KubeCon | CloudNativeCon

Europe 2020

Virtual

How Many CPU Cycles I Need to Invest in Cloud-Native Security?

Ben Hirschberg

whoami

That's not me!

Balling with enthusiasm

Ex-hacker turned cloud native entrepreneur

4 kids -> no spare time

Love teaching!

:toc

KubeCon | CloudNativeCon Europe 2020 Virtual

TLS in cloud native

benchmarking

evaluation

improvements

man whyRwehere

Attackers are everywhere

Our security is nowhere

Find solution anywhere

Costs should stay somewhere



RFC 8446

What is TLS?

a) A psychedelic drug
b) A secure communication protocol over TCP
c) Wait, TCP is the psychedelic drug

What security features TLS gives:

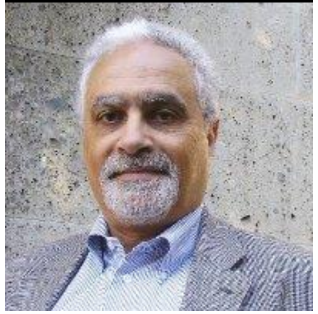
a) Confidentiality
b) Integrity
c) Authenticity
d) All three above

What cryptographic algorithms are used by TLS?

a) AES
b) SHA
c) RSA
d) Which are not?!

Who are the two “original” endpoints of TLS?

a) Alice and Bob
b) Netscape and Httpd
c) Sidecar and sidecar
d) Client software and TLS termination hardware



tcpdump port 443

Parts of TLS protocol

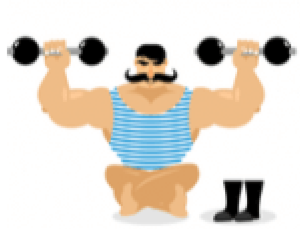
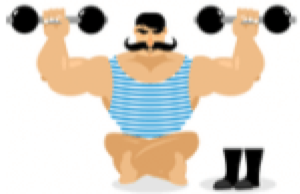
Handshake protocol →
- Happens per connection establishment
- Cipher suit negotiation
- Authentication (mutual) and key exchange
- Uses asymmetric cryptography!

Application protocol →
- Per application message
- Encryption and message authentication
- Uses symmetric cryptography

Alert protocol →
- Happens when there is an error in TLS
- No cryptography involved

top –p $(pgrep nginx)

