Bastion zero

RSAC Innovation Sandbox 2022 FINALIST

Introducing BastionZero

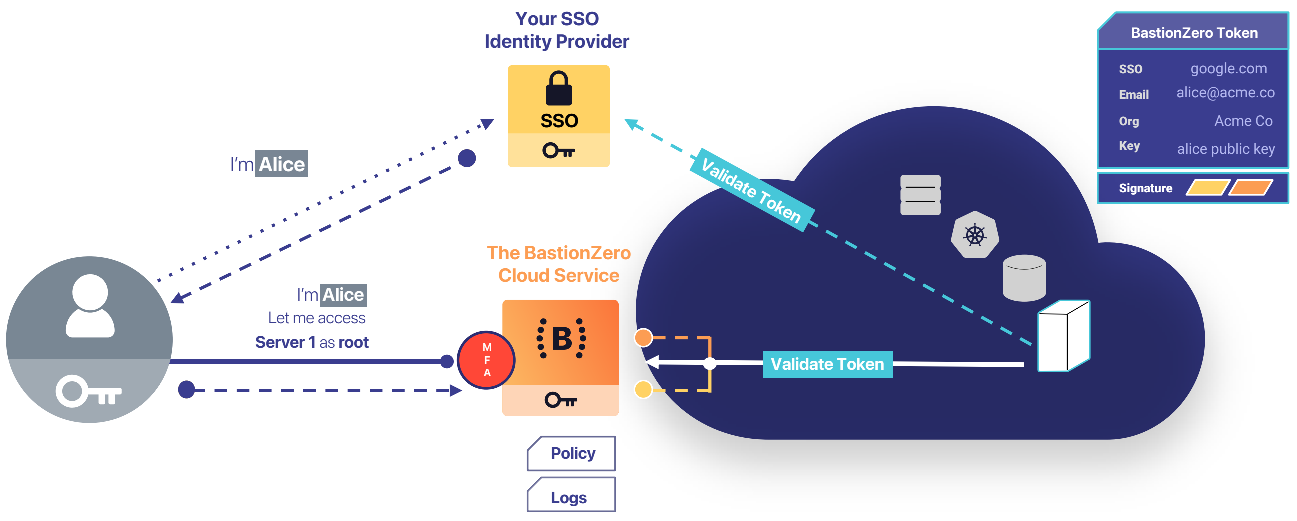
BastionZero's Trustless Access Platform connects teams to infrastructure in any cloud or data center, without any additional infrastructure to deploy or manage. Traditional zero-trust access tools have a fatal flaw. In their rush to improve security by eliminating user-held credentials, they store all those credentials in a single centralized location, where they can be compromised, all in one go, by attackers. BastionZerosolves the problem via a proprietary cryptographic technique that eliminate these single points of compromise.

BastionZero embraces the concept of limiting trust in the user by completely eliminating passwords and credentials for access to individual infrastructure targets. And beyond this, BastionZero avoids the pitfalls lead to the Colonial Pipeline, Uber and Medibank breaches, by going to great lengths to eliminate the trust placed in a single highly-privileged authority. BastionZero does not require organizations parties to store credentials in PAM or a highly sensitive bastion host. Instead, they subscribe to BastionZero cloud service. Using a third-party cloud service has major security implications, so this cloud service never stores the credentials for their access targets. In fact, our primary design philosophy is a radical commitment to minimize and eliminate trust assumptions in the cloud service. We do this by requiring users to authenticate against two independent roots of trust.

BastionZero has a unique trustless security model that uses two independent roots of trust that jointly control access to each infrastructure target. That way, a compromise of one root of trust does not lead to a compromise of your infrastructure.

One root of trust is the BastionZero cloud service, and the other root of trust is the relying party's IdP. That way, if one root of trust is compromised, the other can still protect the organization from a breach. Additionally, each user in MrTAP is issued public keys attested to by their IdP. This prevents BastionZero's cloud service from modifying commands send from user to target, or from injectings its own command. This also allows the cloud service to record logs of the actions that users take, so that potentially compromised end-hosts or servers do not need to be trusted to accurately record and send logs.

Finally, BastionZero is committed to a cloud-native approach where users log into targets, not networks. Instead of requiring organizations to set up VPN, SASE, firewalls and DLP systems to control users' access to a network, BastionZero uses a centralized policy manager to control users' access to individual targets. BastionZero's phone-home architecture eliminates the need for a VPN, and its passwordless solutions leverage cryptographic techniques like multi-party computation to eliminate the need to maintain a PAM or secrets vault that store credentials to individual targets. We do not ask organizations to manage and administrate a growing array of network and cloud security tools, and users to wade through complex multistage workflows to gain access. Instead, BastionZero provides a frictionless solution for authentication, authorization, auditing and access to targets in any cloud or datacenter. BastionZero has native support for remote shells, SSH, webservers, databases, Kubernetes and can secure any client-server application through its service.



BastionZero's Design Philosophy

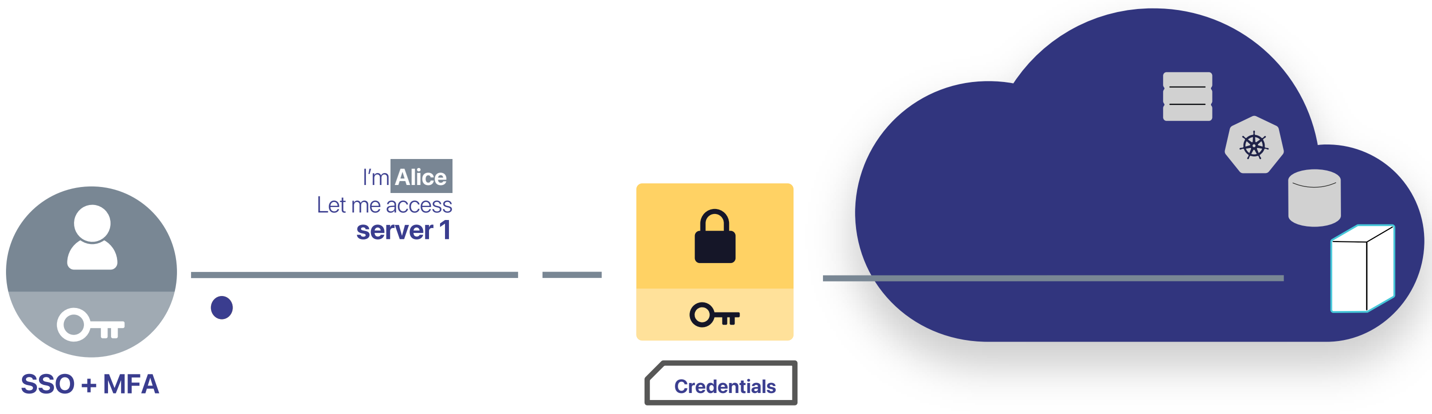
Reducing trust in the user.

