Coder

Zero-Trust Security for Software Development Teams

Organizations are increasingly adopting a zero-trust approach to network security. But what are the implications of this move for software development teams? Learn how a zero-trust approach differs from traditional approaches to network security and how using Coder to manage and orchestrate development environments can actually simplify the implementation of zero trust and help engineers to code faster and more securely while working in zero-trust environments.

contact@coder.com
512-593-8418

With the dramatic increase in remote work in the wake of COVID-19, many companies realized that their traditional perimeter-based approach to network security was no longer sustainable. With dozens, hundreds, or even thousands of workers suddenly needing remote access to all their applications and services, network admins scrambled to ensure that: 1) employees could continue to work productively, and 2) security protocols were being maintained and adhered to. In many cases, admins were forced to improvise with network architectures that were not designed to support remote access, or at least not at the scale and scope now required.

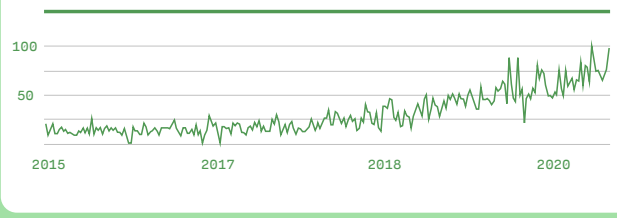
In many ways, COVID-19 simply accelerated, albeit dramatically, a trend toward remote work that has been growing for years. Other trends such as the mass adoption of cloud computing and cloud data storage already had networks creaking under the pressure to secure globally distributed assets.

It is precisely these issues that prompted the U.S. Navy's CIO to **announce** recently that the service was using the disruption from the coronavirus pandemic as fuel to change its cybersecurity architecture. The Navy will move toward a zero-trust model to better defend its networks. The same week of the Navy's announcement, the **Department of Homeland Security announced** that it also was adopting the model. Zero Trust is a dramatically different approach to cybersecurity than traditional firewall-based approaches and is widely seen as better meeting the needs of the modern enterprise.

The zero-trust model was first proposed by security expert **John Kindervag in 2010** when he was with Forrester Research. The problem, Kindervag argued, is that our traditional networks were built with too much trust:

> **"If the current trust model is broken, how do we fix it? It requires a new way of thinking. The way we fix the old trust model is we begin at the beginning and look for a new trust model. Forrester calls this new model "Zero Trust." The Zero Trust Model is simple: Security professionals must stop trusting packets as if they were people. Instead, they must eliminate the idea of a trusted network (usually the internal network) and an untrusted network (external networks). In Zero-trust, all network traffic is untrusted."**

The approach received a massive endorsement in 2014 when Google announced **BeyondCorp**, their implementation of the zero-trust security model.



Google searches for "zero-trust" have risen steadily since the beginning of 2020.

While many companies have implemented the model, it is still far from widely adopted despite having clear advantages over more traditional approaches. Implementing something called "zero trust" may sound daunting to companies that lack the resources of Google (almost every other company in the world). Others may simply worry about having to "rip and

replace" their current architecture. But as with the U.S. Navy, COVID is forcing many to reconsider in order to meet the new normal needs of modern organizations.

Thankfully, the task is not as daunting as it may first appear, and the benefits are well worth the effort.

Here at Coder, of course, we are focused on software development, and our mission is to improve the developer experience so that they are able to innovate rapidly in distributed or remote environments. So why, you may be wondering, are we writing about zero trust? Because both developers and the applications they produce will increasingly be expected to work in zero-trust environments.

In this paper, we will look at how a zero-trust approach differs from the traditional approach to network security and how it increases security for the enterprise in general. We will then examine the implications of zero trust for software development teams and how our product, Coder, can simplify the implementation of zero trust for development teams and help them to code faster and more securely while working in zero-trust environments.

Note: Readers who are already familiar with the principles of zero trust should feel free to skip ahead directly to the discussion of the implications of zero trust for software development teams.

