PRESENTS

Vitess security audit

In collaboration with the Vitess maintainers, Open Source Technology Improvement Fund and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: June 5, 2023

Table of contents

Executive summary

In March and April 2023, Ada Logics carried out a security audit of Vitess. The primary focus of the audit was a new component of Vitess, VTAdmin. The goal was to conduct a holistic security audit which includes multiple disciplines to consider the security posture from different perspectives. To that end, the audit had the following high-level goals:

1. Formalise a threat model of VTAdmin.
2. Manually audit the VTAdmin code.
3. Manually audit the remaining Vitess code base.
4. Assess and improve Vitess's fuzzing suite.
5. Carry out a SLSA compliance review.

These five goals are fairly different. While they allowed the auditors to evaluate the security posture of Vitess from different perspectives, they also offered a level of synergy; Ada Logics found two CVE's during the audit which the threat model goal helped to assess. The threat model was also a force-multiplier for the fuzzing work that led to the discovery of a few missed edge cases when fixing the two CVE's.

The audit started with a meeting between Ada Logics, the Vitess maintainers and OSTIF. After that, all three parties met regularly to discuss issues and questions as they arose during the audit. Ada Logics shared issues of higher severity during the audit.

In this report, we present the work and results from the audit. The audit was funded by the CNCF who hosts Vitess as a graduated project.

Results summarised
12 security issues found
2 CVEs assigned
Formalisation of VTAdmins threat model
3 fuzzers added to Vitess's OSS-Fuzz integration

Notable findings

The most notable findings from the audit are "ADA-VIT-SA23-5, Users that can create keyspaces can deny access to already existing keyspaces" and "ADA-VIT-SA23-12, VTAdmin users that can create shards can deny access to other functions". These two issues allowed a malicious user to create a resource that would then subsequently disallow other operations for other users. For example, a user could create a malicious shard that would prevent other users from fetching or creating shards. The issues would disallow actions against other resource types as well, thus resulting in a denial of service attack vector. The issues were more significant for Vitess deployments that include the VTAdmin component, since a user with the lowest level of privileges in VTAdmin could cause denial of service for all other users in the deployment.
The root cause of the two issues were at the Topology level in Vitess.

Vitess created an advisory for each issue and assigned CVE's for both advisories:

ID	CVE	Severity
ADA-VIT-SA23-5	CVE-2023-29194	Moderate
ADA-VIT-SA23-12	CVE-2023-29195	Moderate

Project Summary

The auditors of Ada Logics were:

Name	Title	Email
Adam Korczynski	Security Engineer, Ada Logics	Adam@adalogics.com
David Korczynski	Security Researcher, Ada Logics	David@adalogics.com

The Vitess community members involved in audit were:

Name	Title	Email
Deepthi Sigireddi	Project Lead & Maintainer	Deepthi@planetscale.com
Andrew Mason	Maintainer	Andrew@planetscale.com
Florent Poinsard	Maintainer	Florent@planetscale.com
Veronica Lopez	Contributor	Veronica@planetscale.com
Dirkjan Bussink	Maintainer	Dbussink@planetscale.com

The following facilitators of OSTIF were engaged in the audit:

Name	Title	Email
Derek Zimmer	Executive Director, OSTIF	Derek@ostif.org
Amir Montazery	Managing Director, OSTIF	Amir@ostif.org

Audit Scope

The following assets were in scope of the audit.

Repository	https://github.com/vitessio/vitess
Language	Go, Typescript

The full Vitess repository was considered in scope, however the main focus of the audit was VTAdmin which is located at https://github.com/vitessio/vitess/tree/main/go/vt/vtadmin.

Threat model formalisation

In this section we outline the threat model of Vitess's VTAdmin component. We first outline the core components of VTAdmin. We then cover how it interacts with the internal components of Vitess. Next, we specify the threat actors that could have a harmful impact on a VTAdmin deployment. Finally we exemplify several threat scenarios based on the observations we made when outlining the core components and the specified threat actors.

We used the following sources for the threat modelling:

- Vitess's documentation including README files from the Vitess repository
- Vitess's source code at https://github.com/vitessio/vitess
- Feedback from Vitess maintainers

The threat model is aimed at three types of readers:

1. Security researchers who wish to contribute to the security posture of Vitess.
2. Maintainers of Vitess.
3. Users of Vitess.

We expect that the threat model evolves over time based on both how Vitess and adoption evolve. As such, threat modelling should be seen as an ongoing effort. Future security disclosures to the Vitess security team are opportunities to evaluate the threat model of the affected components.

Most compromises of VTAdmin have the goal of compromising the full Vitess deployment. As such, the threat model of a Vitess deployment and VTAdmin are closely aligned, but they are also different. Other components of Vitess have different attack vectors, threat actors and security designs. The threat model in this report is solely for Vitess's VTAdmin component.

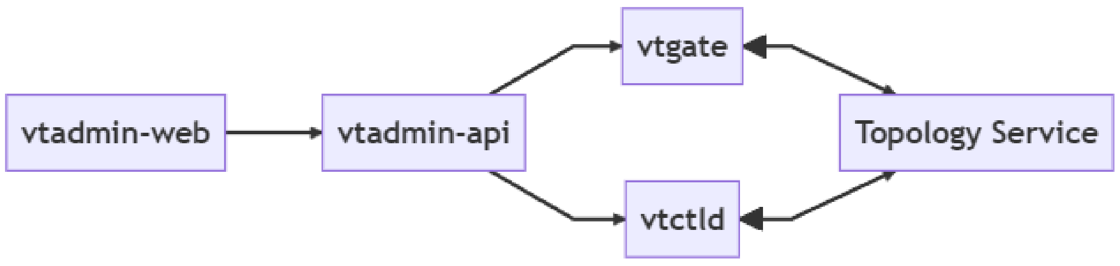
VTAdmin architecture

VTAdmin is a component for managing Vitess clusters. It is intended to be used by administrators, as the name suggests. As such, non-admin users should not be able to perform the actions that the admin users can.