BastionZero allows engineering and security teams to embrace the principle of least privilege. In recent years, it has become a best practice to limit the amount of trust placed in a given user. For example, instead of trusting Alice with a long-lived SSH key that grants her root access to a sensitive server, an organization might use a privilege access management (PAM) system to grant Alice a server credential that expires a few hours later. Instead of trusting Bob to hold a sensitive database password on her laptop, Bob might instead log into a bastion host that stores that database password, and connect into the database from the bastion, without ever seeing the database password.

Adversaries regularly steal user credentials in order to attack organizations. Examples abound. In 2020, PAN Unit 42 found that the number one initial attack vector for ransomware was through Windows RDP services. In 2021, there were gas shortages because Colonial Pipeline was breached by an adversary that stole an old VPN password. In 2022, top technology companies like Uber and Dropbox were breached because an individual user's login credential was compromised. In 2022, private health information of millions of Australian citizens was stolen due to a compromise of a Medibank employee's credential.

Reducing trust in highly-privileged third parties.

BastionZero also solves a key problem that is often forgotten in the rush to limit the trust placed on the user. Credentials have to be stored somewhere. Where should they be stored? You can store credentials on a bastion host, or in a secrets vault, or in a PAM or in a proxy. You can control access via Single Sign On (SSO). You can set up certificate authority that grants short lived credentials. But in all these solutions, the common thread is: there is a single highly-privileged authority that is trusted to store credentials and/or determine which user is granted credentials to access a given target. So what happens if that trusted highly-privileged authority is compromised?



Traditional zero-trust tools create a single point of compromise.
Credentials are stored centrally, rather than on the user's device

The risk here is real. When Colonial Pipeline was hacked, the adversary went after a trusted highly-privileged authority – the Active Directory server that was trusted with admin credentials to most of Colonial Pipeline's infrastructure. When Uber was hacked, the adversary went after a trusted highly-privileged authority – the secrets server that stored admin credentials to critical Uber infrastructure. When Medibank was hacked, the adversary went after a highly-privileged authority – the bastion host that held login credentials to a database containing private health information. As these incidents illustrate, the problem of securing infrastructure access goes well beyond trust placed to users. It also implicates the trust placed in the highly-privileged authority that controls access

To solve this problem, BastionZero has developed a unique trustless security model that uses two independent roots of trust that jointly control access to each target. Neither root of trust has credentials or unilateral access to any of the access targets; that way, a compromise of one of the roots of trust does not lead to a compromise of the organization.

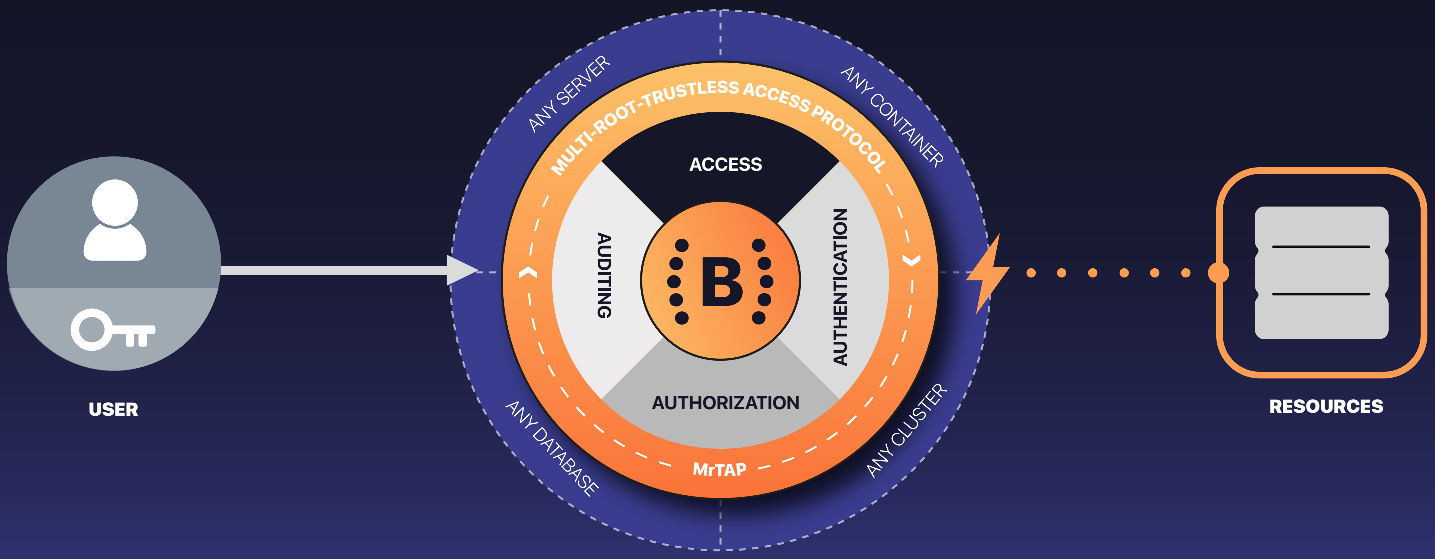
A cloud-native approach: log into targets, not networks!

BastionZero is a cloud-native platform that covers the 4As of remote access — access, authentication, authorization and auditing. The goal is to simplify user workflows and improve administrator's productivity by making it easy to enforce least-privilege access, to on and off-board users, and to capture information about exactly how each user interacted with each target.

With BastionZero, organizations don't need to have one set of solutions that control access to the network (VPN, firewalls, etc) and another set controlling access to the target (passwords, bastion hosts, PAM systems, etc). Instead, BastionZero offers a single cloud-agnostic platform that controls access from the users' machine all the way to each individual target. Instead of setting up ACLs and firewall rules, BastionZero uses a phone-home architecture along with a centralized policy manager to determine who has access to what target, when, and for how long.

A truly passwordless approach.

Instead of handing out SSH keys and database passwords, BastionZero provides passwordless access to targets. Instead of asking users to log into a VPN and maintain multiple passwords, SSH keys, kubeconfig files and database passwords, with BastionZero, users can access targets just by completing an SSO to the identity provider and an independent MFA to the BastionZero cloud service. Credentials are NOT stored in a vault or secret store, and further, credentials are NOT accessible to the BastionZero cloud service; instead, we use cryptographic techniques to split passwords and credentials across multiple roots of trust. Our cryptographic protocols work silently, under the hood, to completely eliminate passwords and credentials for individual targets.



Access.

BastionZero allows users to log into targets, not networks. To do this, BastionZero uses a phone-home architecture where both the user's client and the server-agent running on a target initiate outbound TLS connections to the BastionZero cloud service, and the cloud service then mediates their access; in this way, targets are accessible to users even if the targets have no open ports. Closing open ports limits an organization's attack surface, making their targets undiscoverable to roving malware or malicious port scanning attempts. The phone-home architecture also allows BastionZero to avoid the requirement for a VPN, to provide access to targets across any number of cloud accounts and datacenter environments with requiring organizations to maintain additional infrastructure or set up IAM roles or security groups.

Authentication.