Computational requirements of TLS

	Handshake protocol	Application protocol
CPU		
RAM		
NET		

kubectl create tls

TLS types in cloud native environment

North-south:
- TLS termination
- Mostly one-sided TLS

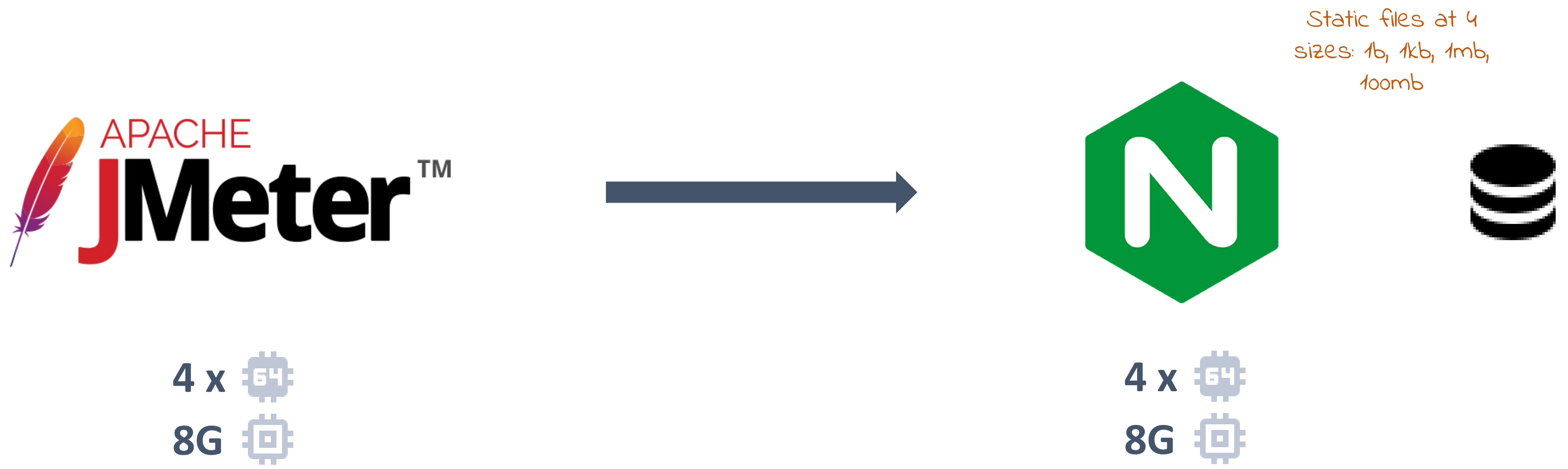
Extra CPU usage

East-West:
- Setup
  - Homebrew
  - Sidecar proxy based
  - Inline based
- Cases of mutual authentication (mTLS)

Extra RAM usage

jmeter run

Simple test setup



cat results | head

Initial results…

round()

Let's do some rough calculations!

time curl https://

TLS handshake timings

Server certification validation (normal)

- Client CPU time ~ 2msec
- Server CPU time ~1msec

Mutual certificate validation (mTLS)

- Client CPU time ~2.5msec
- Server CPU time ~2.5msec

Given Elliptic-curve Diffie-Helman key exchange with RSA signature with a 2048 bit key

time curl https://

TLS application protocol

Encryption

- Pure software AES256 ~250Mb/sec

Message authentication

- Software SHA256 ~300Mb/sec

Rough numbers for the sake of discussion, not final ;)

1Mb traffic requires ~7.3msec CPU time at one side and ~14.6msec on both

143,640kb/s

qalc –t “1000=x”

Formula for calculation of transaction per second

n = number of transactions
H_c = client side handshake CPU time (seconds)
H_s = server side handshake CPU time (seconds)
S = encryption bandwidth (bytes/seconds)
T = bytes in one transaction

$$(H_c + H_S)n + \left(2\frac{T}{S}\right)n = 1s$$

$$n = \frac{1s}{H_c + H_s + 2\frac{T}{S}}$$

jmeter-it

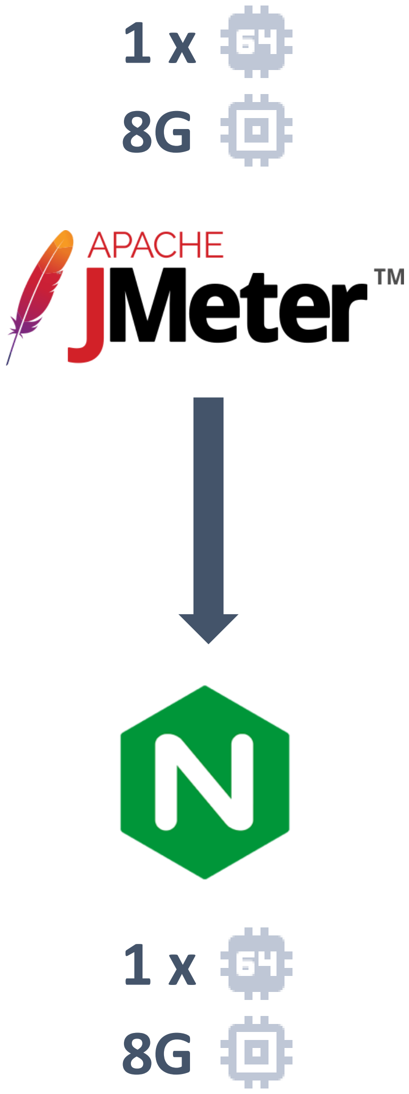
Test application 1.

