Network Evasion

Now you see me, now you don't

James Anderson

What we're covering

- Network Evasion
  - Current State
  - Abusing HTTP
  - External Communications
  - Detection and Evasion

James Anderson

- Red Team Consultant at Fireeye
- Previously Work
  - Reverse Engineering
  - Security Engineer
- Big fan of Board Games, D&D and Hack the box challenges.

MANDIANT
A FireEye® Company

Evasion

"Subtle and insubstantial, the expert leaves no trace; divinely mysterious, he is inaudible. Thus he is the master of his enemy's fate."

~Sun Tzu, The Art of War

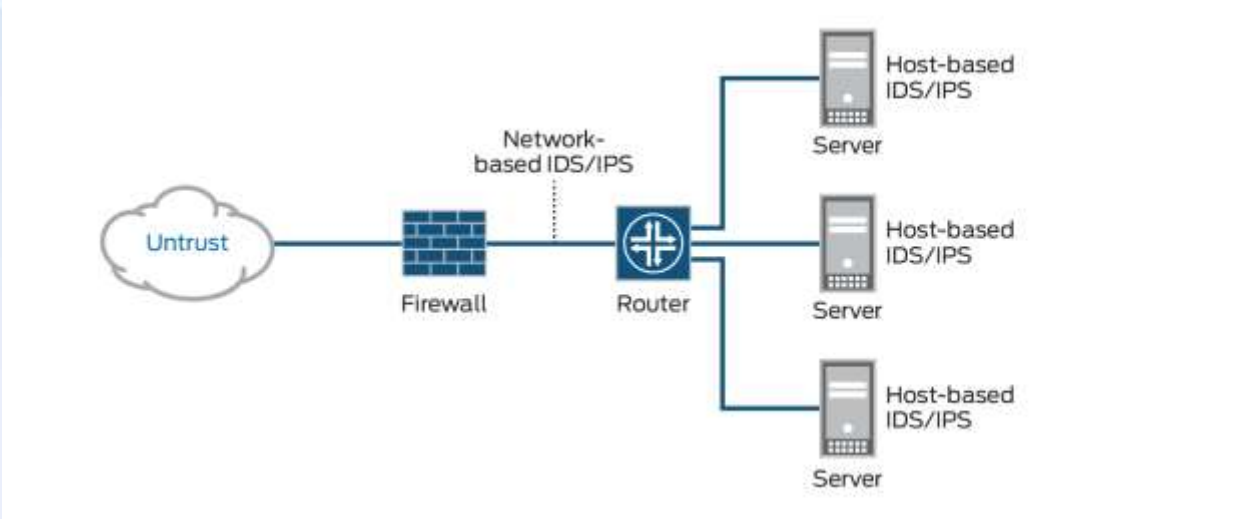
Network Evasion

Network Evasion: bypass an information security device like a firewall or intrusion detection prevention system in order to deliver an exploit, attack or other form of malware to a target network or system without detection.

Network Evasion

- To achieve network evasion a set of techniques, tools practices is used to conceal the true activity to human and automated resources
  - Obfuscation
  - Encryption
  - Stenography

Network Evasion



Source: Juniper Networks

Network Evasion

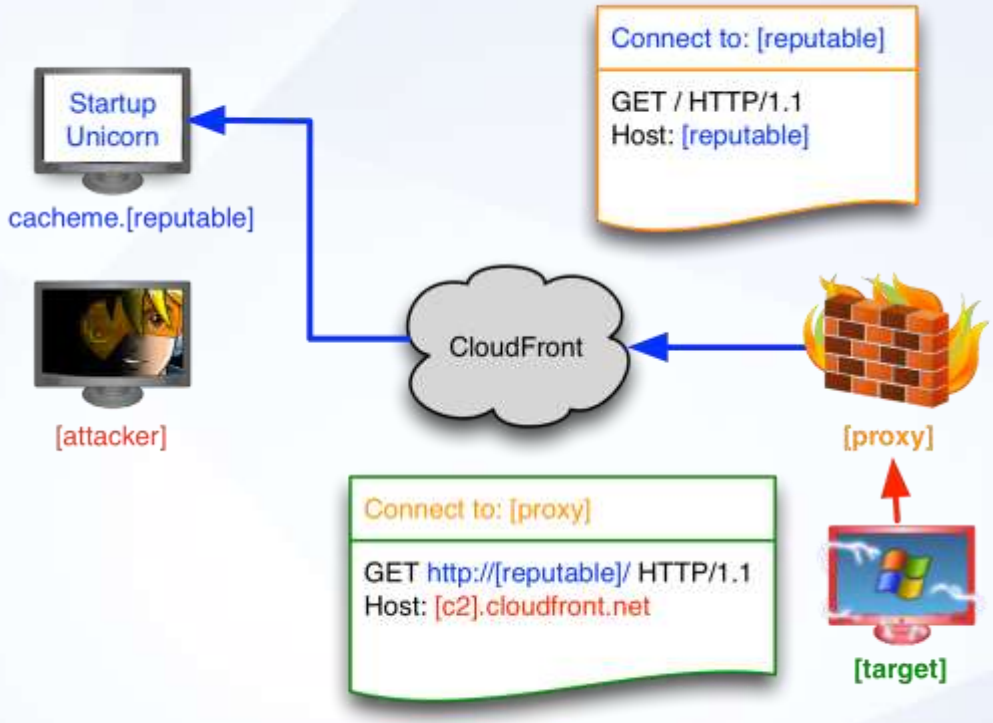
- A World with ubiquitous monitoring
  - Host based EDR products
  - Network Port spanning
  - Machine Learning
  - AMSI – Group Policy Settings
  - NGFW

Network Evasion

- Easy to detect scan traffic (nmap)
  - IDS/IPS can easily to detect
- Org's are getting better at detecting password spraying
- Multiple failed attempts may send you to a shunned portal

Network Evasion

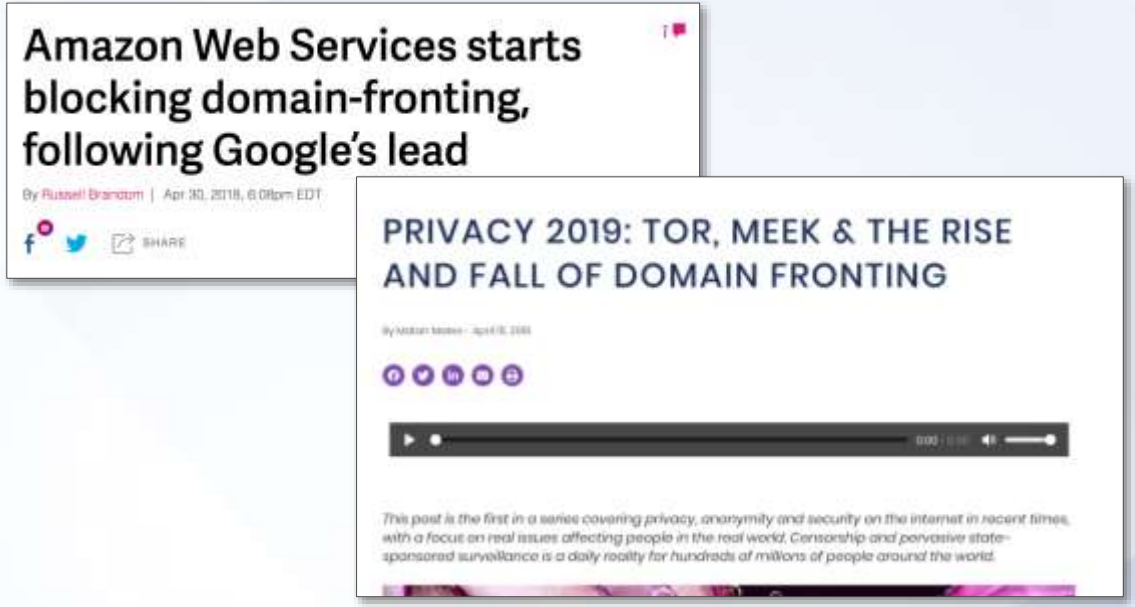
- Domain Fronting
  - Beacons through a high reputation cloud provider
  - Uses the host header that points to a subdomain entry that is a CDN entry to the actual server.



https://cobaltstrike.com

Network Evasion

- Domain Fronting gave us the best method of external communications we could ever ask for.
- While it is still around the window is closing
- Lateral movement techniques are being signature.
- SSL decryption.



Network Evasion

- Some techniques, ideas, OPSEC and resources of how to remain stealthy in a high security environment

Public Platforms

- P2C2 (Public Platform Command and Control)
- CRUD (Create Read Update Destroy)

MANDIANT
A FireEye® Company

SCANNING

Port Scanning

- Three things matter when you are looking at initial scanning
  - Timing between requests
  - Requests per IP
  - Packet Fragmentation
- Slowing you scans down is a must
- Being able to manipulate your source IP's will help

Packet Fragmentation

- Many security devices rely on sessionized data
  - Initiate a connection (handshake)
  - Pass some data
  - Close the connection

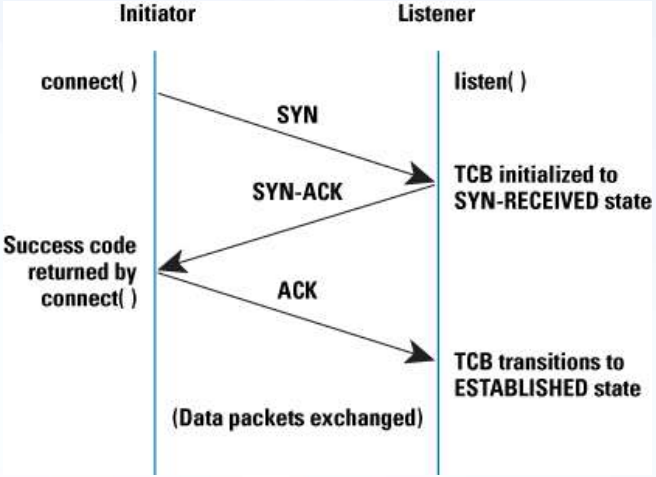
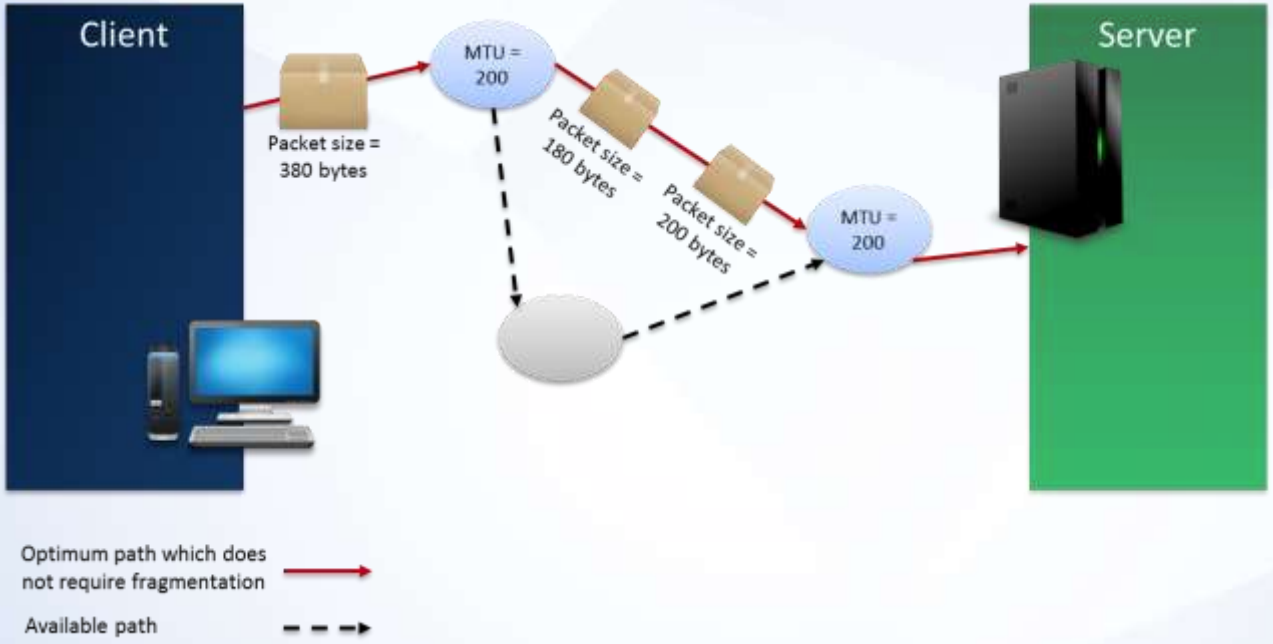