Traditional Approach: "Trust, but Verify"

The traditional approach to network security goes by many names, from the simply descriptive perimeter-based approach to more metaphoric monikers such as walled garden or castle and moat. They are all attempts to describe an architecture designed to keep bad actors out of the network. To protect the castle you build walls high enough that they are difficult to broach. You dig a moat deep enough that it is impossible to wade across (knights wearing armor are seldom strong swimmers). Finally, you restrict access to the castle by using a draw bridge that can be pulled up to prevent brute force attacks at the front gates. Once those measures are in place, your castle should be secure from intruders. But the modern enterprise is not a castle. A recent **National Institute of Standards and Technology report on implementing zero-trust architecture** points out how the enterprise has changed while its approach to network security has failed to keep pace:

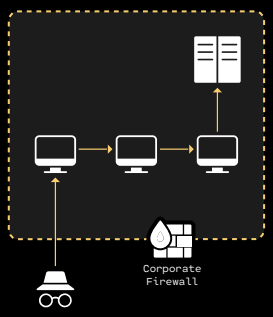
> **"A typical enterprise's infrastructure has grown increasingly complex. A single enterprise may operate several internal networks, remote offices with their own local infrastructure, remote and/or mobile individuals, and cloud services. This complexity has outstripped legacy methods of perimeter-based network security as there is no single, easily identified perimeter for the enterprise."**

At its core, even in a traditional or simple, single location, the perimeter-based approach suffers from a couple of inherent flaws. For a complex modern organization, the approach is untenable and dangerous. First, it sets an impossibly high bar of

keeping out all bad actors. Second, in focusing so much on securing the perimeter, the model tends to pay too little attention to risks inside the firewall. It generally assumes that traffic originating from within the network can be trusted as long as the requestor properly authenticates — a "trust, but verify" approach to borrow a cold war adage.

Once an intruder has cracked the perimeter, or once a trusted individual within the network becomes a bad actor, it is far too easy to move laterally and gain access to data and systems that they shouldn't be able to access. In fact, hackers seldom gain access to their primary target as their first attack surface. Rather they usually gain access to a low-value asset and then move laterally through the network until finding a target of value.

A **2018 report by cybersecurity firm Crowdstrike** reveals that the average time between the initial compromise of a system and the first lateral movement of the attacker is one hour and 58 minutes. Zero trust takes a different security approach, one that directly addresses both of the perimeter-based approach's flaws and is more applicable to today's complex organizations.



Intruders may bypass corporate firewalls and compromise a device through phishing or social engineering. They then typically make several lateral moves before gaining access to a target of value. Because traffic originating from within the firewall is trusted, this lateral movement can occur with surprising speed.

Zero Trust: "Never Trust Always Verify"

The traditional perimeter-based approach utilizes a "trust, but verify" approach — it typically assumes that if the request is coming from inside the network the requestor can be trusted but must verify their identity, usually through password authentication.

In zero trust, the mantra is "never trust, always verify." It takes as a basic assumption that any request for access to a system or data is hostile until proven otherwise, regardless of its origin. A request coming from a laptop sitting in the same room as a server is treated the same as a request from an IP address around the world.

Verification typically requires multiple tests, which may include multifactor authentication, device registration, authorization policies, and other checks. A failure to satisfy any of these tests results not only in denied access but also automated security alerts.

Zero trust also shifts the defense focus from creating a single monolithic perimeter to building multiple microperimeters surrounding an organization's most valued assets. In doing so, access policies can be created at a very fine granularity using the principle of least privilege, which is a key foundation to any zero-trust architecture.

A 2019 **report produced for the Department of Defense** provides a specific example of the value of a zero-trust approach:

> **Some of the most severe cases of network breaches could have been prevented using basic zero-trust principles — for example, had ZTA [zero-trust architecture] access rules been applied to Edward Snowden, he would have been unable to obtain the broad range of documents that he released to the public. Instead, he was given "system administrator" privileges within the NSA network, which provided him blanket access to resources and files. This method of blind trust in users and devices inside the perimeter [of] the network is not sustainable, and will continue to put national security information and operations at risk until it is resolved.**

The claim that zero trust could have prevented the biggest intelligence leak in the NSA's history is quite an endorsement.

Implementing Zero Trust

It is important to understand that zero trust is an approach, not a service or a device that can be purchased and connected to your network to achieve zero trust. As **Tim Woods notes in InfoSecurity Magazine**: "Zero trust cannot be achieved by a single solution. Rather, it's the collective result of incremental evolutions across security infrastructure and operations."

