

ENTRY // Xiongmai Security Research #5 // Advanced Fuzzing

[\[- \]](#) [\[X \]](#)

Date: 2019-09-04 Author: Ian Rash Index: #5

During my testing I noticed an interesting quirk of this protocol, the server (camera) would allow as many connections as one wanted open to the same ip address, so long as they were all on different ports. This meant that if I could create a "pool" of sockets, I could use them to audit multiple magics at once, each waiting for their own replies.

You can view the relevant code on [GitHub](#).

Fuzzing Command::SystemInfo

Time: 00:29:14.794430000

Current: 4096/4097 : 1000

Total Completion: 99.976%

Waiting for magics:

0x0ffc : unused : 15642636659266745398 : 00:00:00.394592000

0x0ffd : unused : 3995498554981886474 : 00:00:00.394431000

0x0ff1 : unused : 16849123052220723596 : 00:00:00.424488000

0x0ffe : unused : 15843022912141103538 : 00:00:00.385055000

0x0ff2 : unused : 666834066939202384 : 00:00:00.424001000

0x0ff3 : unused : 11959220922209025486 : 00:00:00.423846000

0x0ff4 : unused : 9858625403406765244 : 00:00:00.423865000

0x0ff5 : unused : 20212055150009910 : 00:00:00.423179000

0x0fff : unused : 15147142017989187717 : 00:00:00.384266000

0x0ff6 : unused : 16036212785124225768 : 00:00:00.423033000

0x0ff7 : unused : 3934626923425214118 : 00:00:00.423048000

0x0fef : unused : 784495433133620875 : 00:00:00.465630000

0x0ff0 : unused : 8924739629740135316 : 00:00:00.465648000

0x0ff8 : unused : 17166435733447359522 : 00:00:00.422446000

0x0ff9 : unused : 11108002682450497409 : 00:00:00.422467000

0x0ffa : unused : 11116907754345188397 : 00:00:00.421792000

0x0ffb : unused : 8156710575546691230 : 00:00:00.421819000

0x1000 : unused : 6252091348165092127 : 00:00:00.384556000

0x0fe9 : unused : 5183855669207984885 : 00:00:00.751042000

0x0fea : unused : 829040888724800310 : 00:00:00.750799000

Status

Factory: done

Last Check In: 2019-04-17 10:59:17 -07:00

Total Successes: 3983

Total Unique Replies: 49

Total Bad Results: 114

Error:

Errors: {}

Using this we can fuzz a space of 0x0 to 0x1000 in only 30 minutes!

Each fiber will use it's own socket, send a message, then wait to receive on. If the timeout becomes to great, it drops the message and moves to the next one.

If the camera turns off for some reason, a 2 minute grace period is applied to the time out, to wait for it to attempt to come back. This to ensure all the ground is still covered. This does mean that someone does have to be around to restart the camera if it does go down, but maybe I'll make a raspberry pi that shuts the thing down with a relay one day.

You can view an example log [here](#). I'll be dissecting this log for starters.

The most common reply is,

```
"{ \"Name\" : \"SystemInfo\", \"Ret\" : 102, \"SessionID\" : \"0x00000000\" }\\n"
```

With around 4000 magics.

This one is a login failure packet. Interestingly enough, it reports a different error code if it doesn't get a password or username, 205.

```
"{ \"AliveInterval\" : 0, \"ChannelNum\" : 0, \"DeviceType\" : \"DVR\", \"ExtraChannel\" : 10744252, \"Ret\" : 205, \"SessionID\" : \"0x00000000\" }\\n"
Bytes: ["0x03e8"]
```

Whatever these ones do, they report success. Could be interesting to find out exactly what these guys do.

```
"{ \"Name\" : \"\", \"Ret\" : 100, \"SessionID\" : \"0x00000000\" }\\n"
Bytes: ["0x03ea", "0x0410", "0x0416", "0x041a", "0x0578", "0x05e0", "0x05dc", "0x05de", "0x0670", "0x06ea", "0x0684"]
```

This one is the "keep alive", which keeps the connection to the camera going as long as it receives a keep alive packet.

```
"{ \"Name\" : \"KeepAlive\", \"Ret\" : 100, \"SessionID\" : \"0x00000000\" }\\n"
Bytes: ["0x03ee"]
```

After some pretty standard results, as well as the actual reply for SystemInfo, we eventually get to an interesting territory that shows us a little insight into the protocol.

```

"{ \"Name\" : \"OPMonitor\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x0582\", \"0x0585\"]
\"{ \"Name\" : \"OPPlayBack\", \"Ret\" : 100, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x058c\", \"0x0591\"]
\"{ \"Name\" : \"OPPlayBack\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x0590\"]
\"{ \"Name\" : \"OPTalk\", \"Ret\" : 504, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x0596\"]
\"{ \"Name\" : \"OPTalk\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x059a\", \"0x059b\"]
\"{ \"Name\" : \"\", \"Ret\" : 119, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x05a0\"]
\"{ \"Name\" : \"OPLogQuery\", \"OPLogQuery\" : null, \"Ret\" : 100, \"SessionID\" : \"0x0\" }\\n\"
  Bytes: [\"0x05a2\"]
\"{ \"Name\" : \"OPSCalendar\", \"OPSCalendar\" : { \"Mask\" : 0 }, \"Ret\" : 100, \"SessionID\" : \"0x0\" }\\n\"
  Bytes: [\"0x05a6\"]
\"{ \"Name\" : \"\", \"Ret\" : 109, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x05a8\"]
\"{ \"Name\" : \"OPTimeQuery\", \"OPTimeQuery\" : \"2000-12-07 02:55:43\", \"Ret\" : 100, \"SessionID\" : \"0x0\" }\\n\"
  Bytes: [\"0x05ac\"]
\"{ \"Name\" : \"\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x05b4\", \"0x0828\"]