Image source cisco

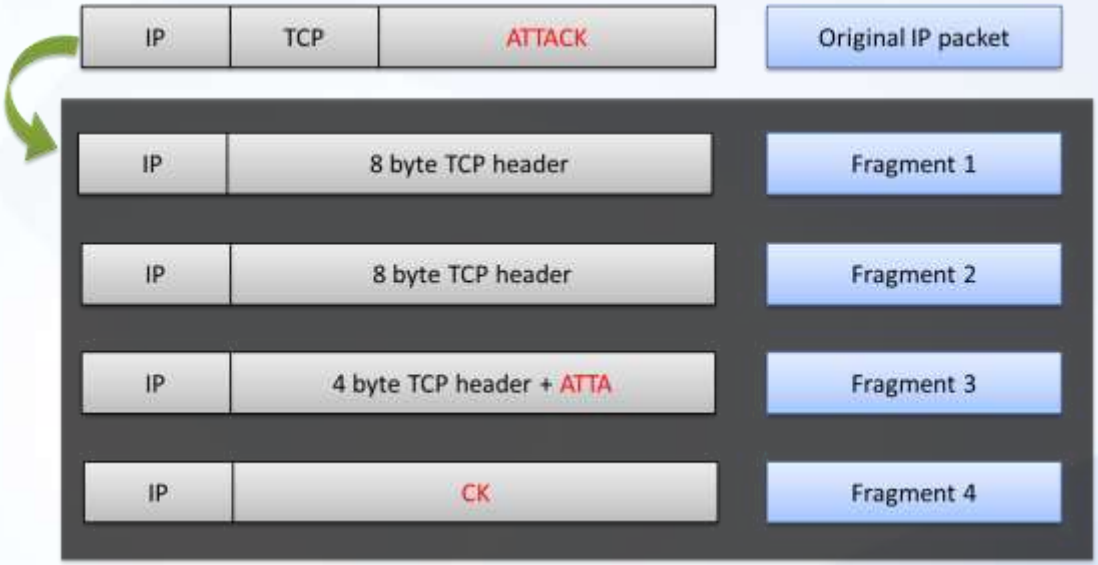
Packet Fragmentation

- The MTU size of a link determines whether there is a need to fragment a datagram into smaller units or not



Packet Fragmentation

- Fragmented Packets to avoid session detection

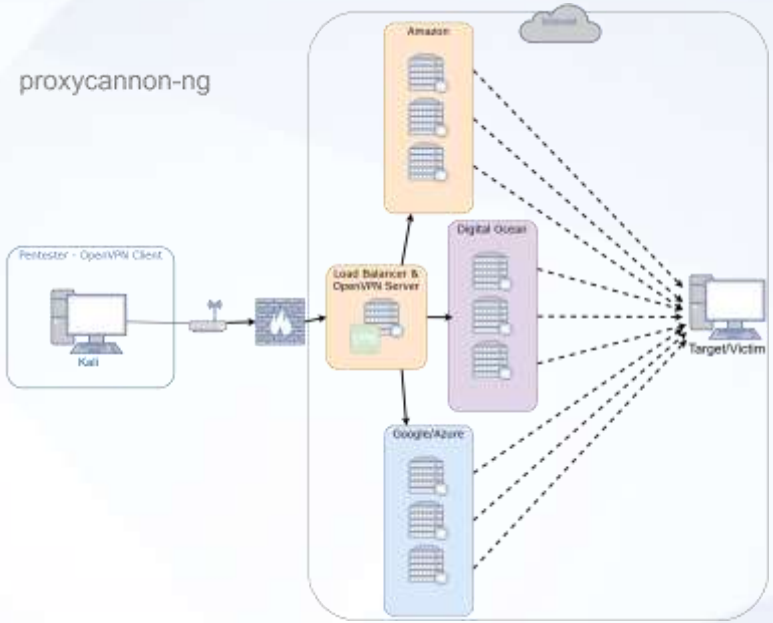


Port Scanning

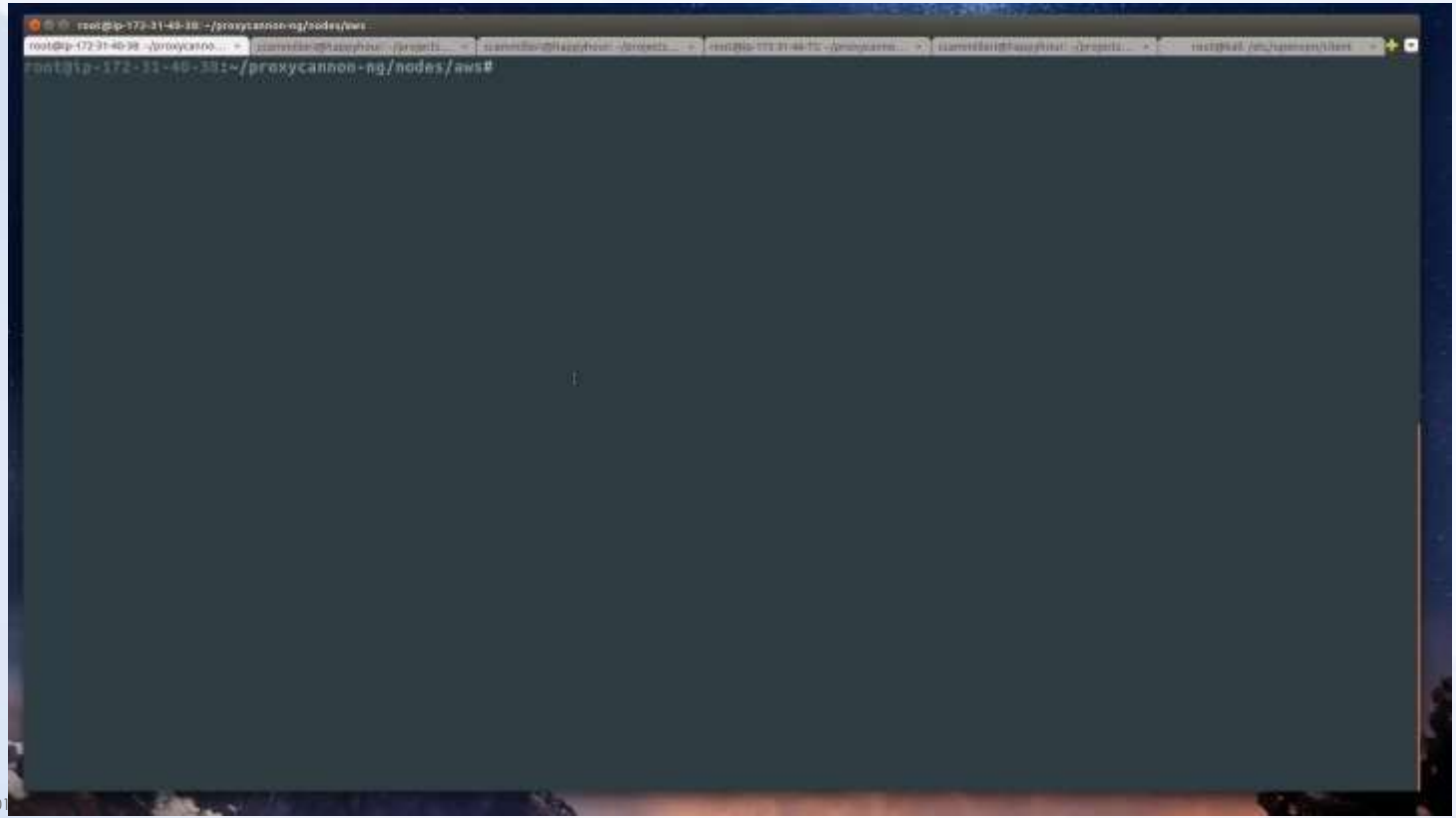
- Nmap
  - Timing options T1: Sneaky(waits 15 seconds) T0: Paranoid (waits 5 minutes)
  - Packet Fragmentation (-f)
  - https://nmap.org.book.man-bypass-firerwalls-rds.html

Port Scanning

- Proxycannon (from ShellIntel) https://github.com.ShellIntel/scripts
  - Can spin up 20 Amazon EC2 instances to proxy scans through
  - Can rotate public WAN IP of nodes



