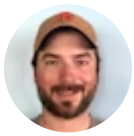
The case for using TLS everywhere

Updated on: May 20, 2024



Mike Malone

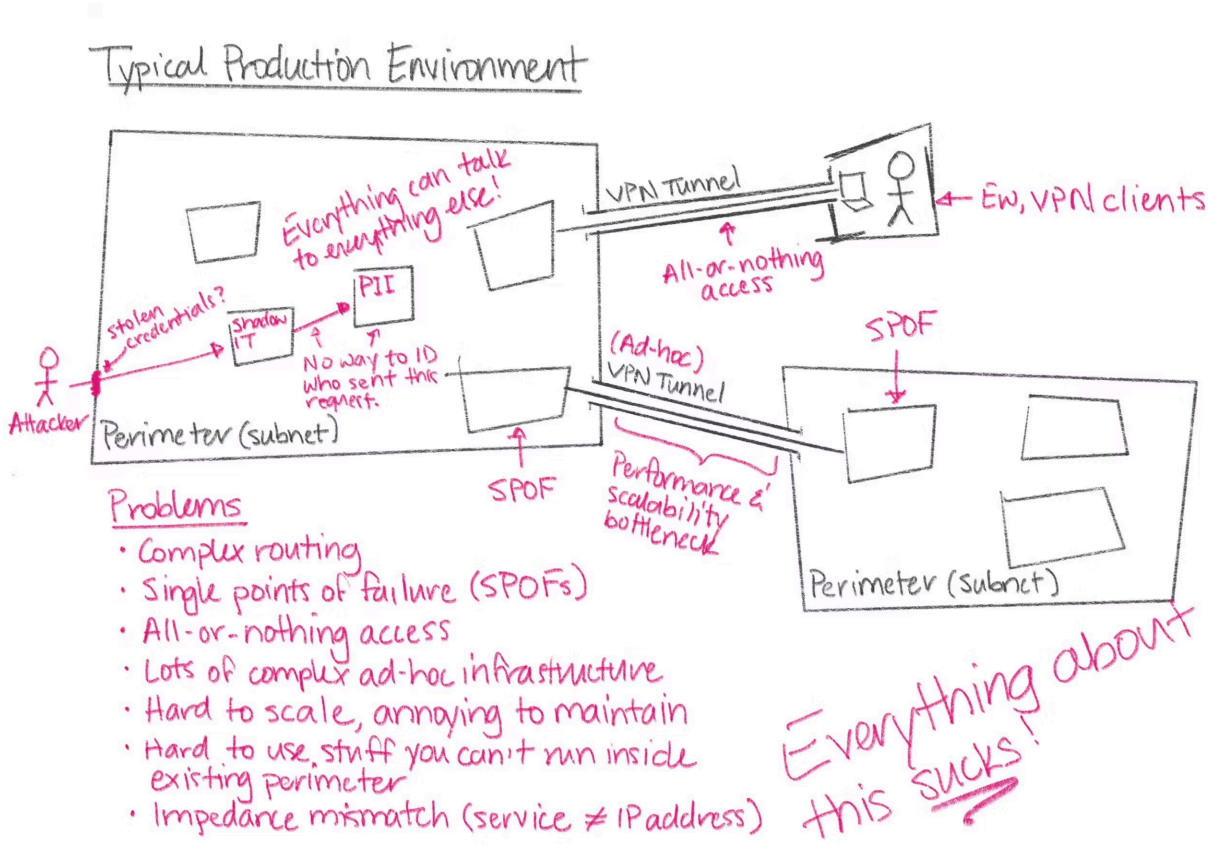
This post has a simple purpose: to persuade you to use TLS everywhere. By everywhere, I mean everywhere. Not just for traffic coming from the public internet to your website and APIs, but for every internal service-to-service request. Not just between clouds or regions. Everywhere. Even inside production perimeters like VPCs.