Server:

- TLS handshake per TCP connection
- TLS Crypto parameters as defined before
- 1Kb application data per transaction

$$n = \frac{1s}{0.002s + 0.001s + 2\frac{1024b}{143{,}640{,}547\frac{b}{s}}} = \frac{1s}{0.002s + 0.001s + 0.0000142578125s} = 331.75$$

Actual: ~0.003043 s/request => **~328/s**



jmeter-it

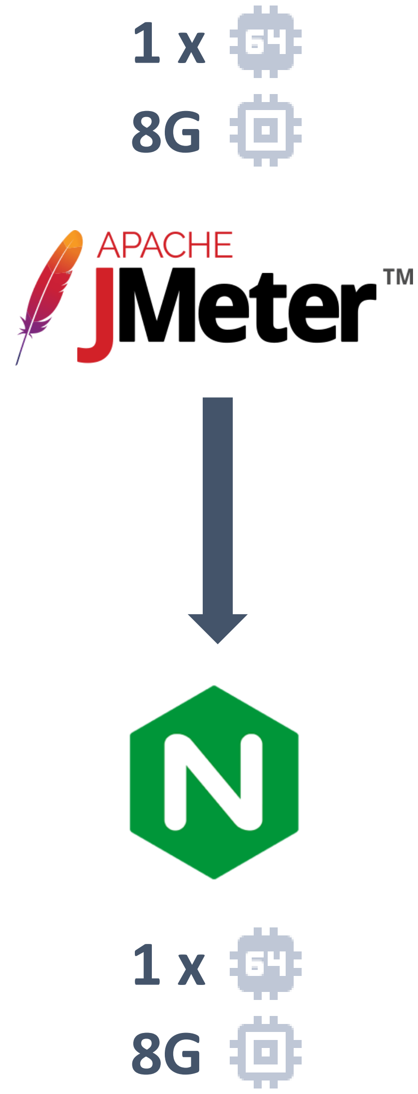
Test application 2.

Server:

- TLS handshake per TCP connection
- TLS Crypto parameters as defined before
- 100Mb application data per transaction

$$n = \frac{1s}{0.002s + 0.001s + 2\frac{104{,}857{,}600b}{143{,}640{,}547\frac{b}{s}}} = \frac{1s}{0.002s + 0.001s + 0.73s} = 0.683$$

Actual: ~1.134 s/request => ~0.7/s



apt-get install openssl

Example calculation of overhead added by TLS

$n = 5000$
$H_c = 2.5$
$H_s = 2.5$
$S = 150Mb/s$
$T = 1024b$

$$(0.0025 + 0.0025)5000 + \left(2\frac{1024}{150Mb/s}\right)5000 = 25.06s$$

Main component is handshake!

rm –rf /tmp/*

Improving handshake: performance of cryptographic algorithms

Key exchange algorithm:	Elliptic Curve Diffie-Hellman (no contest)
Key exchange signing algorithm:	RSA 2048 + SHA256
Certificate validation:	RSA 2048 + SHA256

Least CPU consuming but still secure

history | grep keys

Improving handshake: expedited handshakes

Client-side session tickets

- Enables the client to reconnect server without full handshake
- Makes respective handshakes 10x faster
- Client needs to “remember” the server

PSK – pre shared keys

- Session establishment without using asymmetric cryptography
- Makes every handshake 10x faster
- Only for those who have infrastructure for pre-sharing keys…

tickets

Example calculation with expedited TLS handshake

n = 5000
H_c = 0.2
H_s = 0.2
S = 150Mb/s
T = 1024b

$$(0.0002 + 0.0002)5000 + \left(2\frac{1024}{150Mb/s}\right)5000 = \sim 2.1s$$

way better!!!

grep aes /proc/cpuinfo

Improving application protocol

Encryption

- AES-NI support: can go up-to 3Gb/s
- AES-128 is 33% faster than 256 but is phasing out
- Chacha algorithm is preferable where no AES hardware support

Message authentication

- Intel lacking SHA accelerator support in most servers
- SHA3-256 is the fastest hash with acceptable security
- Authenticated encryption: AES-GCD

KubeCon
CloudNativeCon
Europe 2020
Virtual
HELM
KEEP CLOUD NATIVE
CONNECTED