Proxycannon

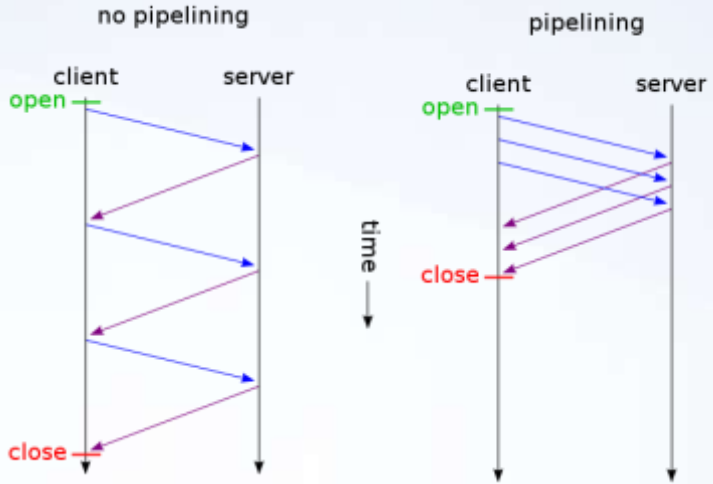


MANDIANT
A FireEye® Company

HTTP PIPELINING

HTTP Pipelining – DigiNinja

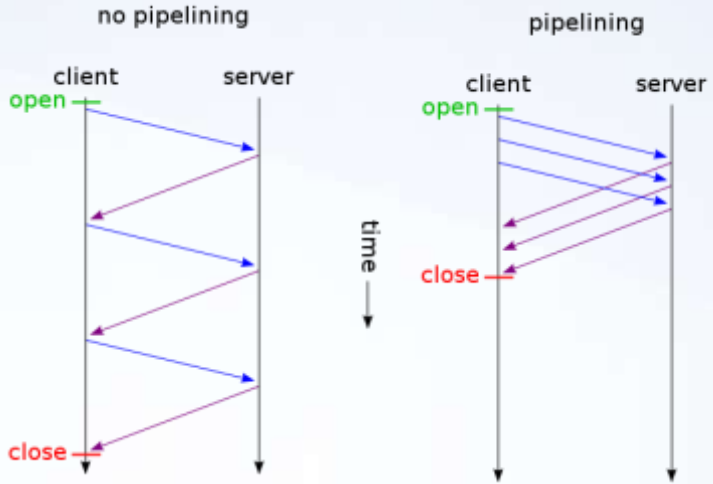
- In the early days each object requested by a client was done in its own TCP connection
- If a page had two images and one JavaScript library, then there would be four connections,
- Pipelining allows for multiple requests at the same time to occur
  - Part of HTTP/1.0, HTTP/1.1



https://digi.ninja/blog/pipelining.php

HTTP Pipelining

- Multiple HTTP requests are sent on a single TCP connection
  - Technique superseded by HTTP2
    - HTTP pipelining is not enabled by default in modern browsers
    - Support still exists in most servers and, more importantly, most CDNs.



https://digi.ninja/blog/pipelining.php

,

HTTP Pipelining

- Lets start with an example request

```
GET /pipeline/page1.php HTTP/1.1
Host: vuln-demo.com
```

```
HTTP/1.1 200 OK
Date: Sat, 14 Sep 2019 03:48:34
GMT
Server: Apache
…
Content-Length: 14


This is page 1
```

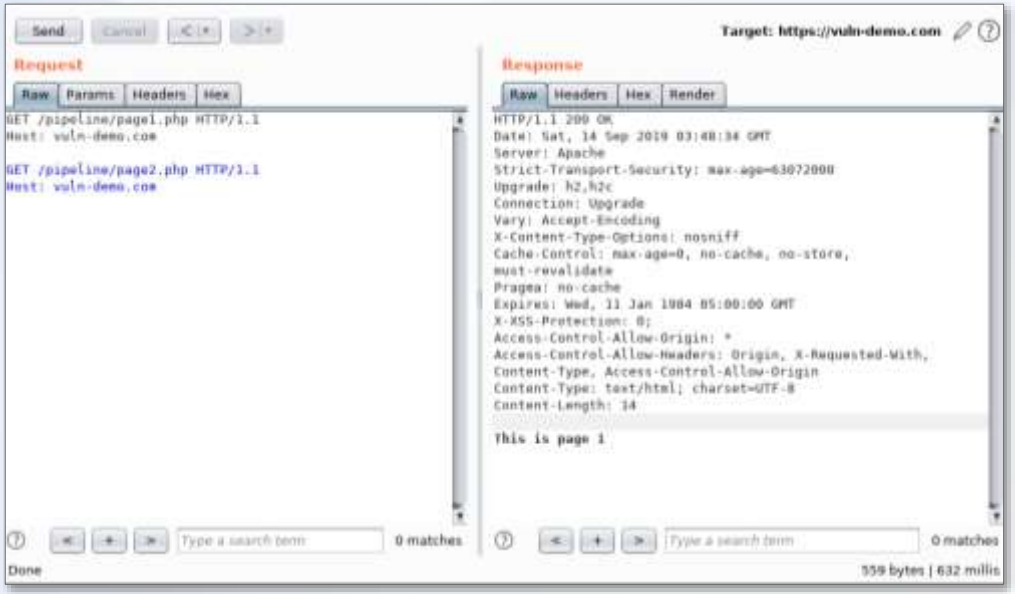
HTTP Pipelining

- Two requests
  - Page1.php
  - Page2.php

```
GET /pipeline/page1.php HTTP/1.1
Host: vuln-demo.com

GET /pipeline/page2.php HTTP/1.1
Host: vuln-demo.com
```

HTTP Pipelining

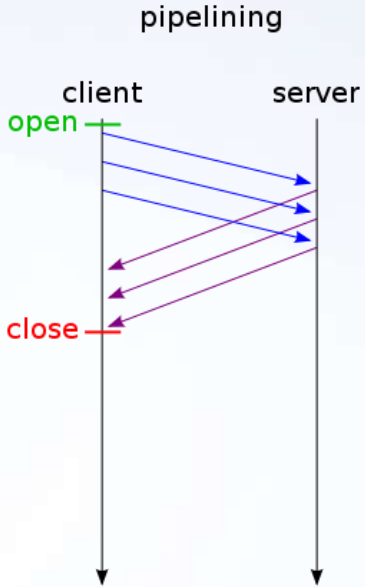
- Burp Doesn't handle this type of request well. Injects content length header



HTTP Pipelining

- These can both be sent, will be processed by the server, and then the responses sent back in order.

```
GET /pipeline/page1.php HTTP/1.1
Host: vuln-demo.com

GET /pipeline/page2.php HTTP/1.1
Host: vuln-demo.com
```



HTTP Pipelining

- Do you require the header: *Connection: keep-alive ?*
- "keep-alive" enables persistent connections which is a different thing to pipelining
  - Persistent connections keep the TCP connection open between requests but enforce the original rule of waiting for any previous requests to return before making new ones
  - In HTTP 1.0, persistence had to be activated with the "keep-alive" header, in HTTP 1.1, persistence is assumed unless a connection is requested to be closed with the Connection: close header.

HTTP Pipelining

- Sending the two requests

```
(echo -e "GET /pipeline/page1.php HTTP/1.1\r\nHost: vuln-demo.com\r\n\r\nGET
/pipeline/page2.php HTTP/1.1\r\nHost: vuln-demo.com\r\n\r\n"; sleep 5) | openssl s_client -
connect vuln-demo.com:443 -servername vuln-demo.com

HTTP/1.1 200 OK
Date: Fri, 08 Mar 2019 20:42:47 GMT
Server: Apache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Access-Control-Allow-Origin: https://vuln-demo.com
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Access-Control-
Allow-Origin Content-Length: 14
Keep-Alive: timeout=5, max=100
Content-Type: text/html; charset=UTF-8

This is page 1HTTP/1.1 200 OK
Date: Fri, 08 Mar 2019 20:42:47 GMT
Server: Apache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Access-Control-Allow-Origin: https://vuln-demo.com
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type,
Access-Control-Allow-Origin
Content-Length: 14
Content-Type: text/html; charset=UTF-8 This is page 2DONE
```

HTTP Pipelining

- Sending the two requests

```
HTTP/1.1 200 OK
Date: Fri, 08 Mar 2019 20:42:47 GMT
Server: Apache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Access-Control-Allow-Origin: https://vuln-demo.com
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Access-Control-
Allow-Origin Content-Length: 14
Keep-Alive: timeout=5, max=100
Content-Type: text/html; charset=UTF-8

This is page 1HTTP/1.1 200 OK
Date: Fri, 08 Mar 2019 20:42:47 GMT
Server: Apache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Access-Control-Allow-Origin: https://vuln-demo.com
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type,
Access-Control-Allow-Origin
Content-Length: 14
Content-Type: text/html; charset=UTF-8 This is page 2DONE
```

Response 1

Response 2

HTTP Pipelining

- Two requests, two responses
  - Pipeline worked!
- Can we use it through a CDN?
  - AWS Cloudfront
    - Actual domain fastpackagedomain.com
    - Fronted Domain. d1sdh26o01490vk5.cloudfront.net

HTTP Pipelining

- Via Cloudfront AWS

```
$ (cat pipe2 ; sleep 5)  |  openssl s_client -connect fronted.fastpackagedomain:443 -
fronted.fastpackagedomain  |  grep "<title>"

depth=2 C = US, O = Amazon, CN = Amazon Root CA 1
verify return:1
depth=1 C = US, O = Amazon, OU = Server CA 1B, CN = Amazon
verify return:1
depth=0 CN = fronted.fastpackagedomain
verify return:1
                    <title>FastPackage - Delivered</title>
                    <title>Fronted Vuln Demo</title>
DONE
```

HTTP Pipelining

- Could be used as a covert channel for communications
- Would require SSL decryption to interpret the payload.
- Some IDS/IPS devices might only parse the first request leaving the second request concealed.

MANDIANT
A FireEye® Company

We need to be QUIC-ER

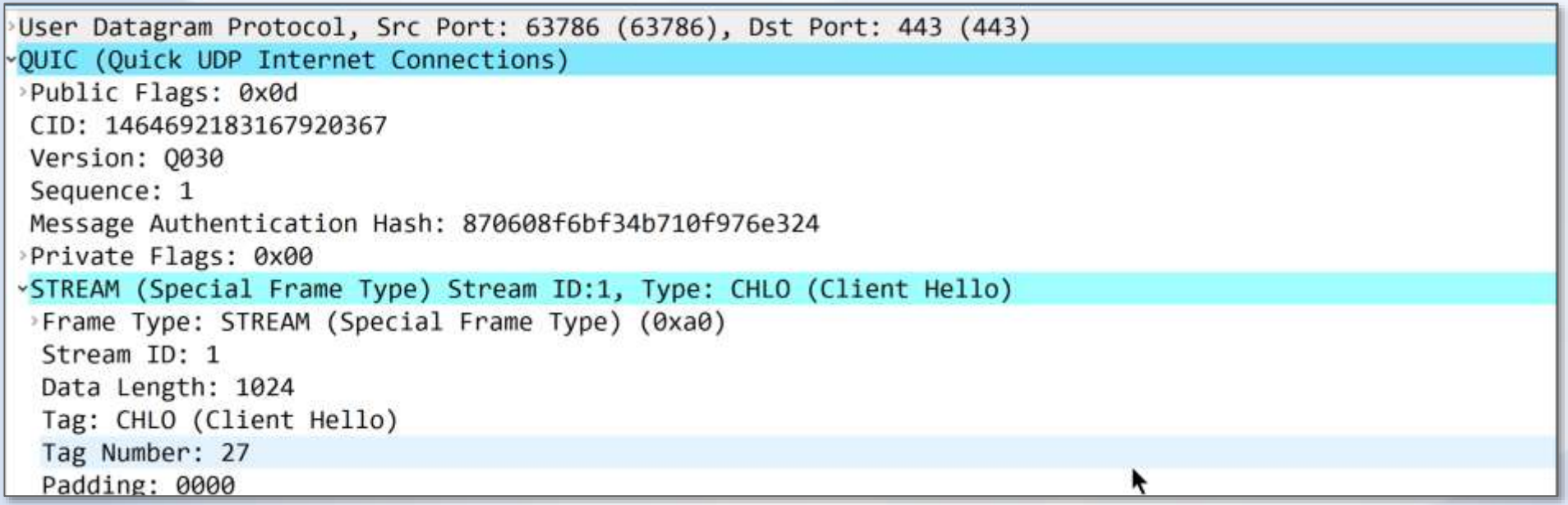
QUIC

- Wait… Firewall was blocking all TCP?

