BLAZE

INFORMATION SECURITY

INTELLIGENT RISK PREVENTION

https://www.blazeinfosec.com

BLAZE

INFORMATION SECURITY

June 4th 2018 • Julio Cesar Fort

IT’S A TRAP!
Remote detection of low & medium interaction honeypots

$ whoami

Julio Fort

- Director of Professional Services at Blaze
- Been around the security community and industry for a while
- Lives in Kraków and quite likes it here

BLAZE

Agenda

Presentation Roadmap

What will be discussed and what to expect

BLAZE

Agenda

- A brief overview of honeypots
- Prior art on the topic of detection
- Why are honeypots prone to detection?
- Case study: detecting a popular SSH honeypot
- Experiment: Internet-wide scanning for SSH honeypots
- Closing remarks

On honeypots

A brief overview of honeypots

Honeypots

- A honeypot *simulates** a real, vulnerable system
- The concept spans back a few decades, probably pioneered by Clifford Stoll (The Cuckoo's Egg)
- Considered an important asset for researchers doing threat intelligence
- Helps to learn more about tools, tactics and procedures of an adversary
- Gaining momentum lately with cyber deception technologies

* actually, sometimes it is a real system

Honeypots – different types

- Can be divided in three broad categories regarding their interaction levels:

- Obviously, each comes with their own advantages and shortcomings

- The focus of this presentation is on the detection of **medium** interaction honeypots

Honeypots

Low **Interaction level**

- Provides basic functionality in an attempt to emulate a service
- Usually catches a specific exploit or a threat pattern (e.g., a certain worm or a particular type of attack)
- As a threat researcher you may probably not get much out of it
- Examples: honeyd, dionaea, spampot

Honeypots

Medium **Interaction level**

- Enhanced interaction
- Pretty popular among the threat research community
- Easy to use, low maintenance and apparently it yields good results
- Examples: Kippo, Cowrie, Glastopf, Conpot

BLAZE

Honeypots

High **Interaction level**

- Fully functioning system with extra instrumentation and logging capabilities

- **Advantages**: can be used to discover unknown vulnerabilities & exploits and are harder to detect
- **Disadvantages**: the compromised host can be used to attack others, high maintenance cost

- Examples: Sebek, custom VM + sensors

PRIOR ART

About previous work on honeypot detection

A word on prior art

- Work on this subject has been done in this past
- Academic work using SVM, etc. – not very practical
- Most stuff revolved around detecting virtual machines (useless these days)
- Sebek was detected and evaded on the fake Phrack 62 and 63 [1]
- Jon Oberhide & Manish Karir wrote a paper on how to detect honeyd [2]

[1] (http://books.gigatux.nl/mirror/honeypot/final/ch09lev1sec2.html)

[2] (https://jon.oberheide.org/blog/2006/02/15/honeyd-remote-fingerprint/)

BLAZE

A word on prior art

- Tod Bearsley of Rapid7 and Andrew Morris did some work detecting Kippo from a pre-authenticated perspective - many of the techniques that were made public on detecting Kippo were patched in Cowrie ☹

- Most detection techniques for SSH honeypots rely on post-auth (within the simulated environment)

- In 2015 Darren Martyn discussed a couple of ways to detect Conpot in “OPSEC for honeypots”

- SHODAN introduced some of the fingerprinting techniques found by Martyn into the search engine.[1]

[1] https://honeyscore.shodan.io

ABOUT DETECTION

Why are many honeypots detection prone?

Honeypots – Proneness to detection

Low **Interaction level**

Medium **Interaction level**

- Nowhere close to fully emulate a real target system
- They have very little or incomplete features
- Or features may behave differently in a real system and in the emulated one

Honeypots – Proneness to detection

High **Interaction level**

- Significantly harder to detect, as it is a real system
- Just realizing that you are inside a VM is so 2002 – no longer a reliable indicator of a honeypot (like discussed in the timeless presentation by GOBBLES' "Wolves among us" at DEFCON 10)
- Still they may be prone to detection, like Sebek
- A live vulnerable system with additional instrumentation and logging can be a good idea, but maintenance is high.[1]

[1] https://doublepulsar.com/eternalpot-lessons-from-building-a-global-nation-state-smb-exploit-honeypot-infrastructure-3f2a0b064ffe