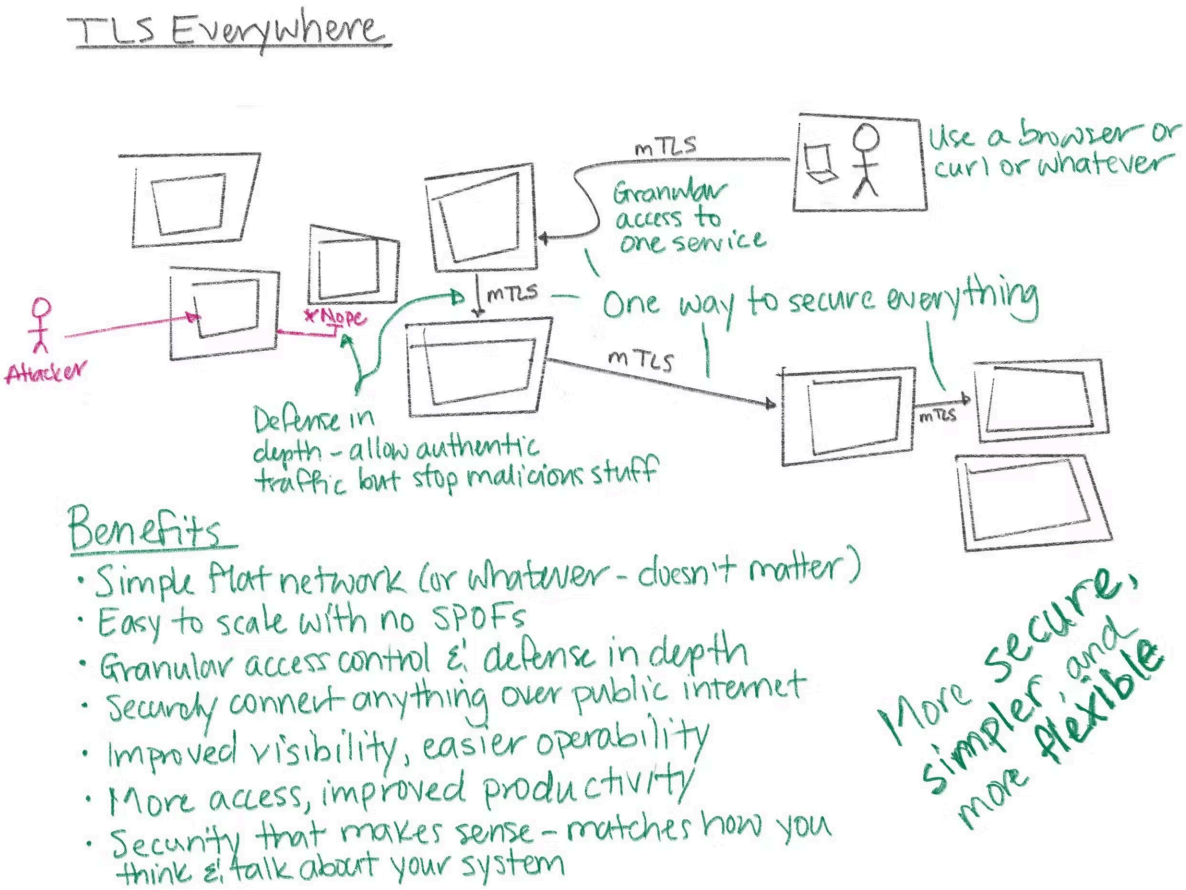
I suspect this recommendation will elicit a range of reactions from apathy to animosity. Regardless, please read on. Using TLS everywhere has a wide range of benefits. Perhaps surprisingly, some of the most compelling benefits have nothing to do with security.

Why?

To start, the common objections to ubiquitous cryptography no longer apply to TLS: TLS is fast, easy to use, and works everywhere. Conversely, the traditional perimeter security approach has nothing to recommend it outside a jaded and atavistic attachment to the status quo. We use network level controls like iptables and VPCs because that's what we've always done. That's dumb. That's not computer science, it's computer religion.

While the traditional perimeter security story is poor, you're probably already using stuff like VPCs and VPNs successfully. TLS everywhere is more secure, but "better security" is a hard sell. There are more compelling reasons to use TLS everywhere. It's liberating: since TLS works everywhere your stuff can run anywhere. It's also simpler: TLS lets you replace countless fragile and frustrating bits of security infrastructure with a single consistent solution that's easier to understand, easier to scale, and easier to operate.





TLS everywhere is more secure

A traditional perimeter consists of a firewall with coarse access controls that allow trusted people and code inside. Within the perimeter, additional security is scarce or non-existent: anything inside can send any request to anything else. This model makes assumptions that no one actually believes to be true: that code never has bugs, that people are incapable of mistakes, are never dishonest or malicious.