QUIC

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
Frame	100.0	9511	100.0	8361781	540 k	0	0
Ethernet	100.0	9511	100.0	8361781	540 k	0	0
Internet Protocol Version 6	0.1	7	0.0	915	59	0	0
User Datagram Protocol	0.0	1	0.0	153	9	0	0
DHCPv6	0.0	1	0.0	153	9	1	153
Internet Control Message Protocol v6	0.1	6	0.0	762	49	6	762
Internet Protocol Version 4	99.7	9478	100.0	8359708	540 k	0	0
User Datagram Protocol	99.7	9478	100.0	8359708	540 k	0	0
Teredo IPv6 over UDP tunneling	0.1	6	0.0	762	49	0	0
QUIC (Quick UDP Internet Connections)	98.5	9365	99.7	8337858	538 k	9365	8337858
Domain Name System	1.1	107	0.3	21088	1362	107	21088
Address Resolution Protocol	0.3	32	0.0	1920	124	32	1920

QUIC



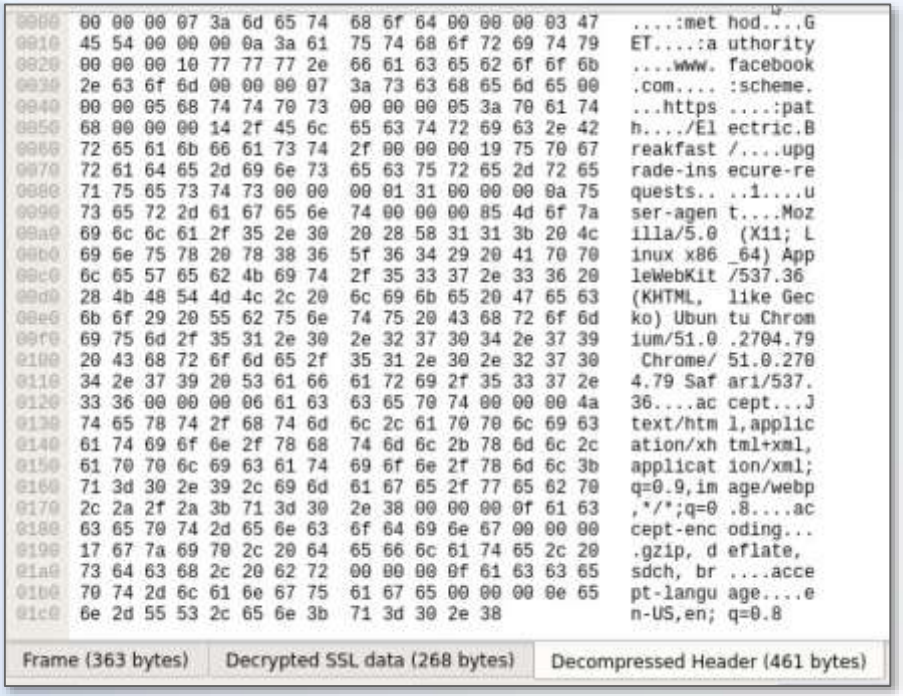
QUIC

- What type of traffic is this?

Wireshark · Follow SSL Stream (tcp.stream eq 78) · wireshark_pcapng_eth0_20160720152239_p5EQBz

```
PRI * HTTP/2.0

SM

.....`.....`...../.%.....A.A.RKRVG...W..yc$./QS....X.?.c..0.....a.<..`"}.....,..o....L.'.}.]y....e..R.+..<.
.....O5$..E....].N6.....Z...z...f.....S...~..j.....!.....2Xm.We.?.....U1..O...;jM.^...kia.....]...=...6.....K....(.3
....S.5#..x,u....V>~.X.....{s..)...c....2.3...=,.P.....RB.@._..!'.R;?Q.-
Kp..Z..@.....@.....d.....4.(.)
v( ........S..._.5#..t...@.I.5S,:(?..a.5_..T..d..Y>..*Cl...q@3pM\.jb..X..~V..."....JTu.`.D....
.i.7a..Y>....m
..q@3pM\.jb..@.....Y&*..;f..
.;.G...k.....Z[..V.H.....=.y.....8yi.....zz.0.8)..W\z/~C..?.....JFIF.....Photoshop 3.0.8BIM.....g..Gi
(.bFBMD01000ac103000028050000430700000ed0700009b080000da0a00007e0d0000df0d0000850e00002c0f00003d130000....ICC_PROFILE.....lcms....mnt
9acspAPPL.....-lcms.....
desc.....^cprt...\....wtpt...h....bkpt...|....rXYZ.....gXYZ.....bXYZ.....rTRC.....@gTRC.....@bTRC.....@de..
{.....sc.....c2.....text....FB..XYZ .....-X
.....o...8.....XYZ .....b.....XYZ .....$.....curv.....c...k...?.Q.4!.).2.;.F.Qw].kpz....|.i.}...0.....C.....
.
```

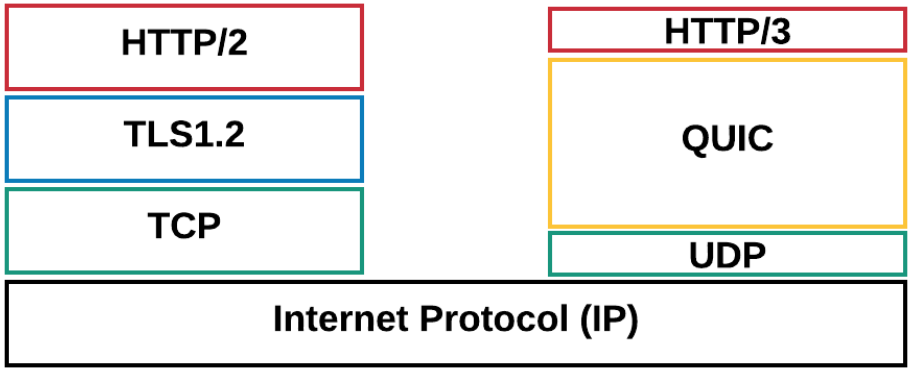
QUIC



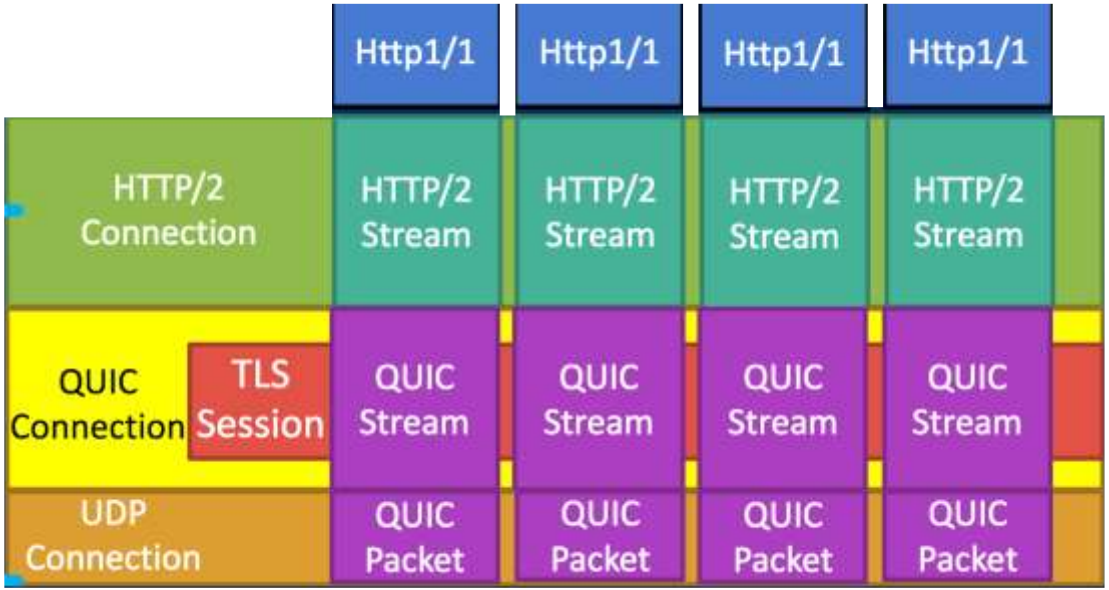
- What's going on here?

QUIC HTTP/3

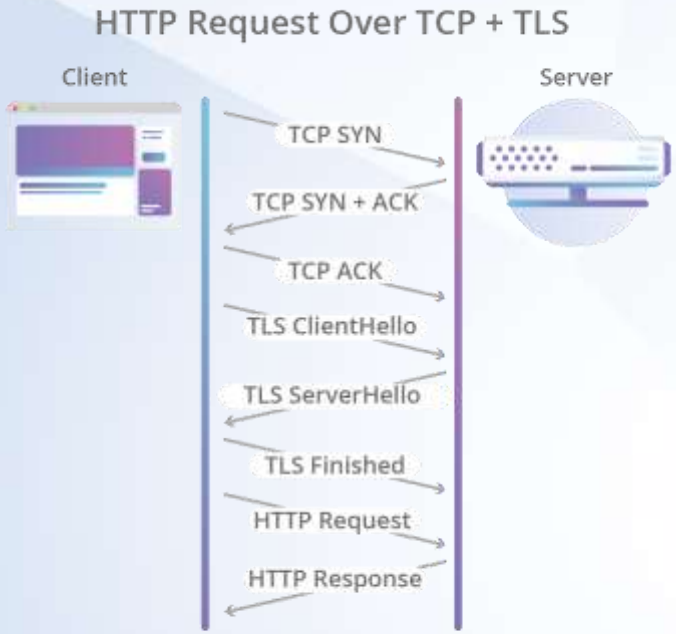
- Quick UDP Internet Connections (QUIC) is a new protocol created by Google to make the web faster and more efficient
  - Enabled by default in Chromium and used by a growth list of sites

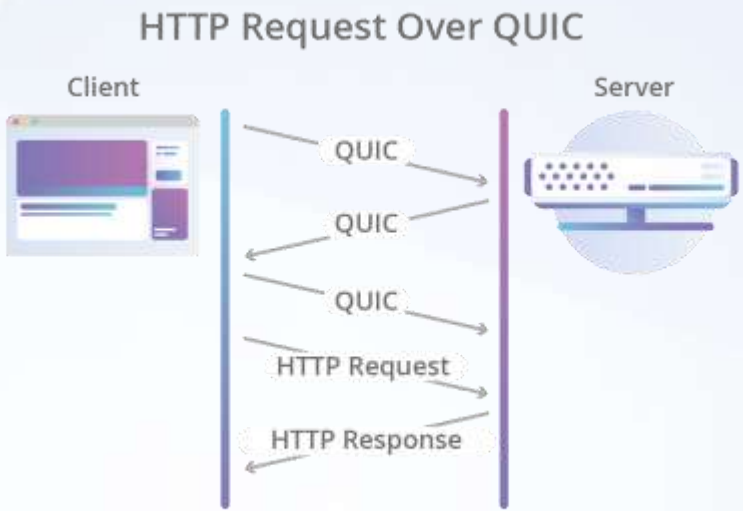


QUIC HTTP/3



QUIC HTTP/3





- Source: cloudflare

QUIC HTTP/3

- Unlike the TCP protocol, QUIC requires 0-RTT in the handshake compared to 1-3 roundtrip TCP + TLS trips
- This ensures security for anyone using the protocol
- Invalidates the possibility of a man-in-the-middle attack
  - A lot of inspection mechanisms don't support QUIC Protocol.