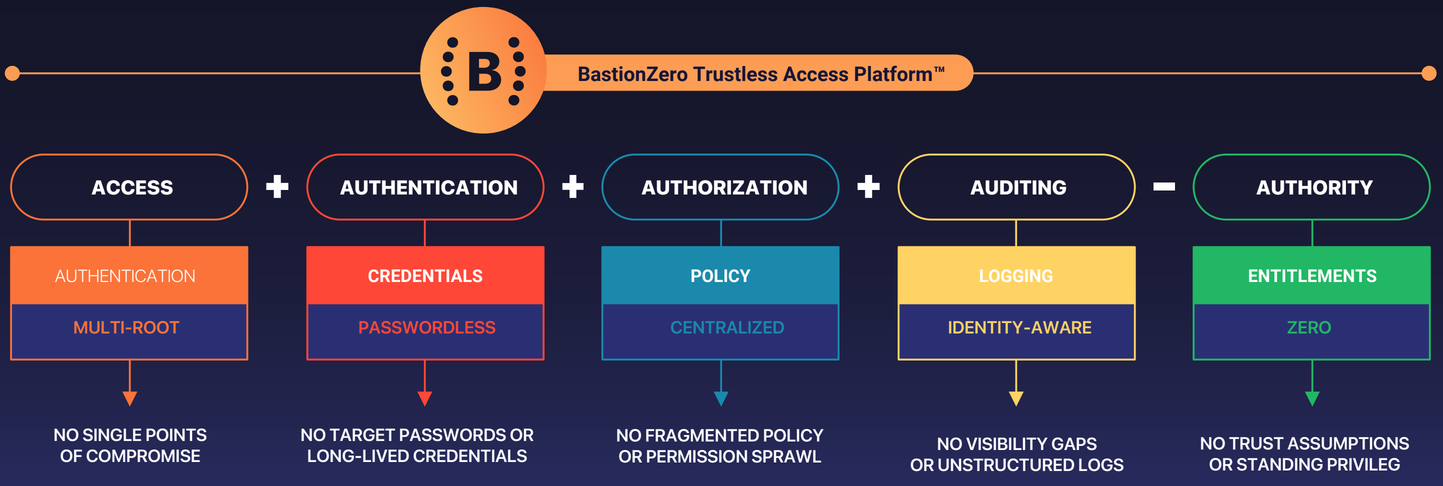
BastionZero uses multi-root authentication. Access gated behind Open ID Connect (OIDC) Single Sign On (SSO) to the organizations existing identity provider (as one of root of trust), and an independent Multi-factor Authentication (MFA) to the BastionZero cloud service (as another root of trust). Because the user authenticates to two roots of trust each time they log in, this limits the risk that compromised user credential OR even a compromised root of trust can lead to a breach of the relying party's infrastructure.

Authorization.

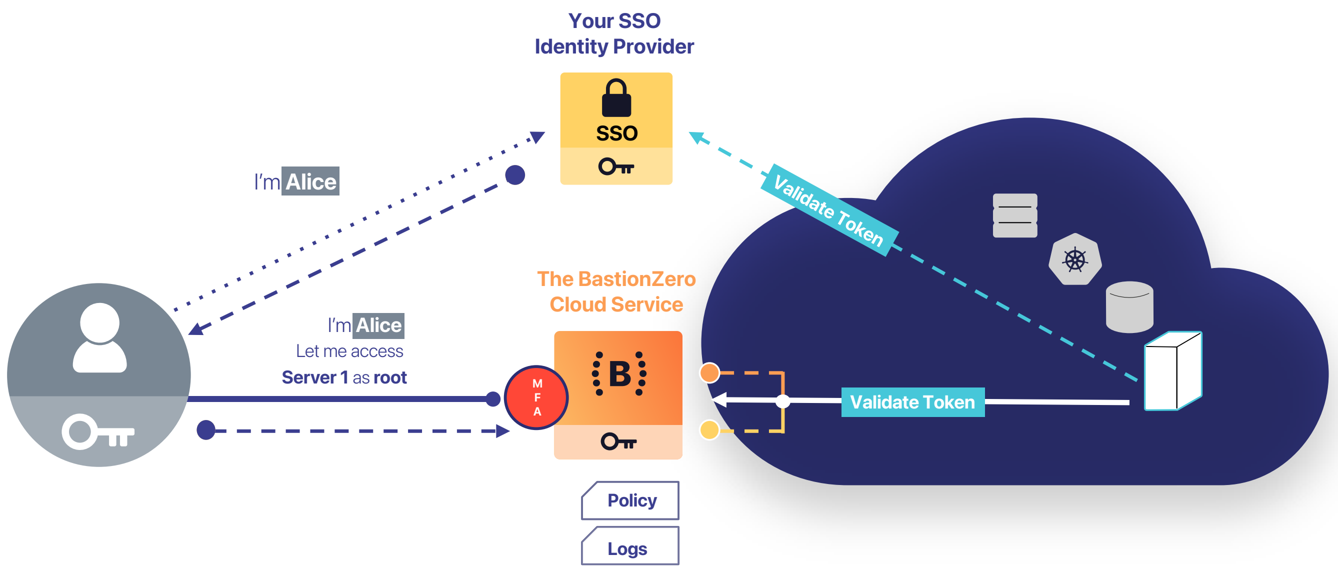
BastionZero's cloud service acts as a centralized policy manager that controls which user can log into which target as which role or account, when and for how long. Just-in-time (JIT) access can optionally be granted via an approval workflow that requires a human administrator to approve any given access request. Centralized policy management presents administrators with a frictionless solution that enforce the principle of least privilege, and to eliminate privilege creep (where everyone eventually gets access to everything).

Auditing.

Auditing is accomplished by the BastionZero cloud service, which logs and monitors access by introspecting on the connections made from users to targets. Auditing is identity-aware and allows auditors and security teams to disambiguate access through shared accounts. (For instance, BastionZero will capture that **alice@acme.com** logged into server1 as root along with the commands that Alice ran and her session recordings.) Our unique security model ensures that the cloud service can log sessions but cannot modify them, or inject commands, without the users permission or knowledge.


BastionZero Trustless Access Platform™
ACCESS
AUTHENTICATION
AUTHORIZATION
AUDITING
AUTHORITY
AUTHENTICATION
MULTI-ROOT
CREDENTIALS
PASSWORDLESS
POLICY
CENTRALIZED
LOGGING
IDENTITY-AWARE
ENTITLEMENTS
ZERO
NO SINGLE POINTS OF COMPROMISE
NO TARGET PASSWORDS OR LONG-LIVED CREDENTIALS
NO FRAGMENTED POLICY OR PERMISSION SPRAWL
NO VISIBILITY GAPS OR UNSTRUCTURED LOGS
NO TRUST ASSUMPTIONS OR STANDING PRIVILEG

Platform Architecture

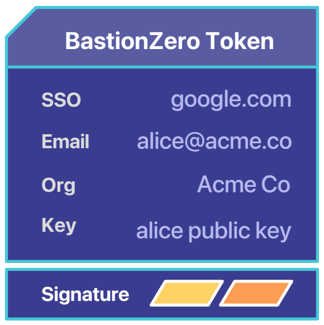


The key components of the BastionZero platform are the client, bastion and server-agent.