TLS provides confidentiality and integrity — attackers can't see or change stuff sent across the network. Beyond that, mutual TLS provides an "identity dialtone" — like caller ID. Two things communicating using mutual TLS know each other's names. You need to know who sent a request before you can decide whether it should be allowed — you must authenticate before you can authorize. Thus, some notion of identity is a necessary prerequisite for security best practices like least authority and defense in depth. TLS satisfies this prereq.

In short, mutual TLS helps to reduce the blast radius after a compromise. When an attacker gets inside your system it becomes much harder for them to move laterally to find and exfiltrate high value data like passwords, PII, or credit card information. This is not a purely theoretical benefit. It's how real attacks work in the real world. For example, it's what happened to Google in 2009 with Operation Aurora. Their response was to adopt end-to-end cryptography everywhere using a proprietary protocol that's essentially identical to modern mutual TLS. Google has demonstrated that this approach makes sense and works at scale.

An "identity dialtone" has other benefits. It dramatically improves runtime visibility: knowing who or what made a request certainly makes it easier to measure and debug. Counterintuitively, identity can also increase access. Access to a perimeter is all-or-nothing. Unable to grant granular authority, access is frequently denied altogether. This is very common. It's also demoralizing and it impedes productivity. With TLS, granular authorizations become possible. As an added bonus, you don't need a clunky VPN client to connect once access is granted. So using TLS everywhere can actually improve access, thereby improving productivity.

Here's the bottom line: identity is fundamental. It's so incredibly basic and broadly useful that further explanation feels superfluous. Yet an inability to identify microservices and other bits in a distributed system remains shockingly pervasive. It's an embarrassment, really. Using perimeters to draw boxes around stuff instead of actually identifying each part might be easier at first, but it's enormously expensive over time, with ramifications that extend well beyond security. Perimeters become barriers, insidiously constraining our systems. We're so accustomed to these constraints that we often take them for granted.

TLS everywhere is liberating

The old school definition of a "system" is physical: a box on a desk or a building housing a data center. This doesn't work in the cloud. VPCs virtualize these old notions, easing the transition. But the way you define your system will also constrain it. The most pernicious form of vendor lock-in is the use of vendor-specific capabilities to define what your system is.

The practical consequences of relying on stuff like VPCs for security are rigidity and complexity. Suppose you're in AWS but want to start running containers with Kubernetes using GKE in Google's cloud. Rigidity manifests as an inability to use GKE because it doesn't fit inside your existing AWS-defined perimeter. Stuff in AWS can't talk to stuff in GCP. Alternatively, you might choose to use GKE regardless of cost. Doing so requires an ad-hoc extension to your perimeter, bridging AWS and GCP (e.g., with an IPSec-based VPN tunnel). This adds substantial complexity.

Every ad-hoc bridge you add has a significant ongoing operational cost. Bluntly, VPN tunnels are an operational ass pain, a scalability bottleneck, and a single point of failure to boot. But staying within one perimeter means you can't leverage best of breed technologies. You're damned if you do, damned if you don't.

While VPN tunnels are annoying, at least they're well understood. The question of how best to extend perimeters to incorporate serverless, edge computing, and IoT remains open. It's hard or impossible to deploy a VPN in a constrained environment, or where you're unable to fiddle with the network stack. TLS works in all of these environments.

Using perimeters to define your system by drawing boxes around your stuff, and using tunnels to add lines between those boxes, is silly. A better option is to define your system exactly how you want using TLS along with your own internal public key infrastructure. Instead of "the stuff inside your AWS-defined perimeter" your system is defined as "the stuff that's been issued a certificate from your certificate authority". You can issue certificates to anything, running anywhere, and securely communicate over the public internet. Using TLS everywhere liberates your system from a particular vendor and, more broadly, from the details of your current operational environment. It's way more flexible. It's also simpler.

TLS everywhere is simpler

Since TLS works everywhere it's a uniform approach to securely communicate with stuff running anywhere. This dramatically simplifies your security story. You can get rid of all your ad-hoc security infrastructure and replace it with... nothing. You can just turn off your VPN concentrators and dump all those complex routing rules. With TLS, communicating between AWS and GCP works exactly the same as communicating within AWS. You'll get lower latency, more throughput, you'll have fewer outages, and

network scalability will become a non-issue. Using TLS everywhere flattens your network and reduces the number of moving parts in your infrastructure.