QUIC HTTP/3

- Quick Example of using QUIC
  - Most stable version of QUIC is written in go by **Lucas Clemente**
  - https://github.com/lucas-clemente/quic-go

QUIC HTTP/3 - SERVER

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "time"
    "net/http"
    "github.com/lucas-clemente/quic-go/h2quic"
    "github.com/lucas-clemente/quic-go/internal/protocol"
    quic "github.com/lucas-clemente/quic-go"
)


type Page struct {
    Title string
    Body []byte
}

func (p *Page) save() error {
    filename := p.Title + ".txt"
    return ioutil.WriteFile(filename, p.Body, 0600)
```

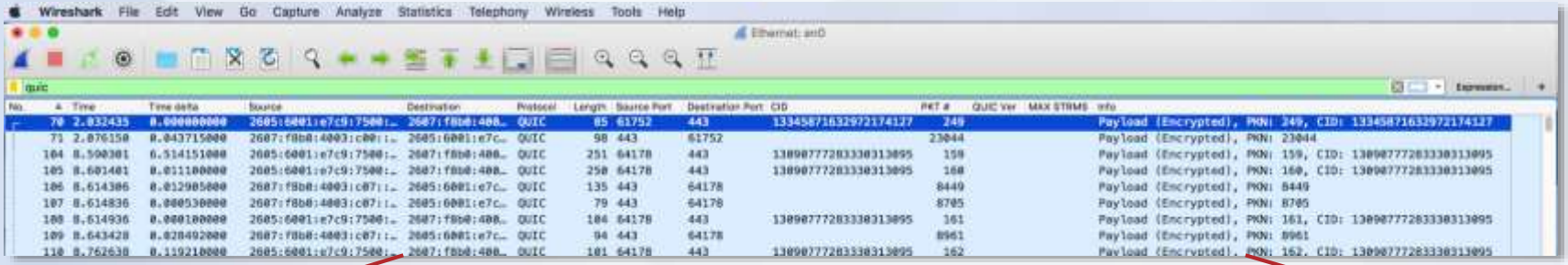
```go
func loadPage(title string) (*Page, error) {
    filename := title + ".txt"
    body, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }
    return &Page{Title: title, Body: body}, nil
}

func viewHandler(w http.ResponseWriter, r *http.Request) {
    title := r.URL.Path[len("/view/"):]
    p, _ := loadPage(title)
    fmt.Fprintf(w, "<h1>%s</h1><div>%s</div>", p.Title, p.Body)
}

func main() {
versions := protocol.SupportedVersions
http.HandleFunc("/view/", viewHandler)
    server := h2quic.Server{
        Server: &http.Server{Addr: ":443"},
QuicConfig: &quic.Config{Versions: versions, IdleTimeout: 30000 * time.Millisecond},
}
    log.Fatal(server.ListenAndServeTLS("fullchain.pem", "privkey.pem"))
}
```

QUIC HTTP/3 - CLIENT

```go
package main

import (
    "bytes"
    "flag"
    "fmt"
    "io"
    "net/http"
    "time"

    quic "github.com/lucas-clemente/quic-go"
    "github.com/lucas-clemente/quic-go/h2quic"
    "github.com/lucas-clemente/quic-go/internal/protocol"
)
```

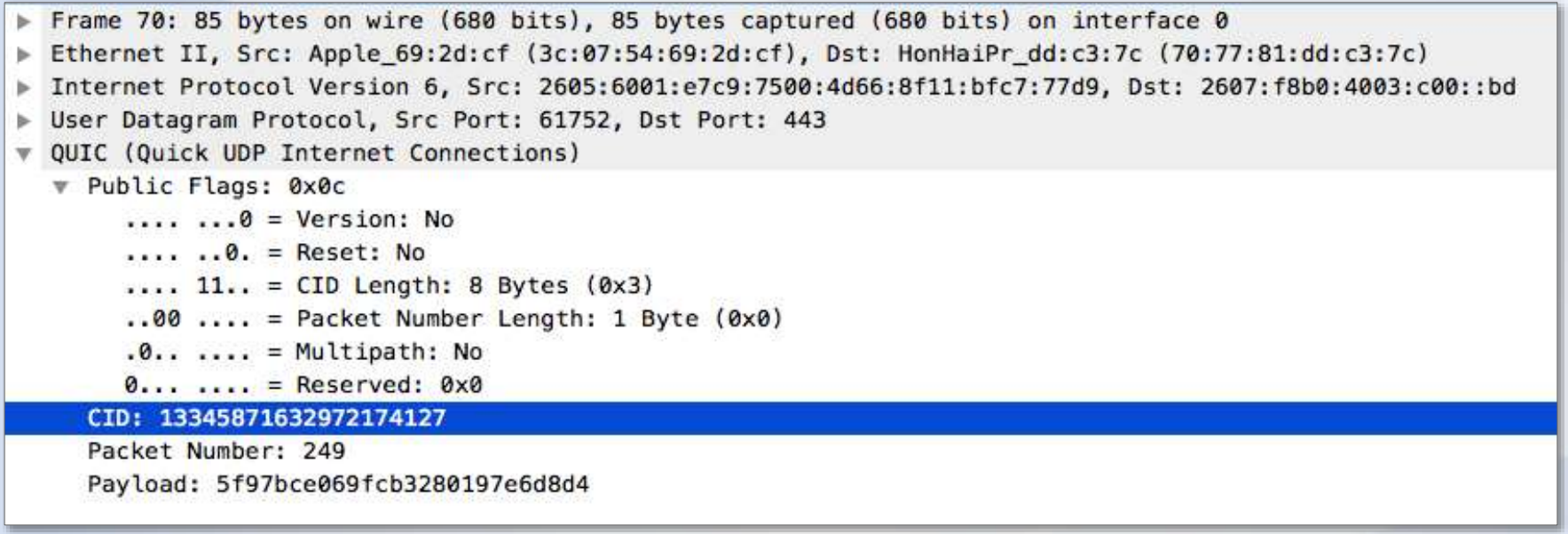
```go
func main() {
    urls := flag.String("url", "https://127.0.0.1:443/", "URL")
    flag.Parse()

    versions := protocol.SupportedVersions
    roundTripper := &h2quic.RoundTripper{
        QuicConfig: &quic.Config{Versions: versions, IdleTimeout: 30000 * time.Millisecond},
    }
    defer roundTripper.Close()
    hclient := &http.Client{
        Transport: roundTripper,
    }

    rsp, err := hclient.Get(*urls)
    rsp.Header.Add("User-Agent", "UnkL4b")
    if err != nil {
        panic(err)
    }