VTAdmin consists of two components:

1. A web interface - VTAdmin-web
2. A server - VTAdmin-api

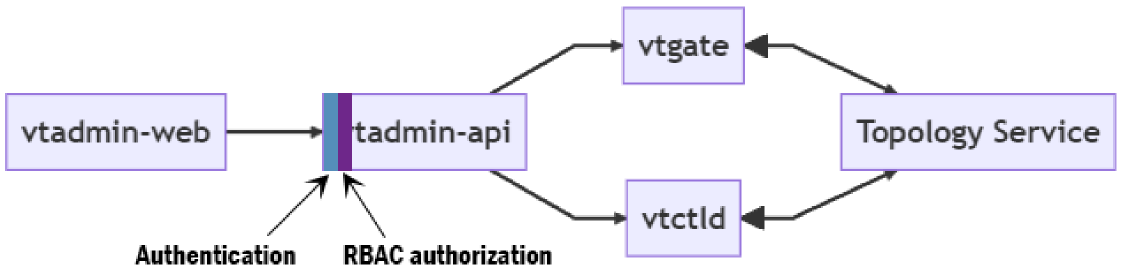
The web interface connects to the server which in turn forwards the requests to the Vitess internals:



From https://vitess.io/docs/17.0/reference/vtadmin/architecture/

Authentication and authorization

VTAdmin does two things when receiving incoming requests: 1) It first authenticates the request, and 2) it then checks the authorization level for the user sending the request. In VTAdmin, authentication is the task of obtaining the actor that is sending the request, and authorization evaluates whether the actor has permission to make the request. Vitess calls authenticated users “actors”. Once VTAdmin has obtained an actor from the incoming request, VTAdmin validates the actor against the RBAC. As such, the flow of handling the permissions of incoming requests looks as such:

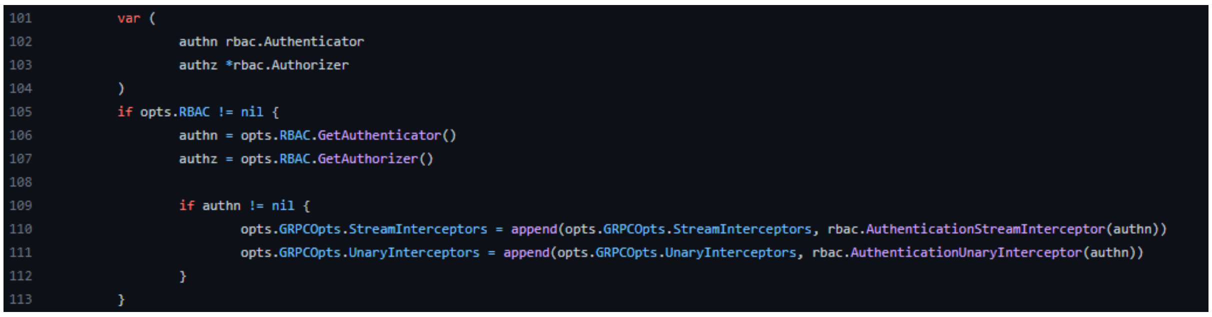


Authentication

Authentication in VTAdmin has the purpose of answering the question of *who* is sending a request. VTAdmin does not have a default authenticator, so users are required to implement their own via the Authenticator interface: https://github.com/vitessio/vitess/blob/da1906d54eaca4447e039d90b96fb07251ae852c/go/vt/vtadmin/rbac/authentication.go#L37. Vitess links to an example authentication plugin which is available here: https://gist.github.com/ajm188/5b2c7d3ca76004a297e6e279a54c2299. This example plugin extracts the actor from either the context of a request or from a cookie.

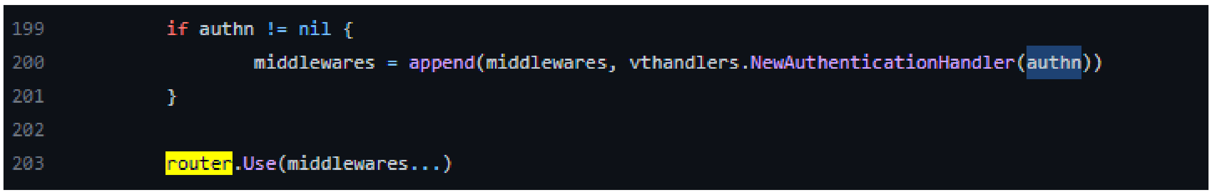
When a Vitess administrator adds an authentication plugin, VTAdmin-api adds it as a middleware at the http mux layer. VTAdmin-api does this in vitess/go/vt/vtadmin/api.go, when the routes are initialized:

First VTAdmin-api checks if the user has registered an authentication plugin:



```
	var (
		authn rbac.Authenticator
		authz *rbac.Authorizer
	)
	if opts.RBAC != nil {
		authn = opts.RBAC.GetAuthenticator()
		authz = opts.RBAC.GetAuthorizer()

		if authn != nil {
			opts.GRPCOpts.StreamInterceptors = append(opts.GRPCOpts.StreamInterceptors, rbac.AuthenticationStreamInterceptor(authn))
			opts.GRPCOpts.UnaryInterceptors = append(opts.GRPCOpts.UnaryInterceptors, rbac.AuthenticationUnaryInterceptor(authn))
		}
	}
```

And later, it gets added to the http mux layer:



```
	if authn != nil {
		middlewares = append(middlewares, vthandlers.NewAuthenticationHandler(authn))
	}

	router.Use(middlewares...)
```

Authorization

Once a request has been authenticated, it can be authorized. In VTAdmin, authorization checks whether an actor can perform an action against a given resource. The logic is implemented here: https://github.com/vitessio/vitess/tree/main/go/vt/vtadmin/rbac.