GETTING PRACTICAL

Case study: detecting Cowrie and Kippo

Case study: detecting Cowrie and Kippo

- Their emulated shell provides a not-very-complete bash terminal
- Many commands and common tools are missing too
- It’s **pretty easy** to detect Kippo or Cowrie once you’re inside the shell
- Also not difficult to detect them before falling into the trap

Case study: detecting Cowrie and Kippo

Shell inside a real Linux system

```
julio@whatever2:~$ for((i=0; i<3; i++)); do echo "Is this real?"; done
Is this real?
Is this real?
Is this real?
julio@whatever2:~$
```

Case study: detecting Cowrie and Kippo

Shell inside a Cowrie honeypot

```
root@svr04:~# for((i=0; i<3; i++)); do echo "Is this real?"; done
bash: for: command not found
bash: i: command not found
bash: i++: command not found
bash: do: command not found
bash: done: command not found
root@svr04:~#
```

Honeypot detection

Detecting Kippo and Cowrie

How to detect them remotely without having to fall into the trap?

Pre-authenticated detection strategy

- The basic idea is to understand the behavior of OpenSSH vs. the behavior of Kippo and Cowrie prior to authentication
- Pre-auth includes version advertisement, cipher suite and key exchange
- How does OpenSSH react to a given input – does Kippo/Cowrie handle it in the same way?
- Map the differences (if any) and take note of them
- Bottom-line: no need to waste time and resources brute forcing credentials for a fake system

Some pre-authenticated detection techniques

- **Bad version**: advertising the client as supporting an invalid SSH version (e.g., SSH-31337.0) will make Kippo and Cowrie throw a “bad version” error
- **Double banner**: advertising the client with two banners of causes Cowrie to throw a “Packet corrupt” error.
- **Spacer**: advertising a client with 8 trailing “\n” characters causes Cowrie to throw a “Packet corrupt” error, and Kippo a “Protocol mismatch” response. OpenSSH, on the other hand, works just fine.
- **Fuzzing key exchange**: some corrupted payloads of key exchange, cipher suite and compression algorithms advertisement throw an exception on Kippo and Cowrie, but errors are handled gracefully in OpenSSH

Pre-authenticated detection – Twisted

```python
def _unsupportedVersionReceived(self, remoteVersion):
    """
    Called when an unsupported version of the ssh protocol is received from
    the remote endpoint.

    @param remoteVersion: remote ssh protocol version which is unsupported
        by us.
    @type remoteVersion: L{str}
    """
    self.sendDisconnect(DISCONNECT_PROTOCOL_VERSION_NOT_SUPPORTED,
        b'bad version ' + remoteVersion)


def dataReceived(self, data):
    """
    First, check for the version string (SSH-2.0-*).  After that has been
    received, this method adds data to the buffer, and pulls out any
    packets.

    @type data: L{bytes}
    @param data: The data that was received.
    """
    self.buf = self.buf + data
    if not self.gotVersion:
        if self.buf.find(b'\n', self.buf.find(b'SSH-')) == -1:
            return

        # RFC 4253 section 4.2 ask for strict `\r\n` line ending.
        # Here we are a bit more relaxed and accept implementations ending
        # only in '\n'.
        # https://tools.ietf.org/html/rfc4253#section-4.2
        lines = self.buf.split(b'\n')
        for p in lines:
            if p.startswith(b'SSH-'):
                self.gotVersion = True
                # Since the line was split on '\n' and most of the time
                # it uses '\r\n' we may get an extra '\r'.
                self.otherVersionString = p.rstrip(b'\r')
                remoteVersion = p.split(b'-')[1]
                if remoteVersion not in self.supportedVersions:
                    self._unsupportedVersionReceived(remoteVersion)
                    return
```

Pre-authenticated detection – OpenSSH

```
        chop(server_version_string);
        debug("Local version string %.200s", server_version_string);

        if (remote_major != 2 &&
            !(remote_major == 1 && remote_minor == 99)) {
                s = "Protocol major versions differ.\n";
                (void) atomicio(vwrite, sock_out, s, strlen(s));
                close(sock_in);
                close(sock_out);
                logit("Protocol major versions differ for %s port %d: "
                    "%.200s vs. %.200s",
                    ssh_remote_ipaddr(ssh), ssh_remote_port(ssh),
                    server_version_string, client_version_string);
                cleanup_exit(255);
        }
}
```