Fortunately, implementing a zero-trust architecture is not particularly difficult. In fact, **a guide to implementation produced by Palo Alto Networks** claims that doing so "is actually much simpler than building legacy 20th-century hierarchical networks." Implementation is also simplified by the fact that the zero-trust network does not have to be stood up fully formed, and it can co-exist with your existing legacy network.

Because zero trust is a mindset, there is no single approach or technique for its implementation. Palo Alto Networks, where zero-trust originator John Kindervag is now Field CTO, identifies a **"five-step methodology"** to implementation. Akamai offers an eight-step "**Comprehensive and Achievable Roadmap**," while security firm Lepide promotes a **three-step approach** that actually mirrors Kindervag's three key concepts of zero trust as originally presented in his 2010 paper.

Regardless of the precise number of steps required to get there, to implement zero trust you must first identify an asset that needs to be secured, sometimes referred to as the protect surface (data, source code, etc.). You then determine what people and systems require access to the asset and under what conditions (e.g., only when using a specific device, or only during a particular time window). With this information, you can then create access control rules that deny access in all other scenarios. Finally, you monitor traffic and take action as required. The process is repeated for each asset that needs to be protected. Eventually, you have numerous customized microperimeters surrounding your organizations' most valued assets.

Zero Trust and Software Development

When considering the implications of zero trust for software development, there are two distinct but related issues to consider: the implications for the software being developed and the implications for developers working in a zero-trust environment.

As developers are creating apps, they should assume that those apps will be deployed into zero-trust environments and the developers should code appropriately. As Palo Alto Networks points out in their implementation guide, “There are no Zero Trust products. There are products that work well in Zero Trust environments and those that don’t.” As zero trust becomes the default across enterprises, you don’t want to be the company producing an application that does not work well in the environment.

Thankfully, just as it isn’t particularly difficult to implement a zero-trust environment, it isn’t overly difficult to ensure that an application can function in such an environment. Developers should follow the same basic principles guiding zero-trust implementation: think about what assets need to be protected, determine who needs access and under what conditions, enforce the proper checks, and finally, of course, never trust, always verify. The key, of course, is ensuring that these issues are considered and addressed as early as possible in the development process rather than trying to retrofit a solution, which can result in costly delays to production.

If this sounds familiar, it should. It’s straight out of the DevSecOps playbook that seeks to incorporate security thinking throughout the development process and as early in the process as possible. Zero trust and DevSecOps are natural partners, with zero trust helping to inform the security checks that are part of the DevSecOps process.

The question about the implications for developers working in a zero-trust environment is more difficult to answer since every implementation of zero trust is different. Without a doubt, a poorly executed zero-trust environment can hurt productivity. Imagine a kitchen where you have to multi-factor authenticate every time you open the refrigerator, turn on the faucet, open a drawer to grab a spoon, or lift a lid to stir a pot. It might be a very secure kitchen, but it would be almost impossible to cook a meal in it.

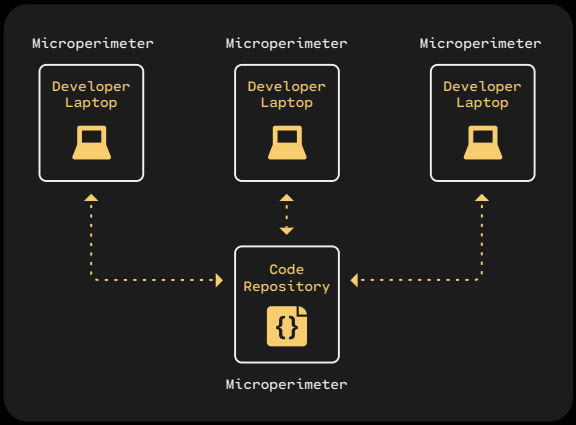
How Coder Simplifies Zero-trust Implementation

A product like Coder can be a great advantage for organizations that are implementing a zero-trust network for software development teams. In the distinction quoted earlier by Palo Alto Networks, Coder not only works well in a zero-trust environment, but it also can help simplify the implementation.

Coder significantly reduces the number of protect surfaces that must be defended in a zero-trust implementation. It does this by moving the software development environment to the cloud. Traditionally most software development is done on individual developers’ personal workstations or VDIs. As a result, these machines contain source code and data that would be considered assets that must be protected. This would require creating and monitoring microperimeters for each device. For organizations with dozens, hundreds, or even thousands of developers this could be unwieldy.