VTAdmin checks RBAC rules in the route handlers with a call to `IsAuthorized`, for example:
https://github.com/vitessio/vitess/blob/da1906d54eaca4447e039d90b96fb07251ae852c/go/vt/vtadmin/api.go#L755

```
func (api *API) GetClusters(ctx context.Context, req *vtadminpb.GetClustersRequest)
(*vtadminpb.GetClustersResponse, error) {
	span, _ := trace.NewSpan(ctx, "API.GetClusters")
	defer span.Finish()

	clusters, _ := api.getClustersForRequest(nil)

	vcs := make([]*vtadminpb.Cluster, 0, len(clusters))

	for _, c := range clusters {
		if !api.authz.IsAuthorized(ctx, c.ID, rbac.ClusterResource,
rbac.GetAction) {
			continue
		}
```

```
                vcs = append(vcs, &vtadminpb.Cluster{
                        Id:   c.ID,
                        Name: c.Name,
                })
        }

        return &vtadminpb.GetClustersResponse{
                Clusters: vcs,
        }, nil
}
```

Authentication and authorization are done at the VTAdmin-api level, not VTAdmin-web; VTAdmin-web is merely a client. In other words, authentication and authorization are not enforced when using the web UI - VTAdmin-web - but when the web UI communicates with the server.

If a threat actor is able to perform an action that they have not been granted access to via RBAC rules, that is a breach of security. An RBAC permission should only allow a user to carry out the actions against the resources that match the RBAC rules specified by the cluster admin.

Authentication and Authorization threat scenarios

Having defined how authentication and authorization work in VTAdmin, we now enumerate a list of threat scenarios and risks concerning VTAdmin.

Users can claim to be a user that they are not

If users are able to claim to be someone they are not, they can launch a number of different attacks against the cluster. For example, by claiming to be a user with higher privileges, they are potentially able to elevate their RBAC permissions. Or the user could disguise themselves under the pretence of another user when performing reconnaissance against the cluster or exploiting a vulnerability.

Users can perform actions that they do not have permission to perform

VTAdmins RBAC has two main goals:

1. Users should be able to perform the actions that they have been permitted to.
2. Users should not be able to perform actions that they have not been permitted to.

The first goal is related to both the reliability of VTAdmin as well as its security posture; If a cluster admin has granted permissions to a user to perform an action against a resource, the user should not be prevented from doing said action. Issues with this goal is related to the reliability and not security of VTAdmin with one exception:

- If User A has permissions to perform an action but cannot perform it because User B has disabled this functionality for User A, and User B should not be able to disable this.

Goal 2 is fully related to the security posture of VTAdmin: If any user can carry out an action that they have not been granted permission to, then it is a breach of VTAdmin-api's RBAC.

An attacker could do this by using tilizing existing RBAC privileges for a given action and resource to obtain permissions to perform actions against resources that the attacker does not have permission to. For example, if a user is able to utilize `create` privileges to cause Vitess to delete a resource, the user has elevated their privileges. The root cause of such an attack scenario is likely to be an implementation error.

The role of VTAdmin and Vitess's attack surface

VTAdmin adds a new, more granular user access control than Vitess has previously had. In a deployment without VTAdmin, a user with permission to perform one action against one resource can perform all actions against all resources. VTAdmin introduces granular permission controls. This may cause users to over-permit access to keep permissions at the same level of simplicity - ie. allow users either full access or none. To this end, users should be well advised in maintaining a well-configured RBAC policy.

Threat actors

A threat actor is an individual or group that intentionally attempts to exploit vulnerabilities, deploys malicious code, or compromise or disrupt a VTAdmin deployment, often for personal gain, espionage, or sabotage.
We identify the following threat actors for VTAdmin. A threat actor can assume multiple profiles from the table below; For example, a fully untrusted user can also be a contributor to a 3rd-party library used by VTAdmin.

Actor	**Description**	**Have already escalated privileges**
Fully untrusted users	Users that have not been granted any permissions and that the Vitess cluster admin does not know the identity of.	No
Limited access users	Users that have been granted some RBAC permissions but not others. Note that this actor is always awarded and never obtained. For example, a fully untrusted user can seek	No

	to become a limited access user, but this would be a security breach performed by the fully untrusted user actor.	
Contributors to 3rd-party dependencies	Contributors to dependencies used by Vitess.	No
Actor with local network or local file access	An actor that has breached some security boundaries of the environment to get to the position of having access to the local network or file system.	Yes
Well-funded criminal groups	Organized criminal groups that often have either political or economic goals.	No