BLAZE

Pre-authenticated detection – Twisted KEx

```
self.otherKexInitPayload = chr(MSG_KEXINIT) + packet
# This is useless to us:
# cookie = packet[: 16]
k = getNS(packet[16:], 10)
strings, rest = k[:-1], k[-1]
(kexAlgs, keyAlgs, encCS, encSC, macCS, macSC, compCS, compSC, langCS,
 langSC) = [s.split(b',') for s in strings]
# These are the server directions
outs = [encSC, macSC, compSC]
ins = [encCS, macSC, compCS]
if self.isClient:
    outs, ins = ins, outs # Switch directions
server = (self.supportedKeyExchanges, self.supportedPublicKeys,
        self.supportedCiphers, self.supportedCiphers,
        self.supportedMACs, self.supportedMACs,
        self.supportedCompressions, self.supportedCompressions)
client = (kexAlgs, keyAlgs, outs[0], ins[0], outs[1], ins[1],
        outs[2], ins[2])
if self.isClient:
    server, client = client, server
self.kexAlg = ffs(client[0], server[0])
self.keyAlg = ffs(client[1], server[1])
self.nextEncryptions = SSHCiphers(
    ffs(client[2], server[2]),
    ffs(client[3], server[3]),
    ffs(client[4], server[4]),
    ffs(client[5], server[5]))
self.outgoingCompressionType = ffs(client[6], server[6])
self.incomingCompressionType = ffs(client[7], server[7])
if None in (self.kexAlg, self.keyAlg, self.outgoingCompressionType,
            self.incomingCompressionType):
    self.sendDisconnect(DISCONNECT_KEY_EXCHANGE_FAILED,
                        b"couldn't match all kex parts")
    return
if None in self.nextEncryptions.__dict__.values():
    self.sendDisconnect(DISCONNECT_KEY_EXCHANGE_FAILED,
                        b"couldn't match all kex parts")
    return
```

Pre-authenticated detection example

- **Bad version** example

```
root@research:~# echo -e "SSH-31337\r" | nc honeypot.whatever.io 22 | strings
SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
Protocol mismatch.
```

```
root@research:~# echo -e "SSH-31337\r" | nc honeypot.whatever.io 2022 | strings
SSH-2.0-OpenSSH_5.1p1 Debian-5
bad version 31337
root@research:~# echo -e "SSH-31337\r" | nc honeypot.whatever.io 2222 | strings
SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2
bad version 31337
```

PRE-AUTHENTICATED DETECTION

LIVE DEMO

Pre-authenticated detection strategy

- Some false positives may arise: for example, sometimes incorrectly configured OpenSSH servers can throw “Packet corrupt” errors
- The script uses multiple techniques to detect a SSH honeypot and attributes score for each successful probe
- If score > 2 it can be said with some degree of certainty it is a honeypot

SCANNING

Internet-wide scan to find SSH honeypots

BLAZE

INITIAL OBJECTIVES

- **Map the entire Internet for Kippo and Cowrie honeypots**
- **Find a suitable tool to allow us Internet-wide scanning**
- **Plot them all on a map for visualization purposes**

FINAL OBJECTIVES

BLAZE

Tools of the trade

Using masscan for our goals

A brief explanation on how to modify and use Robert Graham’s masscan to serve our purposes

Masscan

- Masscan has a not so well-known feature to actually send data to a server upon connecting to the port
- It also has a functionality for banner grabbing

- This combination suits our needs!

```
$ echo -e "SSH-HONEY\r\n" | base64
U1NILUhPTkVZDQoK
```

```
./masscan --range 0.0.0.0/0 -p22 --banner -oL ssh_badversion.scan --exclude 255.255.255.255
--rate 850000 --hello-string[22] U1NILUhPTkVZDQoK
```

Modifying Masscan

modify proto-banner1.c from:

```
struct Patterns patterns[] = {
    {"SSH-1.",        6, PROTO_SSH1, SMACK_ANCHOR_BEGIN},
    {"SSH-2.",        6, PROTO_SSH2, SMACK_ANCHOR_BEGIN},
```

to:

```
struct Patterns patterns[] = {
    {"SSH-H.",        6, PROTO_SSH1, SMACK_ANCHOR_BEGIN},
    {"SSH-P.",        6, PROTO_SSH2, SMACK_ANCHOR_BEGIN},
    {"HTTP/1.",       7, PROTO_HTTP, SMACK_ANCHOR_BEGIN},
```