    body := &bytes.Buffer{}
    _, err = io.Copy(body, rsp.Body)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s", body.Bytes())
}
```

QUIC HTTP/3



Destination	Protocol	Length	Source Port	Destination Port	CID	PKT #	QUIC Ver	MAX STRMS	Info
2607:f8b0:400…	QUIC	85	61752	443	13345871632972174127	249			Payload (Encrypted),
2605:6001:e7c…	QUIC	98	443	61752		23044			Payload (Encrypted),
2607:f8b0:400…	QUIC	251	64178	443	13090777283330313095	159			Payload (Encrypted),
2607:f8b0:400…	QUIC	250	64178	443	13090777283330313095	160			Payload (Encrypted),
2605:6001:e7c…	QUIC	135	443	64178		8449			Payload (Encrypted),
2605:6001:e7c…	QUIC	79	443	64178		8705			Payload (Encrypted),
2607:f8b0:400…	QUIC	104	64178	443	13090777283330313095	161			Payload (Encrypted),
2605:6001:e7c…	QUIC	94	443	64178		8961			Payload (Encrypted),
2607:f8b0:400…	QUIC	101	64178	443	13090777283330313095	162			Payload (Encrypted),

QUIC HTTP/3

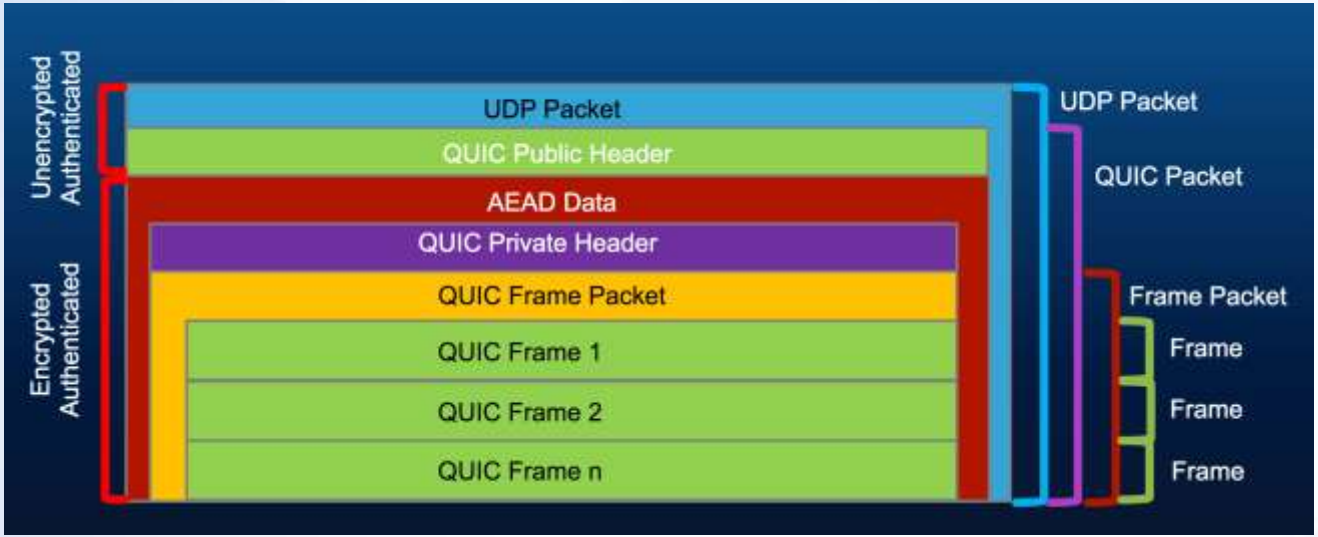


QUIC HTTP/3

- Because the QUIC transport stream does not allow Firewall to perform a deep packet inspection, there is an impact in both reporting and network security that allows attackers to abuse the protocol and avoid detection of malicious actions just changing the **version** in the public header.

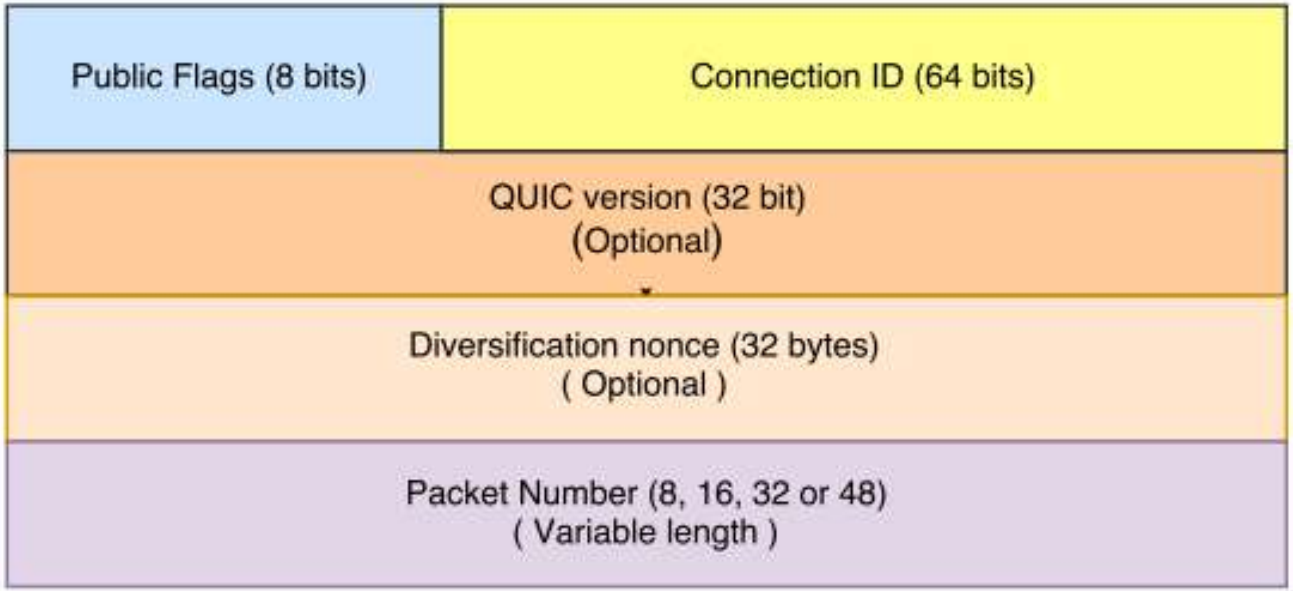
QUIC HTTP/3

- QUIC Structure



QUIC HTTP/3

- QUIC Public Header



QUIC HTTP/3

- QUIC Version
  - The QUIC specification reserves from 0x00000001 to 0x0000ffff for standardized versions of the protocol
  - How some IDS interpret a connection with the QUIC protocol
  - What If we change the header in our comms

Version	Owner	Notes
0x00000000	n/a	This value is reserved as invalid
0x?a?a?a?a	IETF	Values meeting this pattern ((x&0x0f0f0f0f)==0x0a0a0a0a) are reserved for ensuring that version negotiation remains viable.
0x50435130	Private Octopus	Picoquic internal test version
0x5130303[1-9]	Google	Google QUIC 01 - 09 (Q001 - Q009)
0x5130313[0-9]	Google	Google QUIC 10 - 19 (Q010 - Q019)
0x5130323[0-9]	Google	Google QUIC 20 - 29 (Q020 - Q029)
0x5130333[0-9]	Google	Google QUIC 30 - 39 (Q030 - Q039)
0x5130343[0-9]	Google	Google QUIC 40 - 49 (Q040 - Q049)
0x51474f[0-255]	quic-go	“QGO” + [0-255]
0x91c170[0-255]	quicly	“qicly0” + [0-255]
0xabcd000[0-f]	Microsoft	WinQuic
0xf10000[00-ff]	IETF	QUIC-LB
0xf123f0c[0-f]	Mozilla	MozQuic
0xfaceb00[0-f]	Facebook	mvfst
0xff000001	IETF	draft-ietf-quic-transport-01
0xff000002	IETF	draft-ietf-quic-transport-02
0xff000003	IETF	draft-ietf-quic-transport-03
0xff000004	IETF	draft-ietf-quic-transport-04
0xff000005	IETF	draft-ietf-quic-transport-05
0xff000006	IETF	draft-ietf-quic-transport-06
0xff000007	IETF	draft-ietf-quic-transport-07
0xff000008	IETF	draft-ietf-quic-transport-08
0xff000009	IETF	draft-ietf-quic-transport-09
0xff00000a	IETF	draft-ietf-quic-transport-10
0xff00000b	IETF	draft-ietf-quic-transport-11
0xf0f0f0f[0-f]	ETH Zürich	Measurability experiments

QUIC HTTP/3

- **UnkL4b - unkl4b.github.io**
- The tests consist of blocking the QUIC protocol in Fortinet AppControl and running the client to close communication with a server in the cloud that is accepting only the protocol in the Q309 version

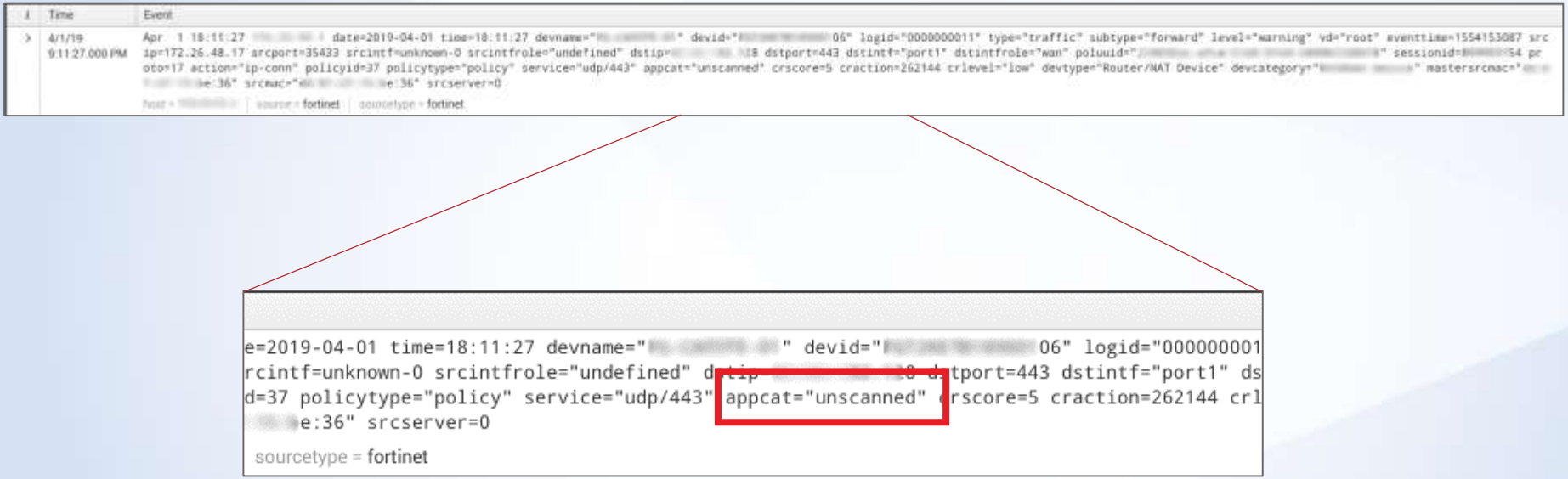
0x5130333[0-9]	Google	Google QUIC 30 - 39 (Q030 - Q039)

QUIC HTTP/3

- Fortigate logs in Splunk
- Blocked when identified as QUIC

QUIC HTTP/3

- Passed with the header version change



QUIC HTTP/3

- New/Unique Protocols are harder for IDS/IPS to track
- For those that to have signatures, changing the public header can effectively mask it again.

HTTP2