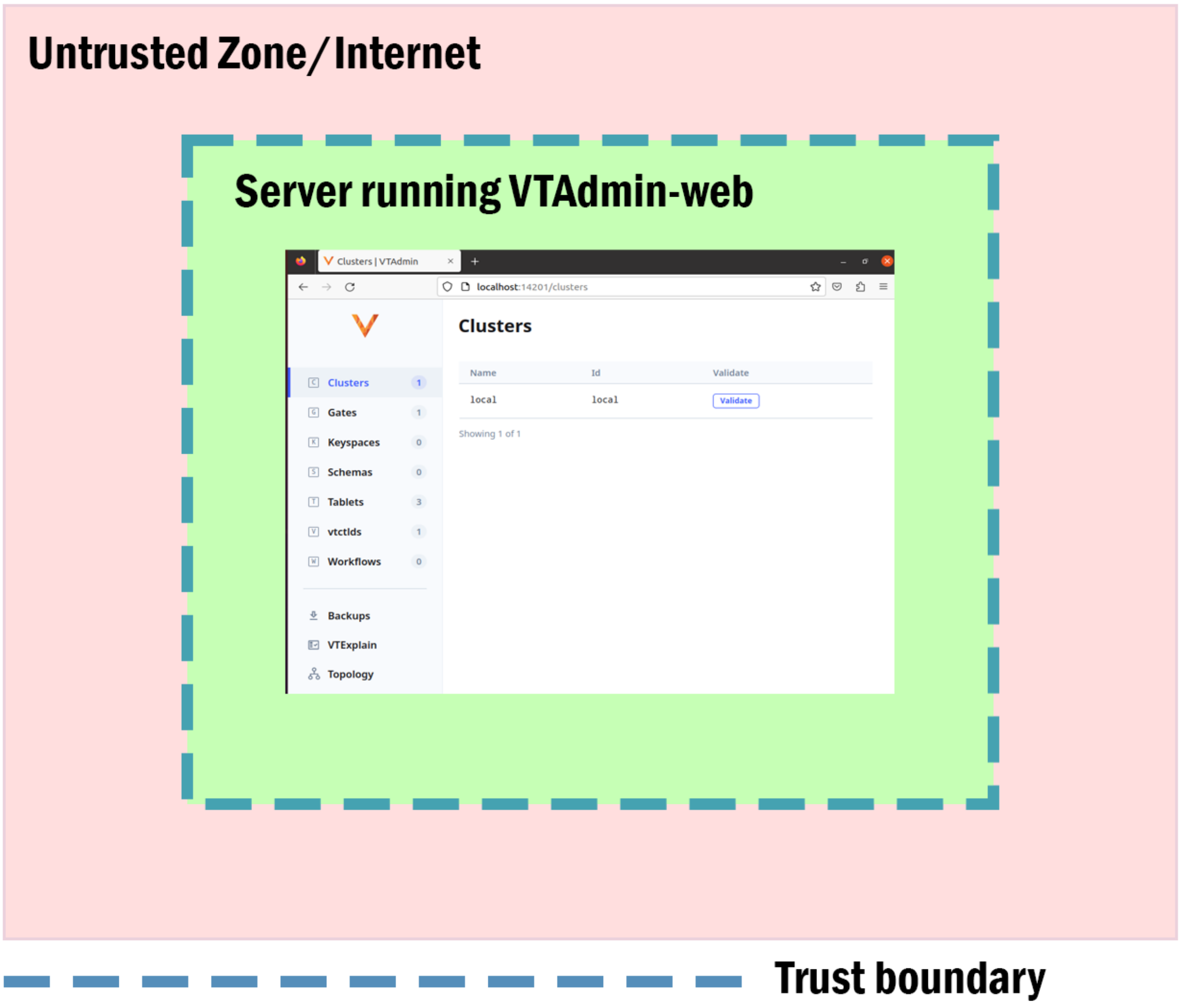
Trust boundaries

A software trust boundary is a boundary within a software system that separates trusted components and actors from untrusted ones. In this section we enumerate the trust boundaries for VTAdmin. We first consider the trust boundaries of VTAdmin-web and then of VTAdmin-api.

VTAdmin-web

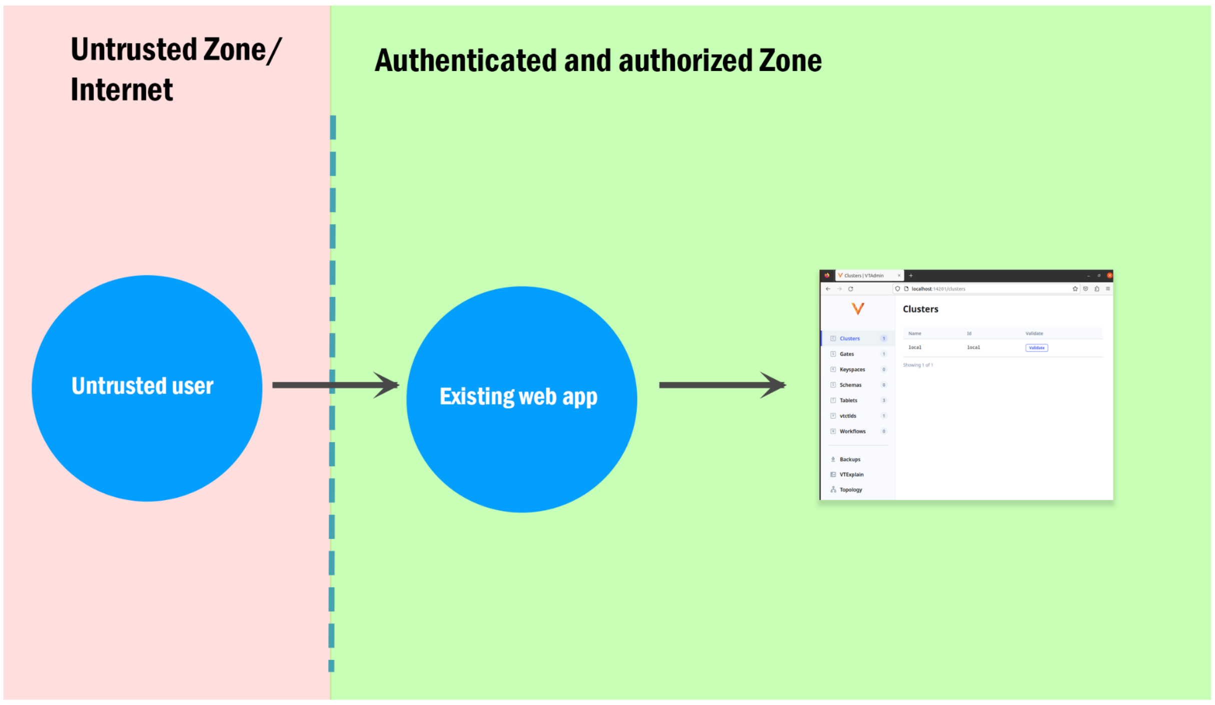
VTAdmin-web is meant to be deployed in a trusted environment, meaning that an attacker needs to compromise the security measures of the environment to gain access to VTAdmin-web. There are two security measures that an attacker can compromise, 1) the file system of a running VTAdmin deployment, and 2) an app that is responsible for authenticating the VTAdmin-web client.