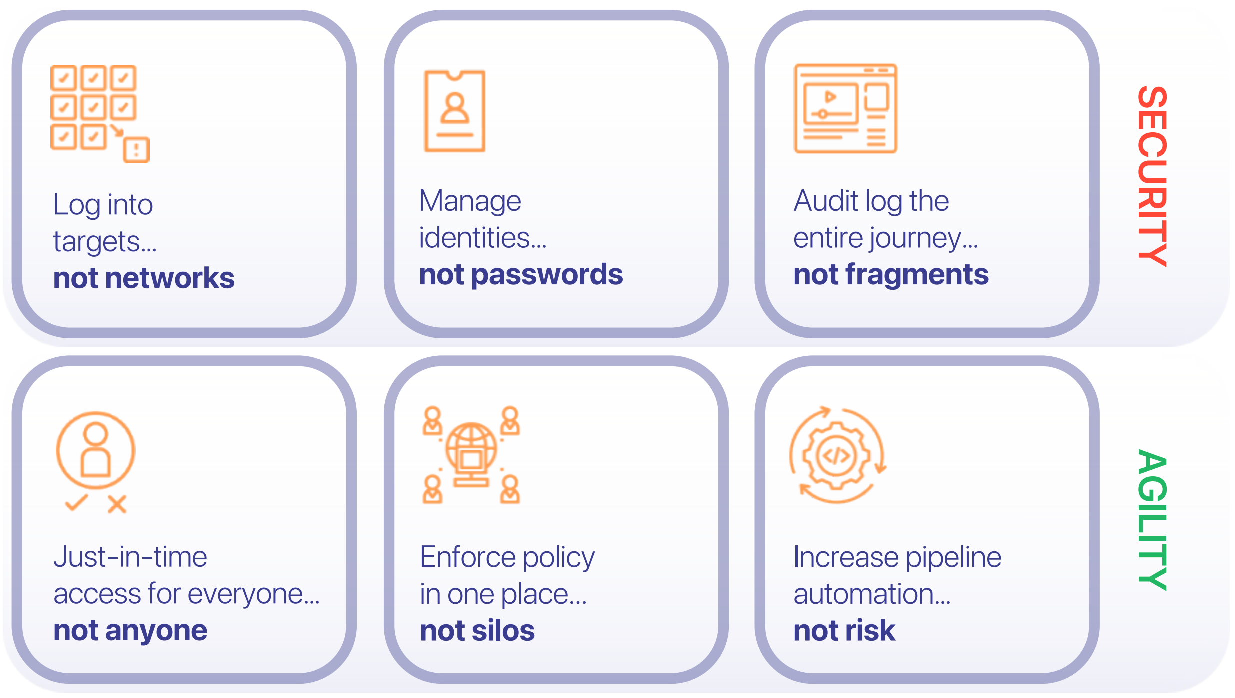
The client. The client is an open-source executable that runs on the user's machine and interfaces between the user and the rest of the system. The client can be installed on Linux, Windows Subsystem for Linux (WSL) and OSx via download or brew.

The server-agent. The open-source server-agent runs on remote targets, enforces simple access controls and interfaces between the host OS and the MrTAP protocol. The server-agent is available as a systemd agent for Linux or alternatively as a container that can be run on a kubernetes cluster.

The bastion. The bastion is a cloud service (aka, bastion-as-a-service) that performs centralizing policy management and audit logging. The bastion is responsible for receiving and delivering messages sent between client and server-agent. The bastion runs a web application and API which allows administrators to set policy, view and download logs. It is implemented as a globally-distributed micro-service architecture managed via Kubernetes with auto-scaling workers to handle many concurrent authenticated channels. To ensure low latency, the bastion is currently staged in the multiple regions, including the US, Tokyo and Frankfurt.



BastionZero Capabilities


Log into targets... not networks
Manage identities... not passwords
Audit log the entire journey... not fragments
SECURITY
Just-in-time access for everyone... not anyone
Enforce policy in one place... not silos
Increase pipeline automation... not risk
AGILITY

Simplified login via SSO+MFA. Users can access targets without disrupting their usual workflows simply by completing an SSO + MFA. This simplified workflow eliminates the need to log into a VPN and then using a separate system (e.g. password manager / ssh keys / PAM) to log into a target. It also reduces the number of tools an administrator needs to maintain, and our passwordless approach limits that number of keys an administrator needs to rotate.

Cloud-native autodiscovery and phone-home architecture. Administrators can save time by avoiding manual configuration of targets. Instead, targets can be provisioned via automated workflows and autodiscovered when the server-agent on the target phones home (via TLS) to the bastion (i.e. BastionZero's cloud service). When phoning home, the server-agent provides the target's name and 'environment' (the target's group); this information is set via a configuration file on the server-agent and can be provisioned using modern infrastructure automation tools. This phone home approach means that the target can connect to the bastion from any cloud account or datacenter environment, as long as the target can make an outbound connection to the Internet; no VPN, public IP or DMZ configuration is required. Messages sent between a user's client and a server-agent are sent via (a) a TLS websocket connection initiated by the client to the bastion and (b) a separate TLS websocket connection from the server-agent to the bastion.

Cloud-agnostic centralized policy. BastionZero simplifies the provisioning of least-privilege access by centralizing policy configuration at the bastion. Agile policies can be established by mapping from users and/or user groups to targets and/or environments (aka target groups); this allows for easy on-boarding and off-boarding of users, and supports ephemeral infrastructure and auto scaling groups. Administrators set policy at the bastion via APIs or a web interface (see Fig 2).

Under the hood, this policy engine is built on OPA (Open Policy Agent) [2], providing a common and expressive policy language. Policies, as exposed to users, take the form of subject, verb, resource. The subject can be one or more users (from the IdP) or groups (from the IdP), the verb is a set of actions the users and/or groups are allowed to take, and the resource is one or more groups of targets they are allowed to take that action on. For instance the policy in Fig 2 says "this set of users are allowed to open a shell/ssh-tunnel/transfer_files as a linux user named "centos" on all hosts in dev environments".

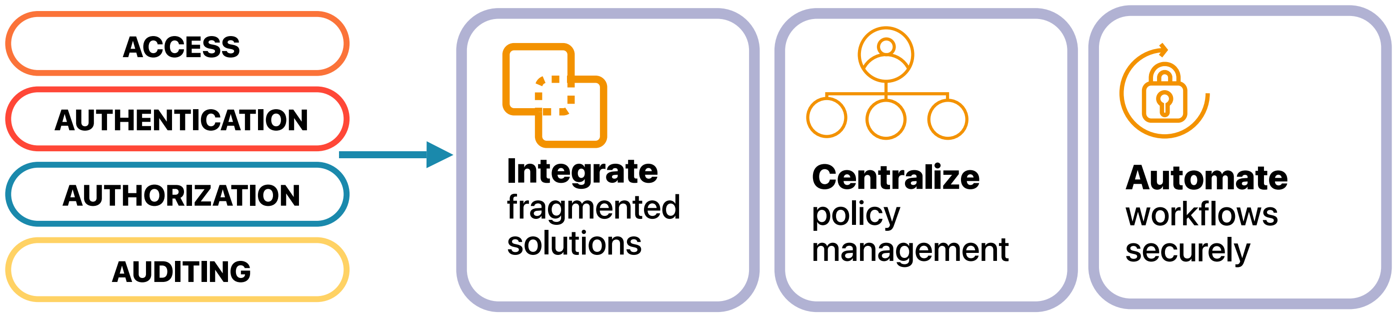
BastionZero also has just-in-time policies that allow administrators to grant temporary access to any target for users or groups within their organization. Granting temporary access can be done: (a) through auto-approval, or (b) through manual approval by a human administrator. Thus, one could write a policy that says"the engineering group of users is allowed to open a connection to k8s-cluster-prod-us-east1 as the cluster-admin user for one hour upon receiving approval from an administrator".