- Merlin is a cross-platform post-exploitation HTTP/2 Command & Control server
  - https://github.com/Ne0nd0g/merlin

MANDIANT
A FireEye® Company

Traffic Normalization

IDS/IPS and Machine Learning

Normalizing Traffic

- Signature based NIDS
  - Look for pre-defined patters of previously known attacks
  - Doesn't require training phase
  - Highly available and popular
  - Can't catch new attacks

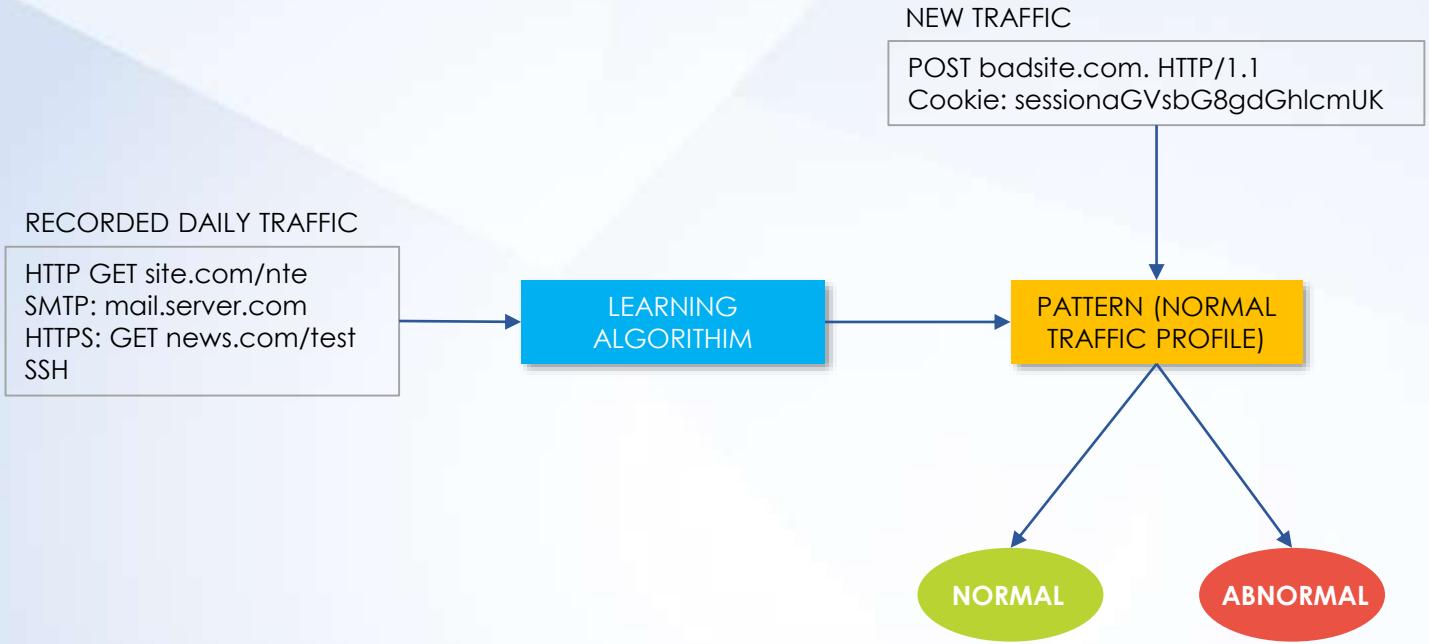
Normalizing Traffic

- Bypassing this form of signature
  - Not hard but not super easy
  - Change traffic elements
  - Don’t match with any signatures

Anomaly based NIDS

- Build a statistical model describing the normal network traffic and flagging the abnormal traffic
- Requires training phase
- Uses math, machine learning and some more sophisticated methods
- Might catch on previously unseen activity.

Anomaly based NIDS

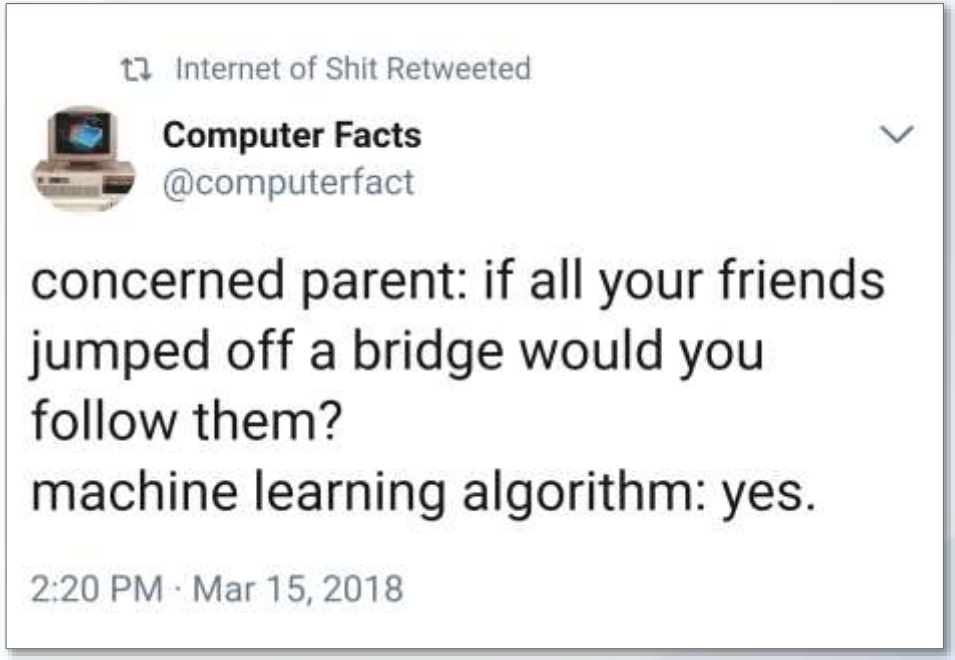


Anomaly based NIDS

- Evasion
  - Pre-Training
  - Post-Training

Anomaly based NIDS

- Pre Training
  - Sending Malicious requests to the system.



Machine Learning

- DARKTRACE
- CISCO

Machine Learning

- Machine Learning Algorithms
  - **Supervised machine learning algorithms**
    - Can apply what has been learned in the past to predict future events using labelled examples.
  - **Unsupervised machine learning algorithms:**
    - Used when the information used to train is neither marked nor classified.
  - **Semi-supervised machine learning algorithms:**
    - Makes use of unlabelled data for training — with a blend of less labelled data and a lot of unlabelled data.

Machine Learning

– Basic features of individual TCP connections

Feature Name	Description	Type
duration	length (number of seconds) of the connection	continuous
protocol type	type of the protocol (tcp, udp)	discrete
service	network service of the destination. (http, telnet ssh..)	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host port; 0 otherwise	discrete
wrong fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Machine Learning

- Common Machine Learning Algorithms
  - K means clustering
  - Bayes Network
  - Random Forest Classifier
  - Multi-Layer Perceptron (MLP)

Machine Learning

- Machine Learning Algorithms

$$Accuracy_{ave} = \frac{TP + TN}{TP + FN + FP + TN}$$

  - No model is perfect
  - We want to sit in the slim but possible False negative category

Machine Learning

- KDD Cup Data
  - Labelled as normal or as an attack
  - 4898431 instances with 41 attributes

Table VI. **Average Accuracy Rate**

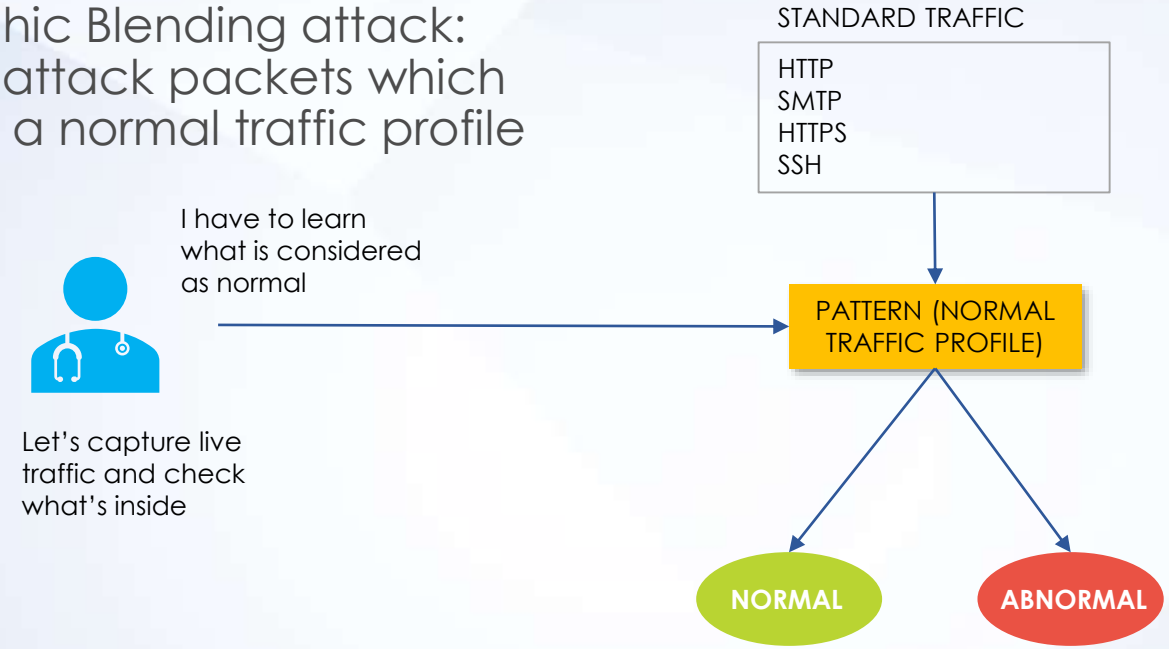
Machine Learning Classifiers	Correctly classified Instances	Incorrectly classified Instances	Accuracy Rate
j48	55865	4135	93.10%
Random Forest	56265	3735	93.77%
Random tree	55345	5655	90.57%
Decision table	55464	4536	92.44%
MLP	55141	4859	91.90%
Naive Bayes	54741	5259	91.23%
Bayes Network	54439	5561	90.73%

Machine Learning

- How do we evade detection?
  - Normalize your behaviour so you look like everything else
  - Avoid similar methods of connection that ML would use as a characteristic.

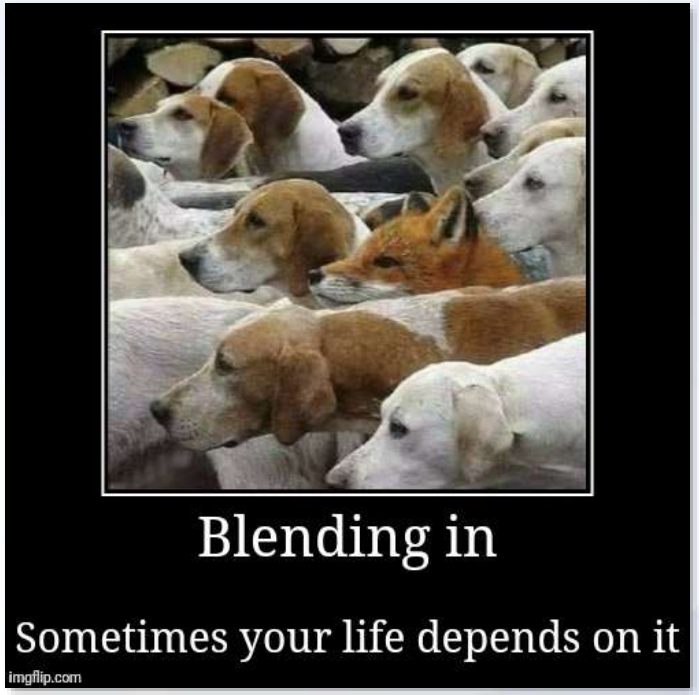
Polymorphic Blanding Attack

- Polymorphic Blending attack: Creating attack packets which match to a normal traffic profile



Blending

- Get traffic capture data of traffic and define normal behaviour of users
  - Which User agents are most common
  - Which ports are used, what kind of server headers are there?
- Alter comms channel to reflect the same pattern
- What hosts is this machine communicating
- https://github.com/tearsecurity/firstorder



Blending

=== Top 10 Port Statistics ===

Port 443: 1677/5937 (28.25%)