File system
Trust increases when an actor obtains access to the local file system. An attacker with local access may be able to access VTAdmin-web. In this case, the trust boundary is the local file system.



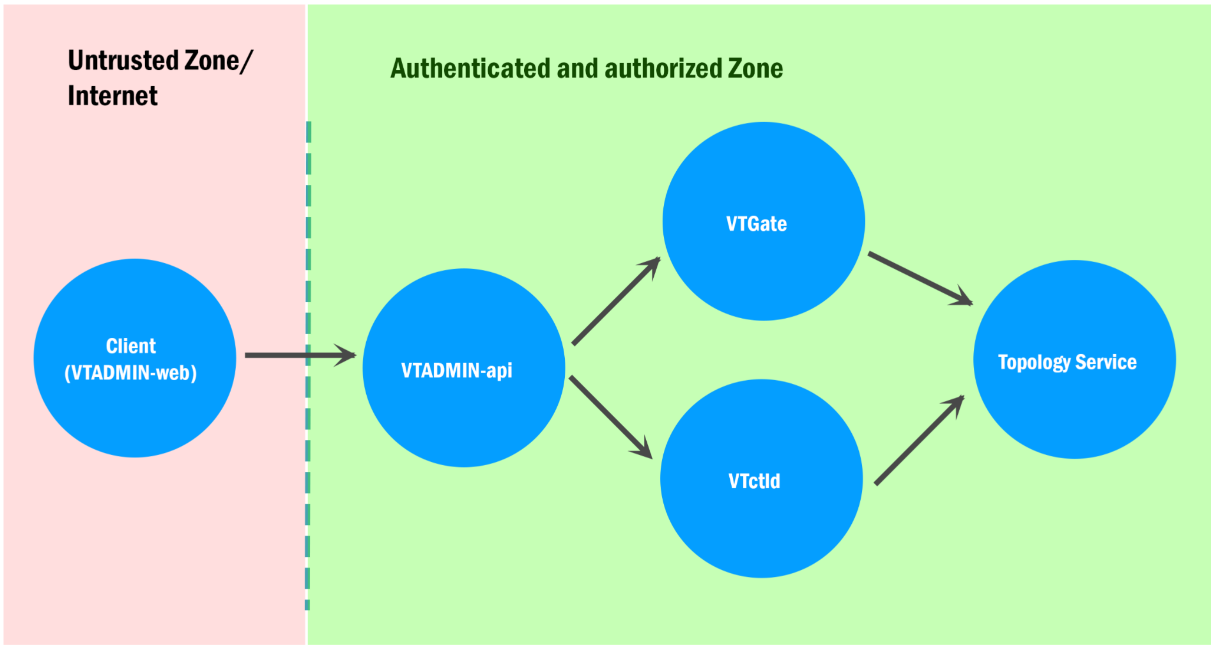
Compromising existing app

Another use case of VTAdmin-web is to integrate it into an existing web app, where the existing web app already contains its own authentication mechanism. As such, users first have to authenticate by way of the existing app to access VTAdmin-web. In this case, a trust boundary exists between the internet and the existing web app:



VTAdmin-api

The threat model of the VTAdmin-api has one trust boundary between the web ui and the VTAdmin-api. Once a request has been authenticated and authorized, it will not cross any further trust boundaries.



The requests made by VTAdmin-Web are unauthenticated and unauthorized until VTAdmin-api authenticates and authorizes them. In other words, the request becomes trusted after it passes VTAdmin-api.

Attack surface

A software attack surface refers to all possible entry points, vulnerabilities, and weak points within a software system that can be targeted or exploited by attackers to compromise its security. In this section we detail the attack surface of VTAdmin.

API endpoints

VTAdmin exposes a series of HTTP endpoints that handle a wide range of different operations, and they are plausible to a wide range of attacks. An attacker will need to be able to send requests to the VTAdmin-api server or have access to an authenticated VTAdmin-web client, but once they have obtained that, the attack complexity is simple; An attacker will launch an attack through requests to the server.

3rd-party dependencies

Security issues in VTAdmins 3rd-party dependencies can have a negative impact on VTAdmin. This can be achieved in several ways; For example, a threat actor could deliberately contribute vulnerable code that has a negative impact on VTAdmins users. VTAdmins dependencies are open source libraries most of which accept community contributions, and carefully placed vulnerabilities in some dependencies would make exploitation of VTAdmin users possible. Alternatively, VTAdmins dependencies could have vulnerabilities that a threat actor knows exist but does not place in the code. Threat actors can obtain information of vulnerabilities in public registries and assess whether projects use the vulnerable version. In either case, a threat actor can use a vulnerability in a 3rd-party dependency to escalate privileges and cause harm to VTAdmin users.

Local attacker

An attacker who has compromised the machine running VTAdmin may escalate privileges by listening on the network. For example, VTAdmin-api connects to Vtctld over GRPC. At this stage the request is already authenticated, and if an attacker can find a way to read traffic, they are potentially able to bypass authentication and assume the highest level of permissions that the RBAC can grant.
A local attacker with limited control over the file system can have a high impact, but the attack surface is small; VTAdmin does not rely heavily on the file system, and an attacker's options are therefore limited, however, an impactful vector could be controlling the `rbac.yaml` which could allow an attacker to assign permission to themselves, thus controlling the authentication at the highest possible level.

Fuzzing

As part of the audit, Ada Logics assessed Vitess's fuzz test suite with the purpose of improving it to cover critical parts of VTAdmin.

