false.

Therefore, this is executed.

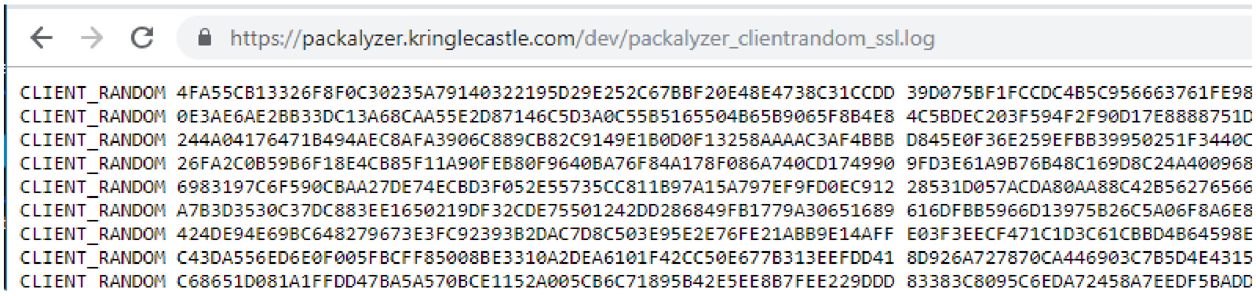
```
__dirname + process.env.DEV + process.env.SSLKEYLOGFILE
```

The internal function `__dirname` gives the current directory. Then `process.env.DEV` must give the value that the DEV environment variable points to. Finally, `process.env.SSLKEYLOGFILE` gives the value of the SSLKEYLOGFILE. So,

```
    __dirname                  is /opt/
    process.env.DEV            is http2
    process.env.SSLKEYLOGFILE  is packalyzer_clientrandom_ssl.log
```

The missing '/' in the code we just examined makes http2 look like part of the file name, but it is not. The file name is `packalyzer_clientrandom_ssl.log.`

We didn't find the file in the /pub directory. The /opt/http2/ directories are local to the server, not what is published by the webserver. Let's hope the web directory is dev/; after all, there is a DEV environment variable.
https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log/



https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log

```
CLIENT_RANDOM 4FA55CB13326F8F0C30235A79140322195D29E252C67BBF20E48E4738C31CCDD 39D075BF1FCCDC4B5C956663761FE98
CLIENT_RANDOM 0E3AE6AE2BB33DC13A68CAA55E2D87146C5D3A0C55B5165504B65B9065F8B4E8 4C5BDEC203F594F2F90D17E8888751D
CLIENT_RANDOM 244A04176471B494AEC8AFA3906C889CB82C9149E1B0D0F13258AAAAC3AF4BBB D845E0F36E259EFBB39950251F3440C
CLIENT_RANDOM 26FA2C0B59B6F18E4CB85F11A90FEB80F9640BA76F84A178F086A740CD174990 9FD3E61A9B76B48C169D8C24A400968
CLIENT_RANDOM 6983197C6F590CBAA27DE74ECBD3F052E55735CC811B97A15A797EF9FD0EC912 28531D057ACDA80AA88C42B56276566
CLIENT_RANDOM A7B3D3530C37DC883EE1650219DF32CDE75501242DD286849FB1779A30651689 616DFBB5966D13975B26C5A06F8A6E8
CLIENT_RANDOM 424DE94E69BC648279673E3FC92393B2DAC7D8C503E95E2E76FE21ABB9E14AFF E03F3EECF471C1D3C61CBBD4B64598E
CLIENT_RANDOM C43DA556ED6E0F005FBCFF85008BE3310A2DEA6101F42CC50E677B313EEFDD41 8D926A727870CA446903C7B5D4E4315
CLIENT_RANDOM C68651D081A1FFDD47BA5A570BCE1152A005CB6C71895B42E5EE8B7FEE229DDD 83383C8095C6EDA72458A7EEDF5BADD
```

Yes! Copy the contents of the page and paste it into a text editor. We can move on to decrypting packets. Finally!

Hand In

Follow the steps in Chris' video and see what you can glean from the pcap file you downloaded from Packalyzer. It would be nice to find an answer to the objective, but if not, credentials are always good!

Note: Download the packet capture file, and then grab the SSLKEYLOGFILE soon afterwards. If there's a large time delay, the keys may not match the capture file.

1) Is there a user name and password in the pcap file?

2) If you find credentials, where would be a good place to use them? If you are lucky, you will find something with 'secret' in the name. What is it?