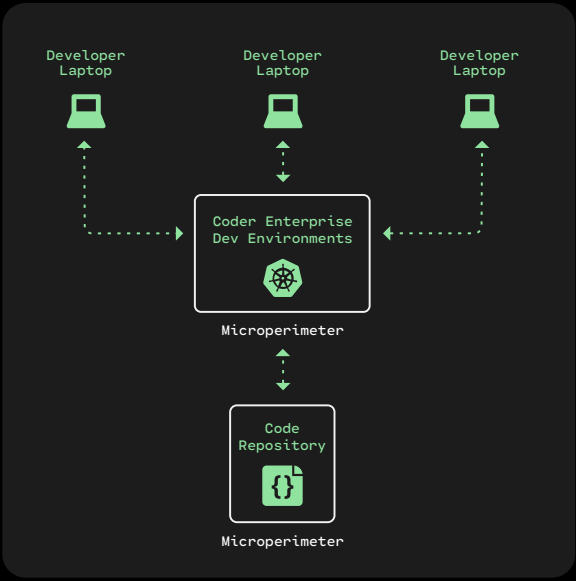
With Coder, when you use either VS Code or JetBrains through the browser all development actions

Dev environments on Individual Endpoints



When dev environments run on local devices that contain code and data, they must be treated as protect surfaces in a zero-trust environment, necessitating the creation, maintenance, and monitoring of a unique microperimeter for each device.

Microperimeters with Coder



Coder reduces the number of protect surfaces by moving the dev environments off of individual devices and onto containers running in the cluster. With no code or data stored on the developers' devices, they do not need to be protected by microperimeters.

are performed on the internal infrastructure — all source code and data remain on the server where the development environments actually reside or in authorized repositories. Developers access those environments through a web browser from their personal devices. No code or data is ever downloaded to the personal device. The developer's laptop is no longer a protect surface requiring its own customized microperimeter, and instead could even be used as one of a series of access controls to verify the developer's request for access to the dev environment.

Coder creates development environments from images stored in a centralized repository These images can be hardened to meet the organization's requirements. Developers create their dev environments in seconds with a simple click of a button. Rather than spending hours setting up and configuring their development environments, developers can start coding immediately, and the enterprise can be assured that security vulnerabilities were not introduced by a misconfigured environment. If a vulnerability is discovered in an underlying image or a patch is issued, all environments created from that image can be automatically updated.

Writing for TechTarget, Michael Cobb notes that modern software is seldom developed entirely from scratch, which has **implications for teams developing in a zero-trust environment:**

> **The zero trust framework also requires not trusting the open source and third-party plugins required to create a modern application. It is vital developers know what components are used and how to track**

and implement updates and fixes. This is best done using a software composition analysis tool, such as Black Duck or WhiteHat Sentinel. ”

The same could be said about not trusting the open source operating systems and third-party tools that make up most development environments. This is another area where Coder's containerization of dev environments can play a crucial role in a zero-trust framework. The Docker images can be regularly scanned for vulnerabilities using tools such as Anchore, AquaSec, or BlackDuck. Images that are found to be out of compliance can be immediately deactivated until a patch can be applied.

Coder is fully compatible with the principle of least privilege, which is a key component of defining access control policies in zero trust. Users can be restricted to creating environments from only specific images, and access can be granted or revoked with a click of a button.

As mentioned earlier, any zero-trust implementation must allow for continual monitoring of network usage. Coder facilitates this by tracking all actions on the system in extensive audit logs. Admins can review and filter these logs by resource type, action, resource target, or user from within the Coder interface, or the logs can be exported for analysis by sophisticated tools such as Datadog or Amazon's GuardDuty that use machine learning, anomaly detection, and integrated threat intelligence to identify and prioritize potential threats.

In Conclusion

Zero trust provides a framework for network security that better meets the needs of the modern enterprise than the traditional perimeter-based approach. For enterprises that include application development teams, Coder is a platform for distributed software development that not only integrates well into a zero-trust environment but also can significantly reduce the effort required to implement zero trust. For more information about how Coder can centralize your organization's development initiatives and unlock substantial gains in both developer velocity and enterprise security visit **coder.com**.