Vitess has done extensive fuzzing work; It carried out a fuzzing audit in 2020 which added coverage to complex text processing routines. Vitess is integrated into OSS-Fuzz which allows the fuzzers to run continuously and notify maintainers in case the fuzzers find bugs. The Vitess source code and the source code for the Vitess fuzzers are the two key software packages that OSS-Fuzz uses to fuzz Vitess.

The current OSS-Fuzz set up builds the fuzzers by cloning the upstream Vitess Github repository to get the latest Vitess source code and the CNCF-Fuzzing Github repository to get the latest set of fuzzers, and then builds the fuzzers against the cloned Vitess code. As such, the fuzzers are always run against the latest Vitess commit.

This build cycle happens daily and OSS-Fuzz will verify if any existing bugs have been fixed. If OSS-fuzz finds that any bugs have been fixed OSS-Fuzz marks the crashes as fixed in the Monorail bug tracker and notifies maintainers.

In each fuzzing iteration, OSS-Fuzz uses its corpus accumulated from previous fuzz runs. If OSS-Fuzz detects any crashes when running the fuzzers, OSS-Fuzz performs the following actions:

1. A detailed crash report is created.
2. An issue in the Monorail bug tracker is created.
3. An email is sent to maintainers with links to the report and relevant entry in the bug tracker.

OSS-Fuzz has a 90 day disclosure policy, meaning that a bug becomes public in the bug tracker if it has not been fixed. The detailed report is never made public. The Vitess maintainers will fix issues upstream, and OSS-Fuzz will pull the latest Vitess master branch the next time it performs a fuzz run and verify that a given issue has been fixed.

Vitess's fuzzers reside in CNCF's dedicated fuzzing repository, https://github.com/cncf/cncf-fuzzing, in which the community maintains them. In addition, community members also maintain the build, so that the fuzzers keep running, in case upstream code changes break the build.

During the audit, Ada Logics wrote 3 new fuzzers:

#	Name	URL	Running on OSS-Fuzz
1	FuzzKeyspaceCreation	https://github.com/cncf/cncf-fuzzing/blob/83bad32323d4a3515717c5f144faf38b2c7d20cb/projects/vitess/fuzz_keyspace_creation.go	Yes
2	FuzzShardCreation	https://github.com/cncf/cncf-fuzzing/blob/83bad32323d4a3515717c5f144faf38b2c7d20cb/projects/vitess/fuzz_shard_creation.go	Yes
3	FuzzTabletCreation	https://github.com/cncf/cncf-fuzzing/blob/bfec152c497f6d8e0786d2f89d99788b890e847f/projects/vitess/fuzz_tablet_test.go	Yes

The fuzzers target APIs at the topology server level responsible for creating keyspaces, shards and tablets and follow a similar pattern. Each fuzzer tests whether it can create a resource that will block subsequent operations against the given type. For example, the fuzzer for the shards will attempt to create a shard and afterwards test if operations - such as get-operations - against shards are rejected or fail. The fuzzers do not target the newly written VTAdmin code base, but they are still relevant for VTAdmin. In fact, during the auditing of the VTAdmin web interface, Ada Logics found two vulnerabilities with root cause at the topology level that were triggerable from VTAdmin. The two vulnerabilities allow users to create invalid keyspaces and shards that will block future operations against keyspaces and shards. An attacker could trigger these by creating the type with a well-crafted name. To test exhaustively for malicious names, Ada Logics wrote the three fuzzers. This proved fruitful instantly, as the shard fuzzer found more special cases in the shard name than were found during the manual auditing.

Ada Logics added the three fuzzers to Vitess's OSS-Fuzz integration, allowing them to run continuously and test for more special cases as well as code changes.

Issues found

Here we present the issues that we identified during the audit.

#	ID	Title	Severity	Fixed
1	ADA-VIT-SA23-1	Missing documentation on deploying VTAdmin-web securely	Moderate	Yes
2	ADA-VIT-SA23-2	Insecure cryptographic primitives	Informational	Yes
3	ADA-VIT-SA23-3	SQL injection in sqlutils	Informational	Yes
4	ADA-VIT-SA23-4	Path traversal in VtctldServers GetBackups method	Moderate	Yes
5	ADA-VIT-SA23-5	Users that can create keyspaces can deny access to already existing keyspaces	Moderate	Yes
6	ADA-VIT-SA23-6	VTAdmin-web ui is not authenticated by default	Moderate	No
7	ADA-VIT-SA23-7	Critical 3rd-party dependency is archived	Low	No
8	ADA-VIT-SA23-8	VTAdmin not protected by a rate limiter	Moderate	No
9	ADA-VIT-SA23-9	Profiling endpoints exposed by default	Moderate	Partially
10	ADA-VIT-SA23-10	Unsanitized parameters in html could lead to XSS	Low	Yes
11	ADA-VIT-SA23-11	Zip bomb in k8stopo	Low	No
12	ADA-VIT-SA23-12	VTAdmin users that can create shards can deny access to other functions	Moderate	Yes

ADA-VIT-SA23-1: Missing documentation on deploying VTAdmin-web securely

ID	ADA-VIT-SA23-1
Component	VTAdmin
Severity	Moderate
Fixed in: https://vitess.io/docs/17.0/reference/vtadmin/operators_guide/#best-practices	

We recommend adding a document on how to securely deploy and use VTAdmin. The purpose of this document is to provide a single source of actionable steps to use VTAdmin securely. The Vitess documentation currently contains limited information about the RBAC of Vitess, which is positive, however we consider the documentation incomplete. Lack of complete documentation on VTAdmins security could result in users unknowingly using VTAdmin in a way that is known to be insecure, and is either not documented, or the user will have to read the docs in full to find out that their deployment is insecure.