EXPERIMENT – INTERNET-WIDE SSH HONEYPOT SCANNING

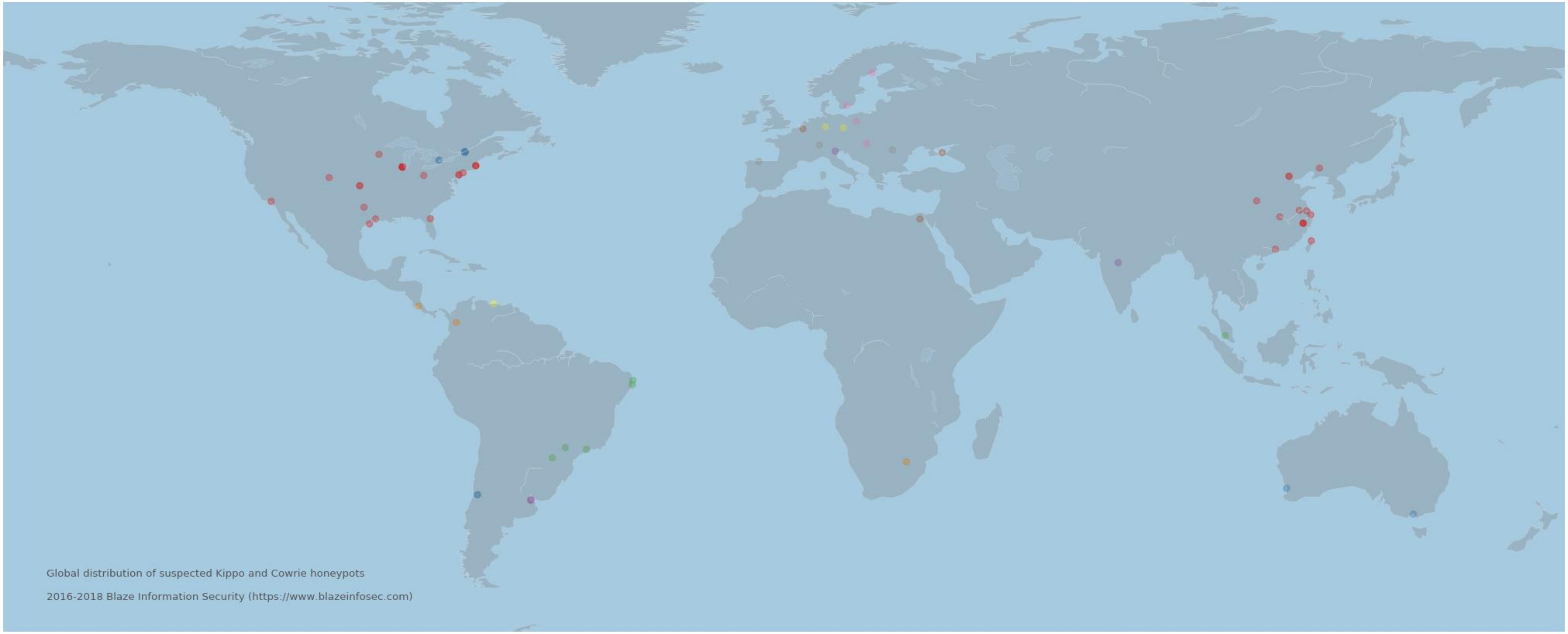
Notes about the experiment

- A cheap VPS will do the job pretty well
- It took ~20 hours to scan the entire Internet for honeypots
- Logs from banner had on average ~900mb
- Be prepared to handle dozens of abuse complaints per day