Centralized audit logs that bridge the identity gap. In many organizations there is a gap between user identity in the IdP (e.g., alice@example.com) and users on individual targets (e.g., the 'root' on a linux server, or cluster-admin on a Kubernetes cluster). Organizations might share credentials for different linux users (eg root, dev, test, secops) across multiple humans or service accounts. So how do you map user identities (alice@example.com) to users on targets (root on a linux server)? Alice, Bob and Eve can all have the authority to spawn an SSH tunnel under the dev role user on a host, but how do you determine which of them actually did?

BastionZero solves both of these problems for Kubernetes, linux shells and SSH/SSH tunnels. The bastion policy allows enforcing which users (OIDC identities e.g., Alice@example.com) can assume which role users in: Kubernetes, linux shells and SSH/SSH tunnels. Additionally the logs that we record contain both the user (OIDC identities) and the role user, thus bridging the identity gap.

Trustless access. BastionZero's trustless access model eliminates the security risks typically associated with controlling access via a cloud service. We introduce the cryptographic MrTAP protocol, that protects the communication from the client to the server-agent against a compromised bastion. With MrTAP, once the user is authenticated, and if policy allows, an authenticated channel is created from the client to the server-agent. (The authenticated channel is established once the user has authenticated to both roots of trust — the IdP and the bastion.) The channel is tunneled inside the client → bastion websocket and the server-agent→bastion websocket. Digital signatures are used to ensure that the bastion cannot violate the integrity of messages sent via this channel. The entire implementation of the MrTAP authenticated channel is open source and can be viewed by inspecting the client and server-agent open-source repositories.

A truly passwordless approach. As part of our commitment to trustlessness, the bastion does NOT hold credentials, passwords or secrets that allow it to access a target. This ensures that the bastion does not have privileged access to a target and prevents it from becoming a single point of compromise. In fact, we completely eliminate the distribution and storage of credentials, which also improves administrator productivity by reducing the workload around secret distribution, revocation and rotation.


ACCESS
AUTHENTICATION
AUTHORIZATION
AUDITING
Integrate fragmented solutions
Centralize policy management
Automate workflows securely

Trustless access

Let's take a deep dive into BastionZero's unique trustless security model. Trustlessness ensures that no part of BastionZero system becomes a single point of compromise; that is, that a compromise of one element of the BastionZero platform does not lead to a wholesale compromise of an organization's targets. BastionZero does this by introducing two independent roots of trust for access: (1) the bastion cloud service and (2) the organization's Identity Provider (IdP). Trustlessness means that neither the IdP nor BastionZero's bastion cloud service become a single point of compromise.

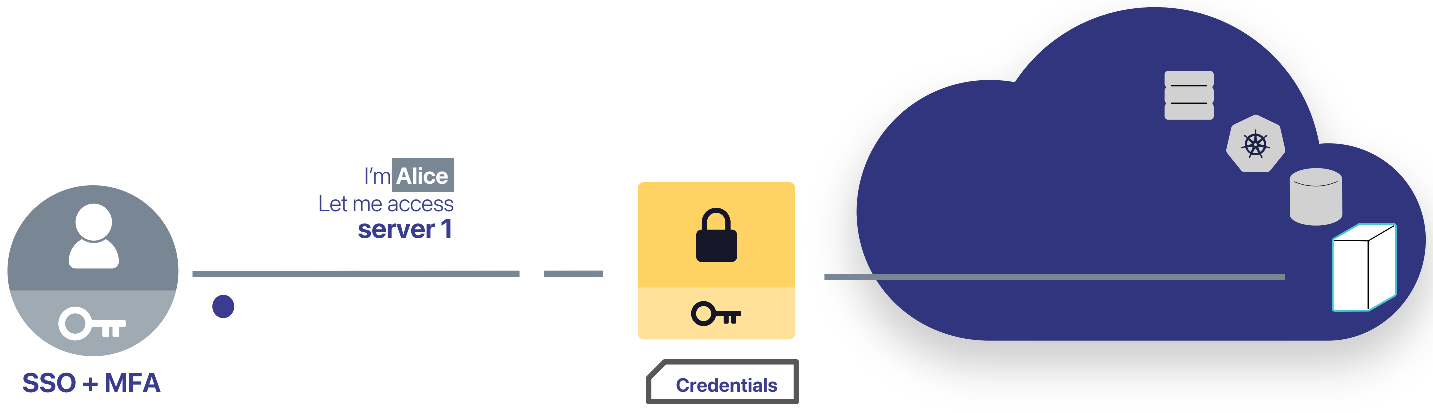
We start by reviewing the classic security model for zero-trust access, highlight key problems with this model, and then discuss trustless access and how BastionZero achieves trustless access.

The problem with classic zero-trust access.

Classic zero-trust. Zero-trust access boils down to ensuring that users do not hold long-lived keys, credentials, or tokens. Zero-trust access also requires that the user's network location (e.g. the fact that they are behind a VPN) is not used as the sole factor to determine whether or not they have access to a target. Instead, each time a user's client wants to access a target, the client must first prove its identity e.g., via IdP authentication and/or MFA. The client is then issued a short-lived token that allows for access until it expires. The use of short-lived tokens reduces the risk that a compromise of the client leads to a compromise of infrastructure targets.

For example, in 2001, the FluffyBunny hacking group replaced SSH clients with a malware that exfiltrated long-lived SSH credentials and then later used those credentials to log into targets, execute a local privilege escalation attack, and repeat. A zero-trust approach would have this limited attack surface because even if the attacker could steal short-lived tokens, these tokens may expire before the attacker could use them.

Multi-Factor Authentication (MFA) plays an important role in any zero-trust approach. MFA ensures that if the user's IdP username and password are compromised, their authentication remains secure as long as their MFA is not also compromised. MFA would have prevented the FluffyBunny attack because even if the attacker had stolen a victim's username and password, the attacker would also need to compromise MFA authentication in order to impersonate the victim.



The problem with traditional zero-trust. A traditional zero-trust approach usually eliminates long-lived user credentials in favor of single, centralized privileged root-of-trust that grants short-lived credentials to users. (Popular approaches include putting all credentials in a secrets vault or a PAM, setting up a certificate authority, or controlling all access via a single centralized IdP.) However, creating a single privileged root-of-trust also creates an attractive single point of compromise for attackers. As an example, in late 2020, the SolarWinds attackers exploited this type of architecture to compromise the IdP in their victim's networks, and then used the compromised IdP to issue tokens to themselves that allowed them to access any target in the victim's infrastructure. The attack technique of compromising an IdP to forge tokens is sufficiently common to motivate MITRE to assign it with an attack technique ID. In 2022, Uber was breached by an attacker that compromised its centralized PAM
and secret store.