A security best practices document outlines the properties that Vitess considers insecure for users. For example, users wishing to write an authentication plugin would benefit from a general security best practices checklist. At the moment, Vitess does not offer guidelines on writing a secure plugin. Vitess provides an example - which is positive - however, the example demonstrates a minimum viable authenticator that has not been hardened for security; For example, the actor name is sent in plain text, and there is no minimum length required for the actor name.

ADA-VIT-SA23-2: Insecure cryptographic primitives

ID	ADA-VIT-SA23-2
Component	Multiple
Severity	Informational
Fixed	Yes

Vitess uses insecure hashing functions in a number of places across different packages. Usage of insecure hashing functions should be justified, and preferably in the code where they are used. Vitess worked on clarifying all usages and found that all uses of insecure hashing functions fall in one of two categories: they are either not cryptographic primitives or Vitess are bound to use a specific hashing algorithm to comply with MySQL's interface. This table illustrate how each case is categorized:

#	Component	Usage
1	MySQL Protocol	To implement MySQL handshake
2	Vindexes	Non-cryptographic hash
3	Evalengine	To support MySQL built-in functions
4	Tmutils	Non-cryptographic hash
5	S3 Backup Storage	Non-cryptographic hash part of the S3 API

As such, this issue did not require any code changes, and it has been kept here in the report as a reference for users that have internal policies that are sensitive to insecure hash functions. Vitess did remove the use of MD5 in Tmutils[1], but this was due to the code not being used rather than a security fix.

1: MySQL handshake

Vitess's mysql package implements a function that computes the hash of a mysql password using SHA1. SHA1 has been broken since 2004, deprecated by NIST since 2011, and security researchers have proven collisions in practice[2].

[1] https://github.com/vitessio/vitess/pull/12999
[2] https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

In the case of Vitess's mysql package, SHA1 is used to hash a password which we consider security sensitive data. We consider this a security issue.

We recommend using a secure hashing algorithm.

The issue exists in vitess/go/mysql/auth_server.go in `ScrambleMysqlNativePassword`:

https://github.com/vitessio/vitess/blob/58e2719069c35c2820e1bf33324f27c3fb5852f1/go/mysql/auth_server.go#L251

```go
func ScrambleMysqlNativePassword(salt, password []byte) []byte {
	if len(password) == 0 {
		return nil
	}

	// stage1Hash = SHA1(password)
	crypt := sha1.New()
	crypt.Write(password)
	stage1 := crypt.Sum(nil)

	// scrambleHash = SHA1(salt + SHA1(stage1Hash))
	// inner Hash
	crypt.Reset()
	crypt.Write(stage1)
	hash := crypt.Sum(nil)
	// outer Hash
	crypt.Reset()
	crypt.Write(salt)
	crypt.Write(hash)
	scramble := crypt.Sum(nil)

	// token = scrambleHash XOR stage1Hash
	for i := range scramble {
		scramble[i] ^= stage1[i]
	}
	return scramble
}
```

2: Vindexes

https://github.com/vitessio/vitess/blob/c43a162ea567f47a89b8d4a506d2995740737b79/go/vt/vtgate/vindexes/hash.go#L139

```go
var blockDES cipher.Block

func init() {
	var err error
	blockDES, err = des.NewCipher(make([]byte, 8))
	if err != nil {
		panic(err)
	}
	Register("hash", NewHash)
}
```

```
func vhash(shardKey uint64) []byte {
	var keybytes, hashed [8]byte
	binary.BigEndian.PutUint64(keybytes[:], shardKey)
	blockDES.Encrypt(hashed[:], keybytes[:])
	return hashed[:]
}

func vunhash(k []byte) (uint64, error) {
	if len(k) != 8 {
		return 0, fmt.Errorf("invalid keyspace id: %v", hex.EncodeToString(k))
	}
	var unhashed [8]byte
	blockDES.Decrypt(unhashed[:], k)
	return binary.BigEndian.Uint64(unhashed[:]), nil
}
```

3: Evalengine
https://github.com/vitessio/vitess/blob/a502fceda310886223342020136db5718ace34a5/go/vt/vtgate/evalengine/fn_crypto.go#L67

```
func (call *builtinSHA1) eval(env *ExpressionEnv) (eval, error) {
	arg, err := call.arg1(env)
	if err != nil {
		return nil, err
	}
	if arg == nil {
		return nil, nil
	}

	b := evalToBinary(arg)
	sum := sha1.Sum(b.bytes)
	buf := make([]byte, hex.EncodedLen(len(sum)))
	hex.Encode(buf, sum[:])
	return newEvalText(buf, defaultCoercionCollation(call.collate)), nil
}
```

https://github.com/vitessio/vitess/blob/a502fceda310886223342020136db5718ace34a5/go/vt/vtgate/evalengine/fn_crypto.go#L39

```
func (call *builtinMD5) eval(env *ExpressionEnv) (eval, error) {
	arg, err := call.arg1(env)
	if err != nil {
		return nil, err
	}
	if arg == nil {
		return nil, nil
	}

	b := evalToBinary(arg)
	sum := md5.Sum(b.bytes)
	buf := make([]byte, hex.EncodedLen(len(sum)))
	hex.Encode(buf, sum[:])
	return newEvalText(buf, defaultCoercionCollation(call.collate)), nil
}
```

https://github.com/vitessio/vitess/blob/641e5c6acc2345a4920d22745a7f9dbeb19e39c5/go/vt/vtgate/evalengine/compiler_asm.go#L3007

```
func (asm *assembler) Fn_SHA1(col collations.TypedCollation) {
	asm.emit(func(env *ExpressionEnv) int {
		arg := env.vm.stack[env.vm.sp-1].(*evalBytes)

		sum := sha1.Sum(arg.bytes)
		buf := make([]byte, hex.EncodedLen(len(sum)))
		hex.Encode(buf, sum[:])

		arg.tt = int16(sqltypes.VarChar)
		arg.bytes = buf
		arg.col = col
		return 1
	}, "FN SHA1 VARBINARY(SP-1)")
}
```