Beyond simplifying your physical infrastructure, using TLS is logically simpler. It makes security easier to implement and easier to reason about. Perimeter security is based on IP and MAC addresses. But we think and talk about systems in terms of services and other logical parts. There's an impedance mismatch here: to enforce a security policy that says "Service A can talk to Service B" we need to maintain a mapping between service names, the hosts they run on, and various network level constructs. Using TLS everywhere eliminates this impedance mismatch. We can describe and implement security policies naturally, at the right level of abstraction. The network is completely abstracted away — you can ignore a bunch of irrelevant network detail. It's just simpler.

What about *something else*?

There are other ways to define your system cryptographically. You could build a software-defined network using IPSec or WireGuard or Calico, for example. A survey of these various approaches would fill a book. Thankfully that's unnecessary since TLS has some obvious advantages that make it the right default.

First, TLS is the safest and cheapest option. TLS is the most widely deployed security protocol in the world. It's undergone extensive "hostile review". No other protocol has been so thoroughly vetted. This makes TLS your safest choice (pragmatically, if not academically). Like any cryptographic protocol, TLS vulnerabilities do pop up from time to time, but they're hard to miss and quickly remedied. And since you're already using TLS for your public websites and APIs (right?), the burden of more widespread internal use is lower than it would be to adopt something new.

The ubiquity of TLS also translates to a better experience for developers, operators, and everyone else who interacts with your system. Everything supports TLS. So using TLS minimizes any impact on existing workflows: you can still connect to stuff from web browsers and using tools like curl and postman.

Since every language and framework supports TLS, it's also the only cryptographic protocol you can reasonably terminate in applications. This sort of end-to-end cryptographic channel is the holy grail for security and performance. While TLS makes

this as easy as it can possibly be, it still requires code changes. That might be politically unpopular or, at the very least, take a long time. Thankfully, you can also terminate TLS in a sidecar proxy with no code change necessary. You can easily mix these deployment modes to balance security, performance, pragmatics, and various operability concerns. TLS is the only option that makes this easy.

Finally, TLS is the only protocol that integrates smoothly with HTTP and other common application layer protocols. Protocol negotiation provides a standard mechanism for protocol discovery, which gives your security layer easy access to request details to make access decisions. In other words, you can granularly authorize requests based on things like request URLs and headers. You can let someone GET, but not PUT or POST.

There are other ways to do all of this stuff, but TLS is the only option that makes all of this easy. While technologies like IPSec and WireGuard exist for good reason, TLS everywhere is the right default for securing modern systems.

Why not?

Security is hard, and it's tempting to offload responsibility as quickly as possible. But rote application of outdated techniques and over-reliance on vendor-specific solutions has pernicious effects. TLS is a better option, with benefits that extend well beyond "better security". Using TLS everywhere puts you back in control of defining your system. It eliminates complex physical infrastructure, and it raises the level of abstraction so we can talk and reason about security more naturally. In short, it's more flexible and it's simpler.

Considered independently, one could argue whether a particular benefit of using TLS everywhere is worth the effort. Adding one ad-hoc VPN tunnel is probably easier than adding TLS to an existing system, for example. But this misses the bigger picture: if we consider the totality of these benefits it's clear that using TLS everywhere is the right default today, and for the future. It's about increased access, ease of use, and more agile and flexible systems that are simpler and easier to operate. TLS everywhere reduces operational costs; it improves scalability, reliability, and visibility; it makes adopting new compute paradigms and best of breed technologies easier, regardless of where they run; it improves productivity and the general well being of the people that build and operate your system, every day.

With that I believe the question “why TLS” has been asked and answered, and the burden of proof has shifted. TLS should be your new default — the presumptive right answer. From now on, if you’re not using TLS the onus is on you to answer: **why not?**

Cropped header photo Escape Velocity by Sam Howitz.

Mike Malone has been working on making infrastructure security easy with Smallstep for six years as CEO and Founder. Prior to Smallstep, Mike was CTO at Betable. He is at heart a distributed systems enthusiast, making open source solutions that solve big problems in Production Identity and a published research author in the world of cybersecurity policy.