Port 58471: 1107/5937 (18.65%)
Port 80: 536/5937 (9.03%)
Port 58457: 454/5937 (7.65%)
Port 54674: 341/5937 (5.74%)
Port 57859: 228/5937 (3.84%)
Port 54119: 157/5937 (2.64%)
Port 58408: 155/5937 (2.61%)
Port 53: 124/5937 (2.09%)
Port 58403: 80/5937 (1.35%)

=== Top 10 Server Headers ===

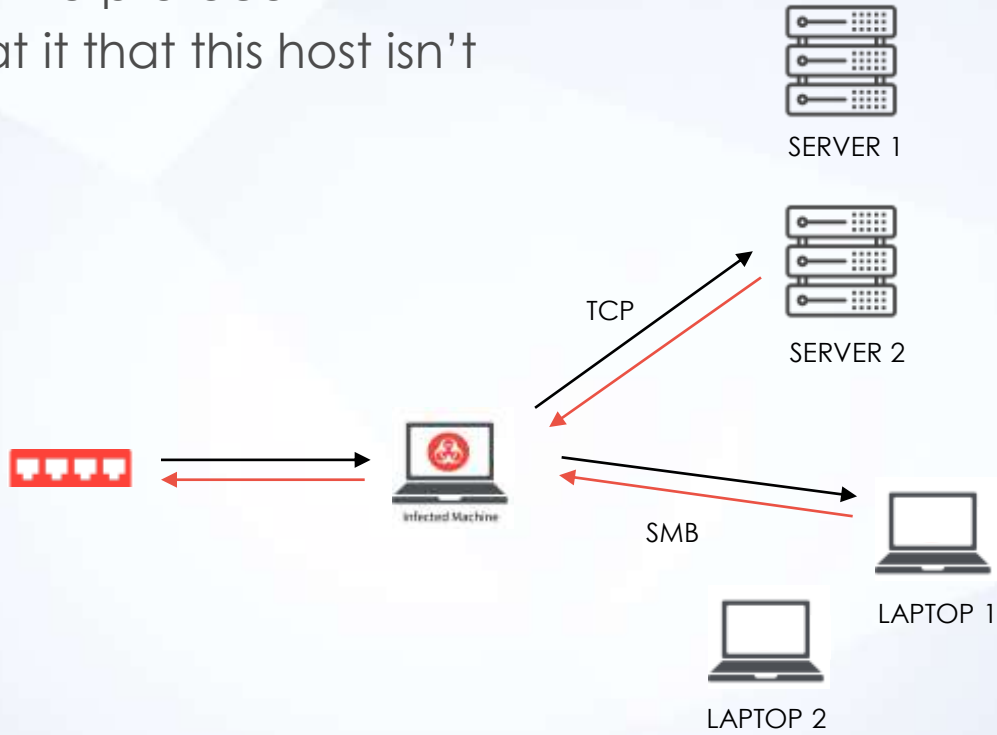
Server: PWS/8.3.1.0.4: 9/36 (25.00%)
Server: RocketCache/2.2: 5/36 (13.89%)
Server: nginx: 5/36 (13.89%)
Server: NetDNA-cache/2.2: 4/36 (11.11%)
Server: None: 3/36 (8.33%)
Server: nginx/1.8.1: 2/36 (5.56%)
Server: cafe: 1/36 (2.78%)
Server: Microsoft-IIS/7.5: 1/36 (2.78%)
Server: cloudflare-nginx: 1/36 (2.78%)
Server: Microsoft-IIS/10.0: 1/36 (2.78%)

=== Top 10 User-Agent Headers ===

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36: 29/32 (90.62%)
User-Agent: Google Chrome/63.0.3239.84 Mac OS X: 3/32 (9.38%)

Blending

- Talk to hosts on the same protocol
- Don't talk to hosts that it that this host isn't talking to



Machine Learning

- How do we evade detection?
  - Normalize your behaviour so you look like everything else
  - Don't use the same methods of connection that ML would use as a characteristic.

MANDIANT
A FireEye® Company

The quieter you are the more you hear

Listen for the packets you want

Listening for packets

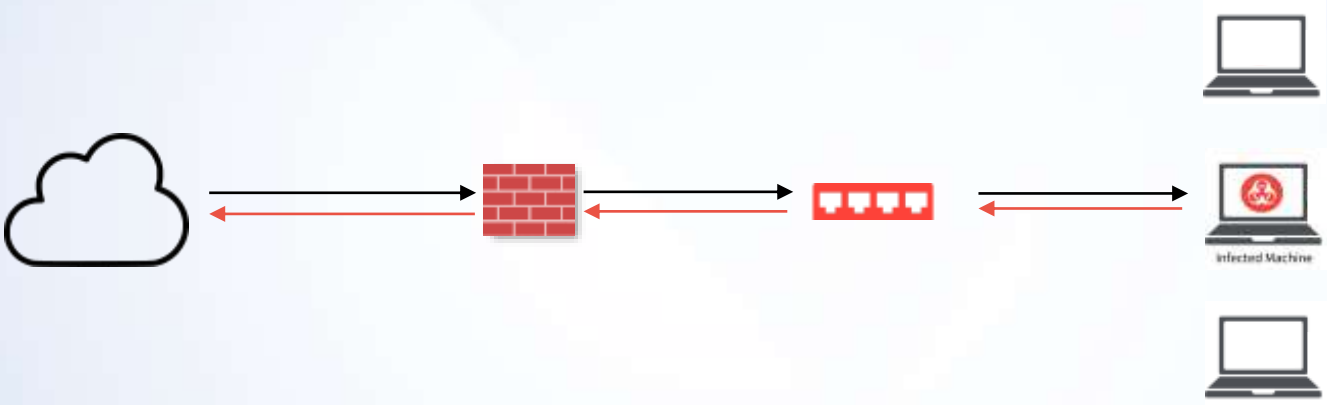
- Can we listen for packets
  - Does require local admin
  - Direct traffic to a wrong node and collect the traffic

Listen for what you want

- Windows starting in XP cannot send data on a raw socket but it can listen
- Alternatively, you may use WinPcap drivers

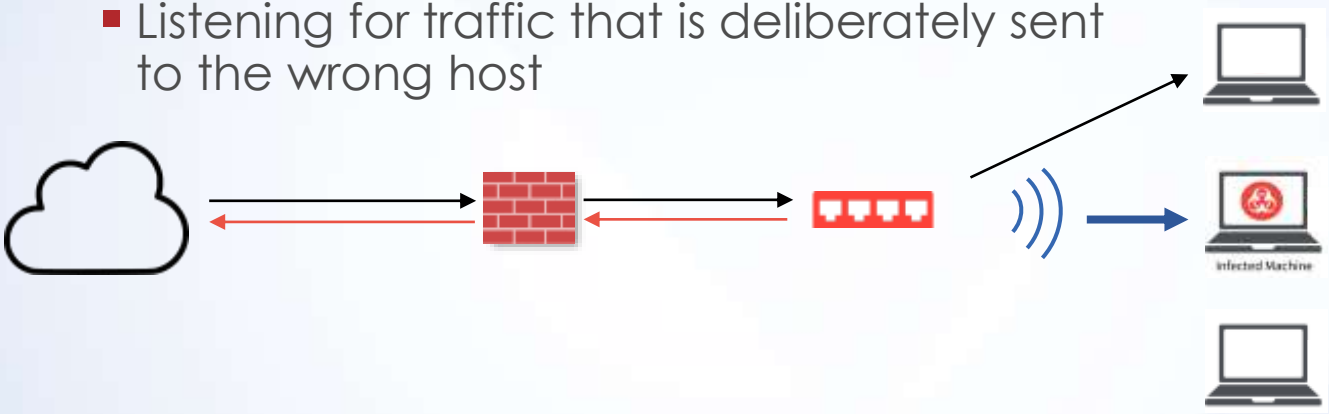
Listen for what you want

- All analysis is based off the idea of hosts talking to each other



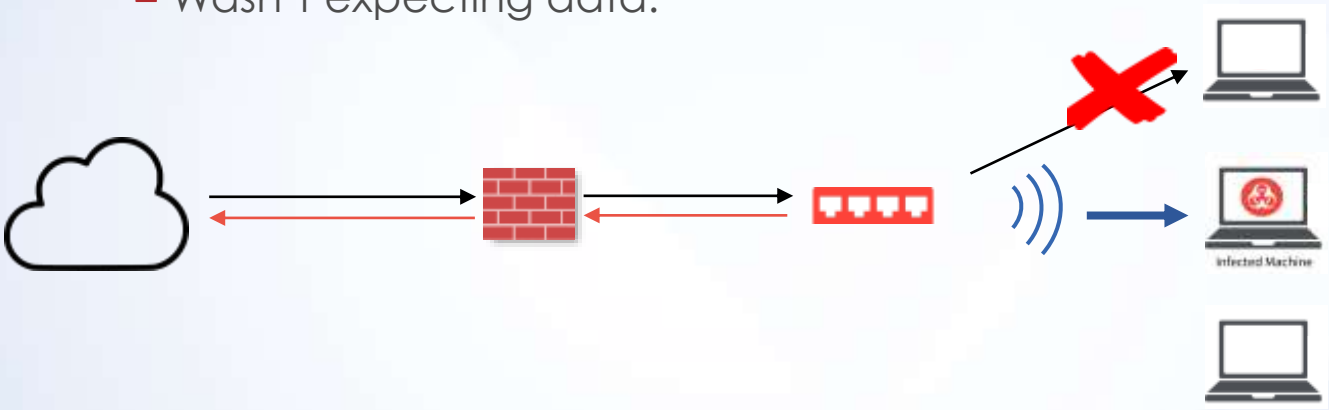
Listen for what you want

- What if instead we sent to a random machine in the network and just listened on a raw socket?
- Listening for traffic that is deliberately sent to the wrong host



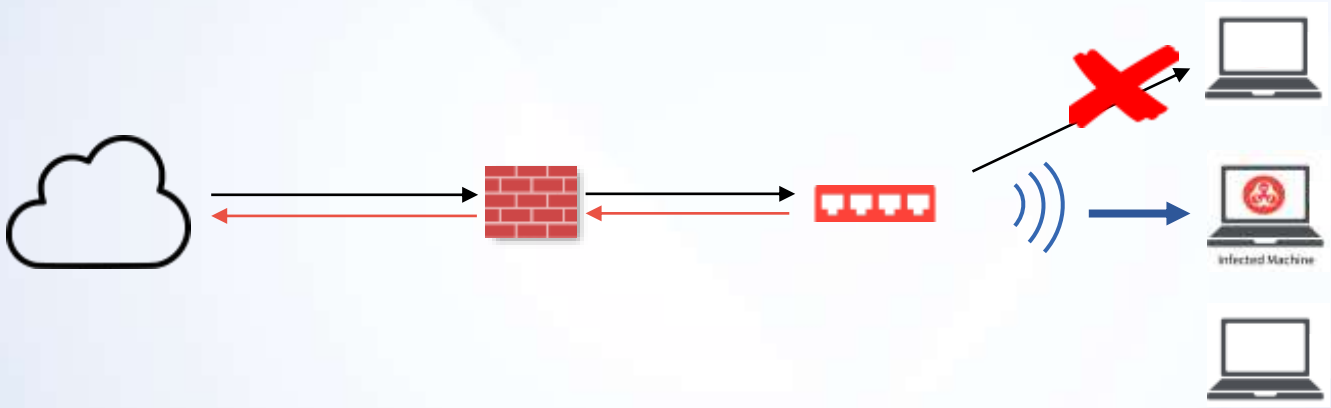
Listen for what you want

- The wrong host is likely to drop the packets
  - Invalid port
  - Wasn't expecting data.



Listen for what you want

- Finally mask the traffic similar to how previous packets of data have appeared being sent to the host.



Listen for what you want

- To send tasking the controller needs to transmit tasking encapsulated in valid TCP network traffic
- Typically requires connecting to an (open) listening port on the victim
- Send raw packets, skips the triple handshake.

Listen for what you want

- REDSALT
  - A loader that decodes a second-stage loader that decodes another payload containing a backdoor capable of listening for commands using a raw socket or obtaining commands from a URL or file.

MANDIANT
A FireEye® Company

The Future?

The game keeps changing and you have to change with it

Wrapping up

- Defences are improving.
  - Defence in depth continually adds layers

Wrapping up

- Blend in so defenders can't distinguish between you an attacker and legitimate activity
  - Machine learning
    - Has a margin of error, we need to sit under it.
- Other Technologies not covered
  - WCF
  - Packet Stuffing
  - Serverless Computing

MANDIANT
A FireEye® Company

Thank you

"Be extremely subtle even to the point of formlessness. Be extremely mysterious even to the point of soundlessness. Thereby you can be the director of the opponent's fate."

~Sun Tzu, The Art of War

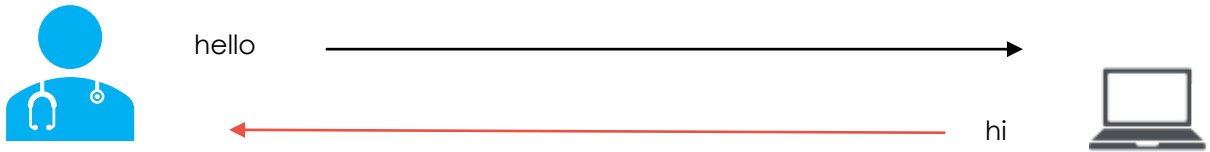
QUIC

- Packet Fragmentation is back?

To Encrypt or not Encrypt?

- Encrypted

- Plain text



Markov Obfuscation

- Cylance Spear Team