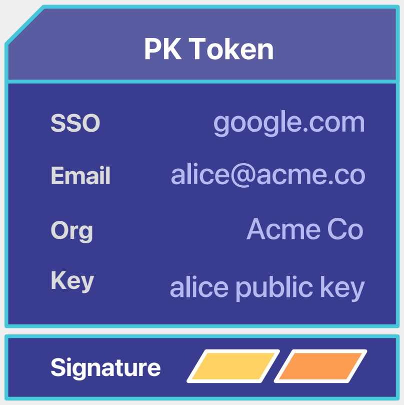
Importantly, MFA alone does not prevent these attacks. In most commercial products that use MFA, the MFA is checked by the centralized root-of-trust (e.g., the IdP). But if the centralized root-of-trust is compromised, it does not matter if it is checking MFA because the attacker will just ignore the MFA and issue tokens for whatever target it wants to attack.

How BastionZero achieves trustless access

Trustless access still requires a user to prove its identity before it accesses a target. However, a trustless access does this without introducing a single point of compromise. Instead, BastionZero uses multiple independent roots-of-trust for authentication.

The first root-of-trust is the organization's IdP, which authenticates the user via SSO, MFA and other mechanisms (e.g., device context). The other root of trust is run as a subsystem of the bastion, which authenticates the user with an independent MFA. The independent MFA performed against the bastion cloud service protects against a compromised IdP.

The PK Token. To access a given target, the client must present the target with token which we denote as the PK Token . This token acts almost like a certificate that attests to the user's short lived public key, pku. The PK Token is digitally signed both by the IdP and the bastion cloud service. The two signatures on this token are validated directly by the server-agent on the target, using the public key of the IdP and the public key of the bastion. It is very important that the server-agent independently checks these two signatures against the two independent roots-of-trust (rather than just blindly trusting the bastion to signal when a user should be allowed to access the target). The two independent signatures that are validated against the two independent roots-of-trust is a key factor in eliminating the single point of compromise created by traditional zero-trust solutions.



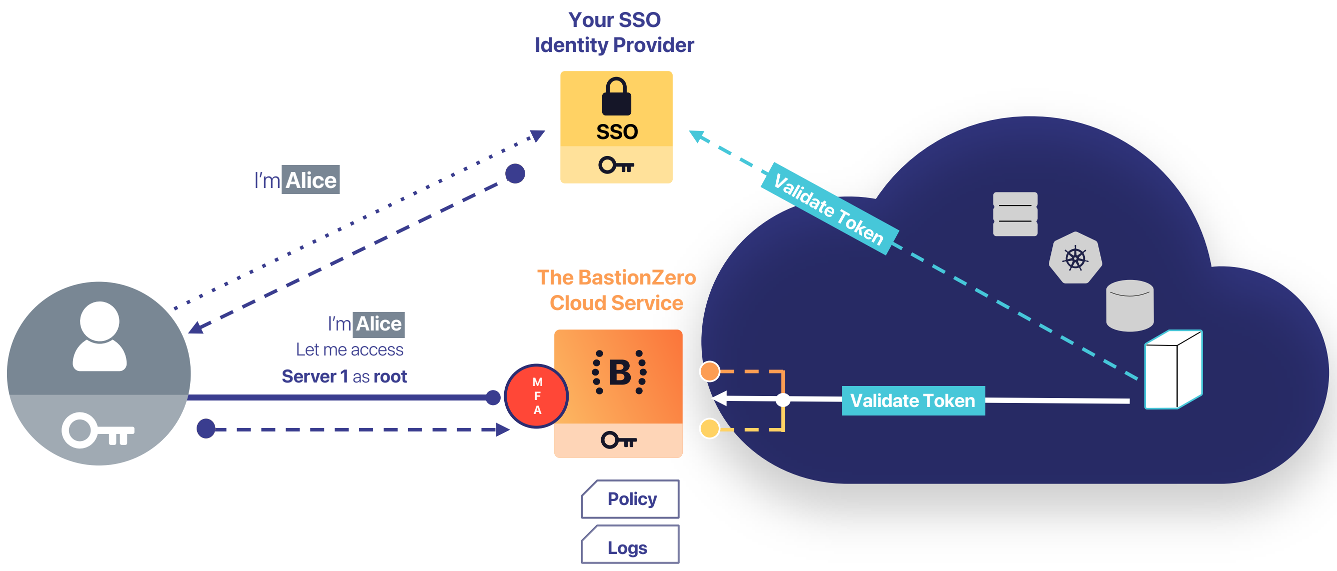
No more bearer tokens, sign your messages! A common pattern in zero-trust systems to to give user short-lived bearer token (e.g., a JWT) that the user then repeatedly presents to multiple different targets in order to gain access. A bearer token is essentially a token that contains a secret; that secret proves that the user is authorized to gain access to a target. Bearer tokens require a choice between two evils: (1). if messages are sent in the clear, then a compromised intermediary can tamper with the messages sent by the holder of the bearer token, or to steal the bearer token itself and use it to impersonate the user.

Alternatively, (2). messages could be encrypted to protect the bearer tokens (thus also encrypting the bearer tokens), but this comes at the cost of allowing intermediaries to perform security-critical functions like logging and inspecting messages. Because our cloud bastion must not be a single point of compromise, and because it must also log and enforce policy, neither of these options are acceptable. Instead, we use digital signatures to sign and authenticate the very commands and communication content that users are sending and receiving from infrastructure targets.

There are two key virtues of cryptographic digital signatures. First, signatures allow the MrTAP bastion to read messages without granting the bastion the ability to tamper with the messages. This follows because signatures are cryptographically bound to the message being sent, while bearer tokens are not. Second, with signatures, the user's authentication secret key sku never needs to be sent outside of the client. With bearer tokens, the secret is necessarily exposed in the bearer token itself. Signing messages creates an authenticated channel that prevents bastion from impersonating the user.

Identity, policy and access control. Identity is authenticated and enforced at the server-agent and bastion based on organizational membership, groups and other claims made in the PK Token and approved by the root of trusts. To prevent a compromised bastion or IdP from impersonating a user, we require the server-agent on the target to validate every connection made by a user using the PK Token, which contains a signature from both root of trusts.

Why it works. To constrain a compromised bastion from compromising the security of the server-agent, the server-agent validates the PK Token against the two roots-of-trust. Because the bastion can not authenticate over OIDC as a user of that organization, it follows the bastion can not generate a valid PK Token which attests to membership in that organization. The bastion can learn a user's PK Token, but learning a PK Token does not provide access unless you can also control the key pair (sku, pku) certified in the PK Token (so that you can use the secret key to generate a signature. Securityi is maintained because the bastion never has access to the user's secret key sku.