Mass scanning methodology

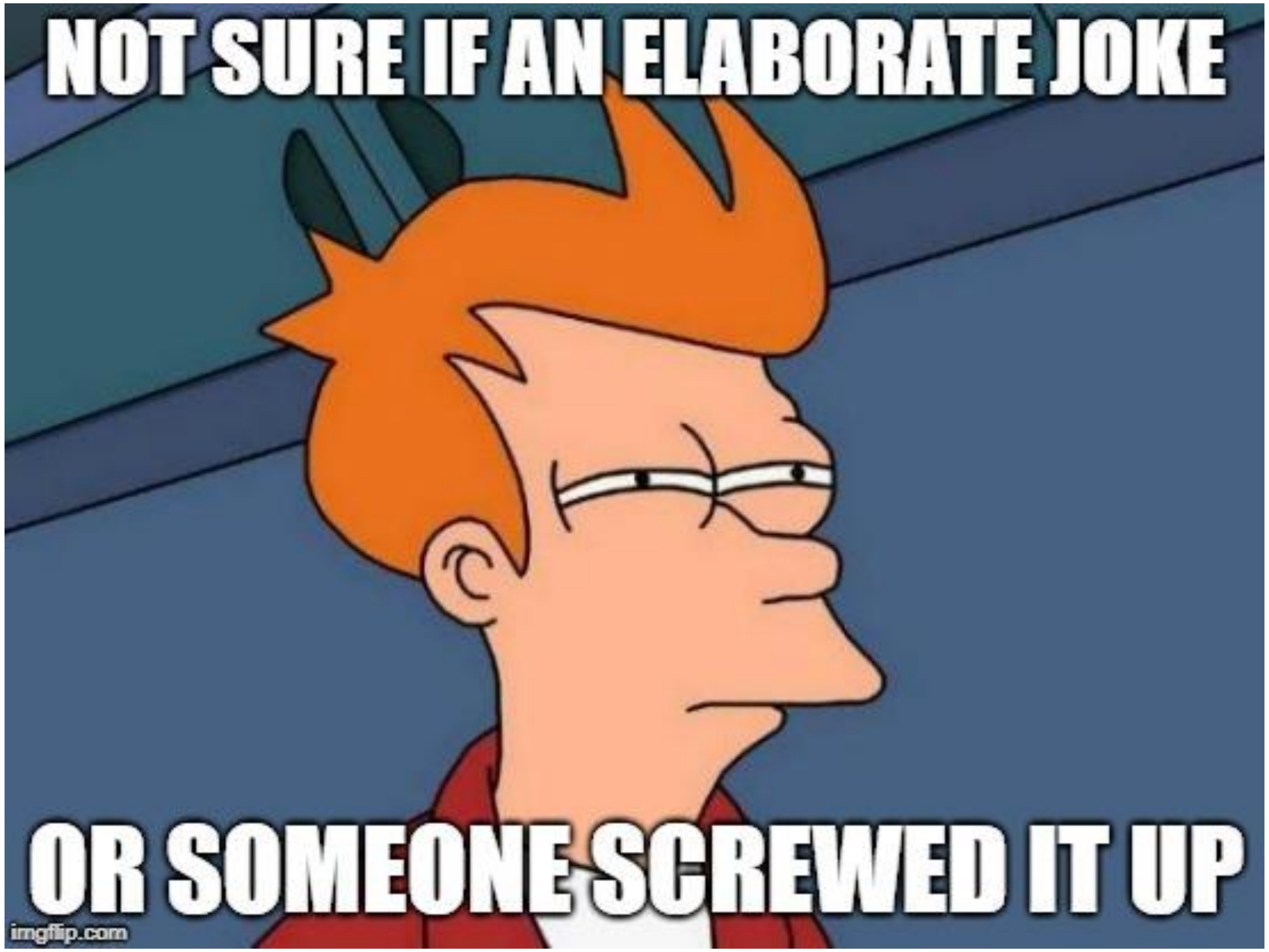
- Masscan ports 22, 2022 and 2222 with the detection strings
- Triage the results using the response patterns we already learned
- Some results are obviously honeypots, others are only suspected (e.g., “Packet corrupt” errors may happen in other situations)
- Feed the suspected IPs into the detection script

Results



Global distribution of suspected Kippo and Cowrie honeypots
2016-2018 Blaze Information Security (https://www.blazeinfosec.com)

Unexpected results



Unexpected results

- While scanning the Internet, a funny SSH banner appeared…

```
julio@whatever2:~/research/ssh-honey/old/mass-scans$ telnet 194.[redacted] 102 22
Trying 194.[redacted].102...
Connected to 194.[redacted].102.
Escape character is '^]'.
SSH-2.0-OpenSSH_5.8p2_hpn13v11 GCHQHONEYPOT34
^]
telnet> q
Connection closed.
```

```
julio@whatever2:~/research/ssh-honey/old/mass-scans$ telnet 194.[redacted].150 22
Trying 194.[redacted].150...
Connected to 194.[redacted].150.
Escape character is '^]'.
SSH-2.0-OpenSSH_5.2p1 gchqhoneypot12
```