```

The original command we were fuzzing is "SystemInfo". Why is the returning name something different, like OPSCalendar? This is interesting, it means that not only does the magic field control command type, some of these commands are not programmed with very much error checking, so they can produce some wild results when poked and prodded in the right way. We also now have new command names to fuzz.

```

"{ \"AuthorityList\" : [ \"ShutDown\", \"ChannelTitle\", \"RecordConfig\", \"Backup\", \"StorageManager\", \"Accou
  Bytes: [\"0x05be\"]
\"{ \"Ret\" : 100, \"SessionID\" : \"0x00000000\", \"Users\" : [ { \"AuthorityList\" : [ \"ShutDown\", \"ChannelTitle\", \
  Bytes: [\"0x05c0\"]
\"{ \"Groups\" : [ { \"AuthorityList\" : [ \"ShutDown\", \"ChannelTitle\", \"RecordConfig\", \"Backup\", \"StorageMan
  Bytes: [\"0x05c2\"]

```

Now we start getting into the tastiest meat!

```

{ \"AuthorityList\" : [ \"Monitor_01\", \"Replay_01\" ], \"Memo\" : \"user group\", \"Name\" : \"user\" } ], \"Ret\" : 1

```

Here we find a hidden user account, "default" with stream only privileges. This is great! It's a new avenue to check. It's possible the "authorities list" wasn't set up right, and it may allow us to access functions of the camera without the need for a login. This account also isn't mentioned anywhere in the manuals for the device.

We can also see the full authority list for "admin", which includes shutdown and upgrade privileges. Juicy!

```
BINARY FILE "{ \"command\" : \"sync\", \"  
Bytes: [\"0x0666\"]
```

This one is particularly interesting, my fuzzer flagged it as a "binary file" because it couldn't parse it into JSON. It seems like the command cut off part way through sending, maybe something interesting is happening (like crash).

```
BINARY FILE "PK\u0003\u0004\u0014\u0000\u0000\u0000\b\u0000\u0000\u0000 \u0000\u0000FiP\xB9\ad\u0000  
Bytes: [\"0x0606\"]  
BINARY FILE "PK\u0003\u0004\u0014\u0000\u0000\u0000\b\u0000\u0000\u0000 \u0000\u0000xE6\xE5\x90\u0018\u0000  
Bytes: [\"0x0608\"]  
BINARY FILE "PK\u0003\u0004\u0014\u0000\u0000\u0000\b\u0000\u0000\u0000 \u0000\u0000xC4\u0003#\u0018\u0000  
Bytes: [\"0x066c\"]
```

We also get some zip files which contains settings dumps, while interesting don't contain anything we didn't already know about the camera.

```
BINARY FILE "\xFF\xD8\xff\xE0\u0000\u0010JFIF\u0000\u0001\u0001\u0000\u0000\u0001\u0000\u0001\u0000  
Bytes: [\"0x0618\"]
```

0x0618 gives us an image from the camera, will be useful for later.

Overall, we've gained some interesting insight into the device, and we haven't even fuzzed all the commands, and the unauthenticated user account yet!

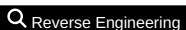
The user account "default" ends up giving us a clear picture of what's going on. Since it only has stream privileges, most of its replies are stream related.

Command results: Started at 2019-04-18 20:07:00 -07:00

Total time: 00:51:34.994305000

```
"{ \"AliveInterval\" : 0, \"ChannelNum\" : 0, \"DeviceType\" : \"DVR\", \"ExtraChannel\" : 10976316, \"Ret\" : 205, \"S
  Bytes: [\"0x03e8\"]
\"{ \"Name\" : \"\", \"Ret\" : 102, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x03f2\", \"0x080e\"]
\"{ \"Name\" : \"OPMonitor\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x0585\"]
\"{ \"Name\" : \"OPPlayBack\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x0590\"]
\"{ \"Name\" : \"OPTalk\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x059a\"]
\"{ \"Name\" : \"GetSafetyAbility\", \"Ret\" : 103, \"SessionID\" : \"0x00000000\", \"authorizeStat\" : null }\\n\"
  Bytes: [\"0x0672\"]
\"{ \"Name\" : \"OPRecordSnap\", \"Ret\" : 100, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x07fc\"]
\"{ \"Name\" : \"\", \"Ret\" : 105, \"SessionID\" : \"0x00000000\" }\\n\"
  Bytes: [\"0x0852\"]
\"{ \"Name\" : \"\", \"Ret\" : 106, \"SessionID\" : \"0x000066E9\" }\\n\"
  Bytes: [\"0x02ee\", \"0x0192\", \"0x00a7\", \"0x0e27\", \"0x0041\", \"0x01c1\", \"0x0032\", \"0x0fa6\", \"0x03f7\", \"0x0740\", \"0x0d85\",
```

TAGGED SKILLS:

 Fuzzing  Security Research  Reverse Engineering