Usecases

BastionZero supports access to kubernetes clusters, starting and reattaching to interactive shell sessions, SSH connections/tunnels and TCP, HTTPS or database connections. The MrTAP protocol provides both a communication channel to transport these protocols to targets in private networks and an authentication layer to ensure only authorized parties can communicate with

these targets. Because the MrTAP bastion can read and understand these protocols, the bastion can provides application-specific logging and policy enforcement.

Kubernetes access

BastionZero supports access to kubernetes using only the kubernetes API, which makes it completely cloud-agnostic and interoperable with both managed and unmanaged kubernetes clusters in any cloud or datacenter. Our approach eliminates passwords and long-lived keys for kubernetes authentication, as well as the need to manage kube-config files. It also reduces attack surface because kubernetes APIs no longer need to be exposed to the public internet. (This is a serious problem; in 2022 a study found over 300k kubernetes APIs exposed to the public internet.) Our approach allows users to use their existing tooling to
interact with the cluster (e.g., kubectl, k9s or lens) while the MrTAP protocol works silently in the background.

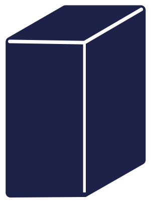
Automated deployment is supported, because installing BastionZero on a cluster only requires an administrator (or provisioner) to add a bastion generated helm chart or yaml file to the kubernetes cluster config. This file instructs the cluster to create a new container running a server-agent on the kubernetes cluster and then binds this server-agent to the control plane of the cluster. The server-agent in the container then phones home is autodiscovered by the bastion. Once the user logs in with the client the client creates a (ephemeral) local kubernetes config file instructing the user’s kubernetes tools to route their API calls through the client. The client wraps them in an MrTAP authenticated channel and sends them through the bastion to the server-agent.

zli kube. A common problem with logging in kubernetes is that when a user’s runs a command (e.g., ‘kubectl get pods’) the kubernetes cluster only sees the API calls, but not the command that generated those API calls. (A single kubectl command can generate between 10-200 API calls, making it difficult for administrators to glean useful information from the resultant logs.)

To address this issue we have developed a tool and packaged it with the MrTAP client which wraps the kubectl cli, capturing the user’s command, and associating that command with the resulting API calls in our logs. This tool, zli kube, runs the same set of commands as traditional kubectl, but records the english text of the command in the BastionZero logging service.

Server Access

Our server-agent can be installed on Linux targets via package managers (yum or apt) or using an ansible or bash script. The server-agent is designed so that new targets can be provisioned and phone home to the bastion without any manual intervention. This supports common infrastructure operation patterns like autoscaling groups of servers, or the spinning down of test servers at night and the spinning them up again in the morning.



Interactive shells. A user can start interactive shell sessions with the target, if allowed by policy. The user interacts with a terminal emulator on client, and her keystrokes are signed and sent over a MrTAP authenticated channel. The output from the shell is returned to the user via an auxiliary output channel.

The bastion can log all keystrokes sent, and record the terminal output. The bastion then build a searchable command history of all commands entered by all users (like '.bash_history' but searchable). By default the bastion only records the commands a user entered. An admin can set a policy for a particular host that will record all shell input and output aka, session recording.

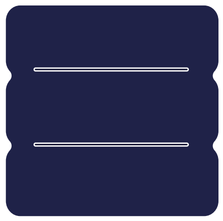
SSH and SSH Tunnels. SSH and SSH tunnels tend to be central to developer and operator workflows and scripts that encode venerable institutional wisdom. There are several reasons why the ability to create SSH tunnels in BastionZero is so powerful.

- Our approach improves administrator productivity by eliminating SSH key management. Instead, SSH keys are created on the fly, and authorized via MrTAP PK Tokens and revoked when the SSH tunnel is torn down.
- BastionZero also simplifies SSH configuration management, because the client autogenerates a local SSH config file on the user's machine that specifies the SSH hosts that should be accessed via BastionZero.
- BastionZero also reduces attack surface, because our phone home architecture allows organizations to close all the open SSH ports on their targets, while still using SSH. The SSH server, running on the same host as server-agent can optionally be configured to only accept SSH connections from localhost, so that even hosts on the same private network would not be able to connect to the SSH server.
- The server-agent does not interfere with any existing SSH clients running on the target, and can run alongside an existing SSH client without disrupting it.
- SSH tunnels allow BastionZero to support communication that is confidential from the bastion (i.e., the bastion cannot introspect on this communication), while preserving its other multi-root zero-trust security benefits.
- By suppporting SSH, BastionZero can tunnel any protocol that can be tunneled through SSH, i.e., any protocol built on TCP.

  Whenever an SSH tunnel is created in BastionZero, the client and server-agent establish a one-time SSH key for that tunnel using the MrTAP protocol. To do this, the SSH key is generated by the client and sent over the MrTAP authenticated channel to the server-agent. The SSH tunnel is encapsulated in an HTTPS websocket session from the client to bastion and a separate HTTPS websocket session from the server-agent to the bastion.

Web Servers and TCP.

In addition to SSH tunnels MrTAP supports proxying HTTPs, database and TCP connections from a localhost socket on a user's machine through MrTAP authenticated channel to server-agent and from server-agent to the final destination. This way, a sensitive web application or databases can be hosted in a private network, and access can be controlled by fine-grained policy set at the MrTAP bastion. Unlike most VPNs, our access policy is granular to users and services.