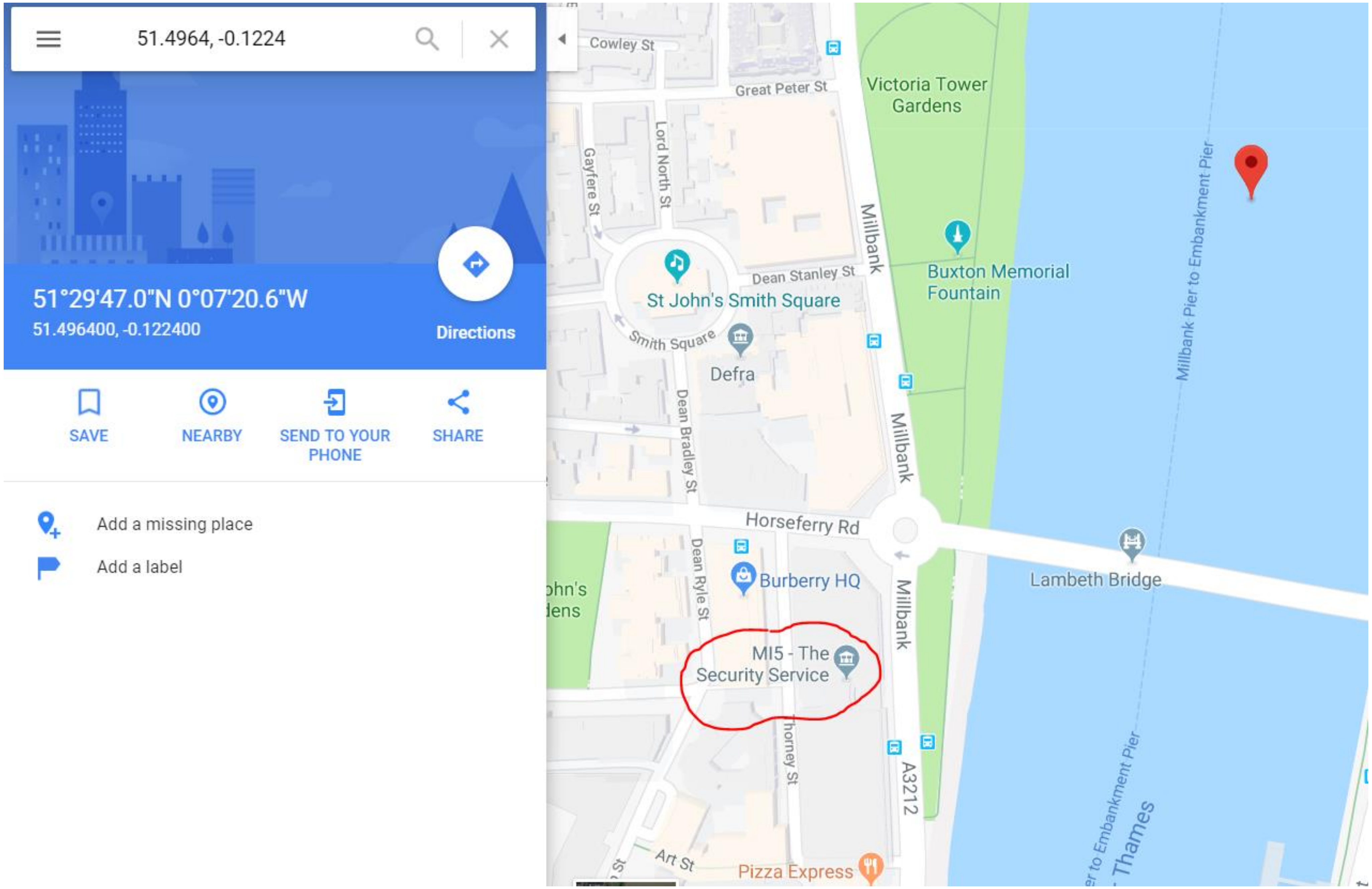
BLAZE

Unexpected results

GeoIP2 City Results

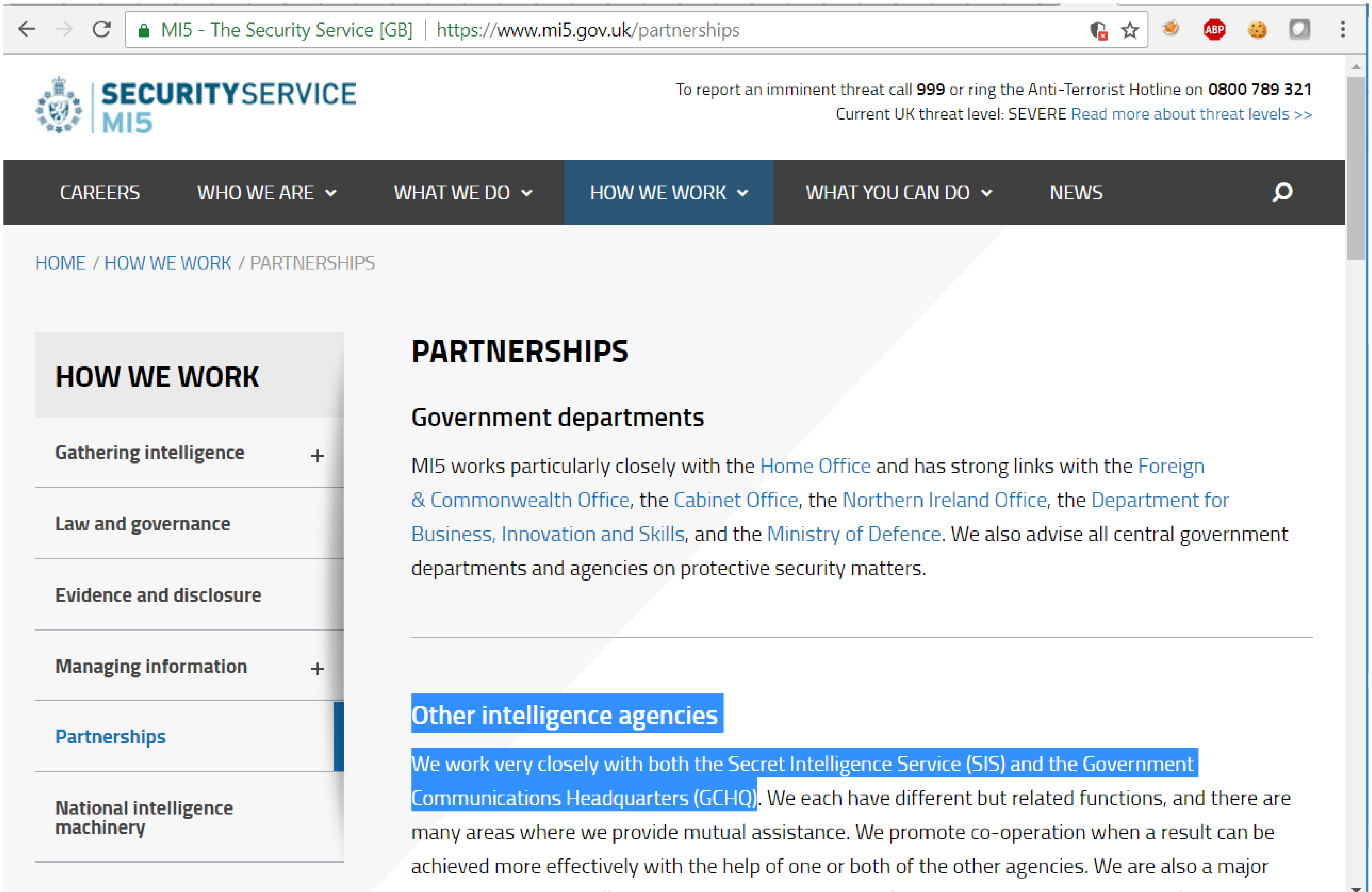
IP Address	Country Code	Location	Postal Code	Approximate Coordinates*	Accuracy Radius	ISP	Organization
194[redacted]102	GB	United Kingdom, Europe		51.4964, -0.1224	200	High Availability Hosting Limited	High Availability Hosting Limited
194.2[redacted] 150	GB	United Kingdom, Europe		51.4964, -0.1224	200	High Availability Hosting Limited	High Availability Hosting Limited

Unexpected results



BLAZE

Unexpected results



BLAZE

CLOSING REMARKS

Closing remarks

- Honeypots are valuable tools for threat analysis and research…
- … however, in many cases it is trivial to detect them
- Finding new detections & patching them will be an eternal cat-and-mouse game, with the attacker always winning
- Solely relying on honeypots for threat research is a flawed assumption
- A high interaction honeypot will certainly yield better results but it comes with a cost

Scripts and supporting material at our repository:

https://github.com/blazeinfosec/detect-kippo-cowrie

Thank you!

Julio Cesar Fort • julio@blazeinfosec.com

https://www.blazeinfosec.com