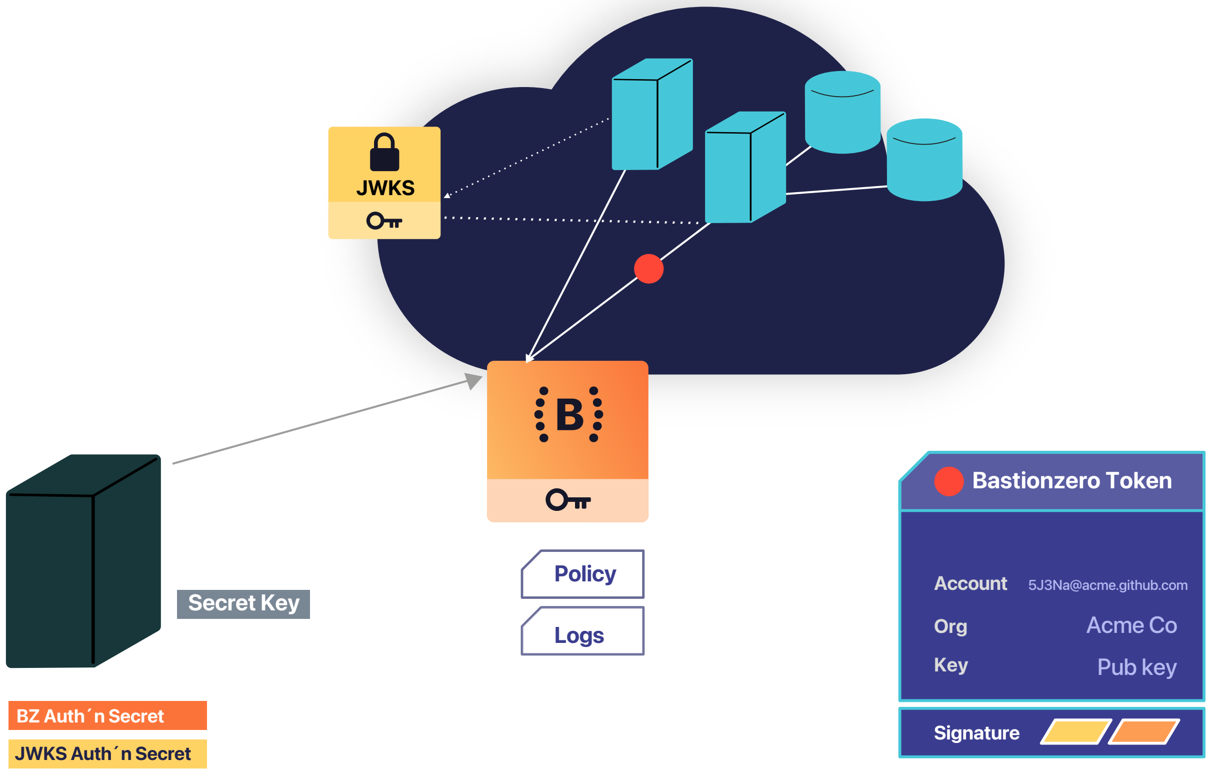


Service accounts.

Infrastructure access isn't just for humans; it's also crucial for scripts and build processes and other automated processes. And the problem of managing access for these automated processes is just as important as the problem of managing human access; we don't want to be handing out long-lived SSH keys to CI/CD runners and Jenkins servers, for the same reasons we don't want to give them out to humans. This is where BastionZero's service account comes in. In a traditional solution, a service account is given a single a long-lived credential that it stores and uses for access, which goes against the ethos of a zero-trust security model. BastionZero sidesteps this by using a multi-root authentication model.

To support trustless access, BastionZero service accounts authenticate to two roots of trust: one root of trust is the the bastio and the other one is any independent OpenID provider. To authenticate to BastionZero, a service account must hold two secrets: (1) the secret used for MFA to BastionZero and (2) the private key corresponding to the public key in the JWKS url. Authentication to two independent roots of trust stops either root of trust becoming a single point of compromised, supporting trustless access. Also, if these two secrets are stored in different locations (e.g., one in a TPM and the other in a secret vaults), we also get multi-factor authentication for service accounts.

BastionZero service account support several exciting usecases, including eliminating SSH deploy keys for CI/CD runners, and setting up just-in-time polices that limit the access of a CI/CD runner to known time windows while providing robust audit logging capabilities.

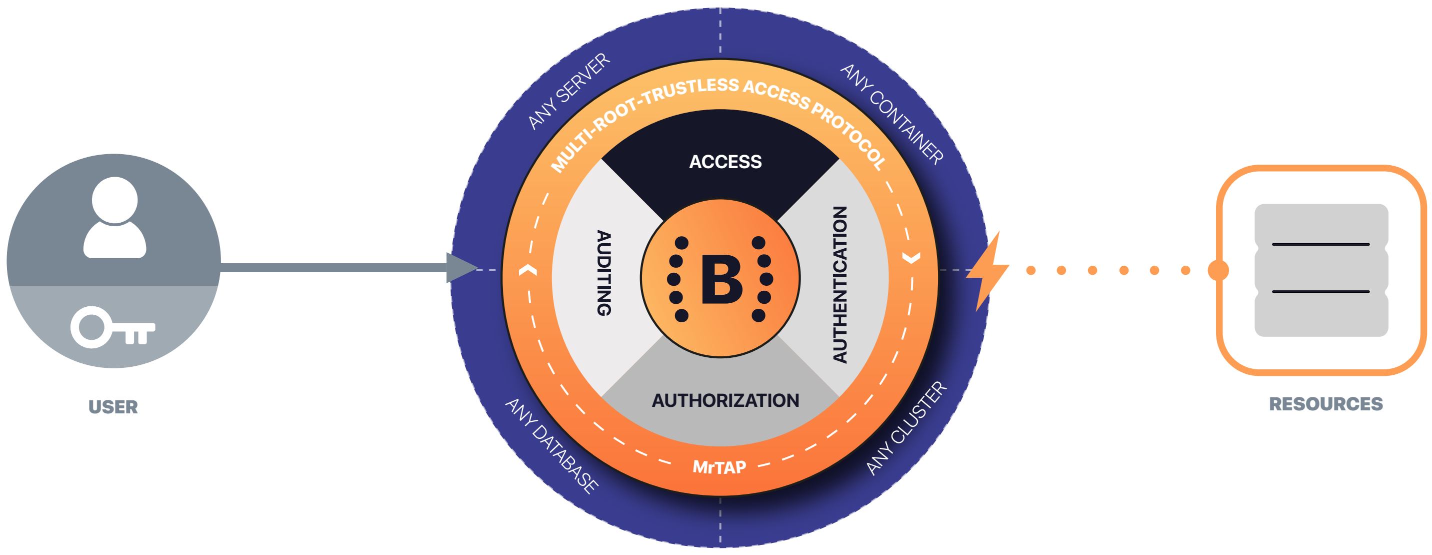


Conclusion

BastionZero supports access to infrastructure by consolidating fragmented solutions for access with a future-proof trustless security model. The platform increasing administrator's productivity by eliminating passwords to individual targets, and increases user productivity by providing a frictionless access experience that doesn't disrupt existing user workflows.

At the heart of BastionZero platform is the cryptographic MrTAP protocol. MrTAP that uses multiple roots of trust to eliminate passwords and long-lived credentials and to provide trustless access. The MrTAP cryptographic protocol enables a user to have their public key certified by an OIDC Identity Provider (as one root of trust), and then have this certification cosigned by the BastionZero cloud service (as a second independent root of trust). That way, if one root of trust is compromised, the other root of trust is still able to protect the integrity of infrastructure access. While OIDC was not inherently designed to support this, the MrTAP protocol is compatible with all deployed OIDC Identity Providers. This OIDC-based certificate is used to authenticate and digitally sign all messages sent from the user via the bastion to a target (linux machine, kubernetes cluster, database). The protocol also allows the bastion cloud service to record audit logs of the actions taken by the user; thus, compromised targets need not be trusted to accurately record and send logs.

BastionZero allows users to log into targets, rather than networks. This simplifies and secures infrastructure by eliminating VPNs, bastion hosts, PAMs and SSH/kubernetes/database key management systems. Because the platform is cloud-agnostic, there is no need to set up IAM roles across different clouds and accounts or manage disparate access systems in different accounts and environments. Because the platform integrates with existing identity providers, it simplifies the process of on- and off-boarding users, and providing them with short-lived access to sensitive targets. BastionZero's rich identity-aware audit logs support an organization's compliance efforts and empower security and devops team to respond to security incidents and conduct forensic analyses.


USER
ANY SERVER
ANY CONTAINER
ANY DATABASE
ANY CLUSTER
MULTI-ROOT-TRUSTLESS ACCESS PROTOCOL
ACCESS
AUDITING
AUTHENTICATION
AUTHORIZATION
B
MrTAP
RESOURCES