https://github.com/vitessio/vitess/blob/641e5c6acc2345a4920d22745a7f9dbeb19e39c5/go/vt/vtgate/evalengine/compiler_asm.go#L2992

```
func (asm *assembler) Fn_MD5(col collations.TypedCollation) {
	asm.emit(func(env *ExpressionEnv) int {
		arg := env.vm.stack[env.vm.sp-1].(*evalBytes)

		sum := md5.Sum(arg.bytes)
		buf := make([]byte, hex.EncodedLen(len(sum)))
		hex.Encode(buf, sum[:])

		arg.tt = int16(sqltypes.VarChar)
		arg.bytes = buf
		arg.col = col
		return 1
	}, "FN MD5 VARBINARY(SP-1)")
}
```

4: Tmutils

https://github.com/vitessio/vitess/blob/d1685d96bd7c2a57fc48a7e42ac38e4897741824/go/vt/mysqlctl/tmutils/schema.go#L206

```
func GenerateSchemaVersion(sd *tabletmanagerdatapb.SchemaDefinition) {
	hasher := md5.New()
	for _, td := range sd.TableDefinitions {
		if _, err := hasher.Write([]byte(td.Schema)); err != nil {
			panic(err) // extremely unlikely
		}
	}
	sd.Version = hex.EncodeToString(hasher.Sum(nil))
}
```

5: S3 Backup Storage
https://github.com/vitessio/vitess/blob/adb2535cf79926d2d9ecf9710a280d657103f74a/go/vt/mysqlctl/s3backupstorage/s3.go#L253

```
func (s3ServerSideEncryption *S3ServerSideEncryption) init() error {
	s3ServerSideEncryption.reset()

	if strings.HasPrefix(sse, sseCustomerPrefix) {
		sseCustomerKeyFile := strings.TrimPrefix(sse, sseCustomerPrefix)
		base64CodedKey, err := os.ReadFile(sseCustomerKeyFile)
		if err != nil {
			log.Errorf(err.Error())
			return err
		}

		decodedKey, err := base64.StdEncoding.DecodeString(string(base64CodedKey))
		if err != nil {
			decodedKey = base64CodedKey
		}

		md5Hash := md5.Sum(decodedKey)
		s3ServerSideEncryption.customerAlg = aws.String("AES256")
		s3ServerSideEncryption.customerKey = aws.String(string(decodedKey))
		s3ServerSideEncryption.customerMd5 =
aws.String(base64.StdEncoding.EncodeToString(md5Hash[:]))
	} else if sse != "" {
		s3ServerSideEncryption.awsAlg = &sse
	}
	return nil
}
```

ADA-VIT-SA23-3: SQL injection in sqlutils

ID	ADA-VIT-SA23-3
Component	sqlutils
Severity	Informational
Fixed in: https://github.com/vitessio/vitess/pull/12929	

The sqlutils package contains an SQL Injection vulnerability. The root cause of the vulnerability is that sqlutils will generate an sql query without sanitising the input thus essentially allowing the user to control the full query.

https://github.com/vitessio/vitess/blob/8263d6301ce1809891afb27c85294fcb3572395e/go/vt/external/golib/sqlutils/sqlutils.go#L423

```go
func WriteTable(db *sql.DB, tableName string, data NamedResultData) (err error) {
	if len(data.Data) == 0 {
		return nil
	}
	if len(data.Columns) == 0 {
		return nil
	}
	placeholders := make([]string, len(data.Columns))
	for i := range placeholders {
		placeholders[i] = "?"
	}
	query := fmt.Sprintf(
		`replace into %s (%s) values (%s)`,
		tableName,
		strings.Join(data.Columns, ","),
		strings.Join(placeholders, ","),
	)
	for _, rowData := range data.Data {
		if _, execErr := db.Exec(query, rowData.Args()...); execErr != nil {
			err = execErr
		}
	}
	return err
}
```

The vulnerable code was not used in any Vitess release and was removed from the project.

ADA-VIT-SA23-4: Path traversal in VtctldServers GetBackups method

ID	ADA-VIT-SA23-4
Component	vtctld server
Severity	Moderate
Fixed in: https://github.com/vitessio/website/pull/1471	

A path traversal vulnerability exists in the VtctldServers `GetBackups` method from a path being created from parameters of the incoming requests. This allows a request to pass a value that could traverse the storage.

https://github.com/vitessio/vitess/blob/aa87bc4e9f05e3de3955e9641799eca9114d83bb/go/vt/vtctl/grpcvtctldserver/server.go#L1161

```
func (s *VtctldServer) GetBackups(ctx context.Context, req
*vtctldatapb.GetBackupsRequest) (resp *vtctldatapb.GetBackupsResponse, err error) {
	span, ctx := trace.NewSpan(ctx, "VtctldServer.GetBackups")
	defer span.Finish()

	defer panicHandler(&err)

	span.Annotate("keyspace", req.Keyspace)
	span.Annotate("shard", req.Shard)
	span.Annotate("limit", req.Limit)
	span.Annotate("detailed", req.Detailed)
	span.Annotate("detailed_limit", req.DetailedLimit)

	bs, err := backupstorage.GetBackupStorage()
	if err != nil {
		return nil, err
	}
	defer bs.Close()

	bucket := filepath.Join(req.Keyspace, req.Shard)
	span.Annotate("backup_path", bucket)

	bhs, err := bs.ListBackups(ctx, bucket)
	if err != nil {
		return nil, err
	}

	totalBackups := len(bhs)
	if req.Limit > 0 {
		totalBackups = int(req.Limit)
	}
```

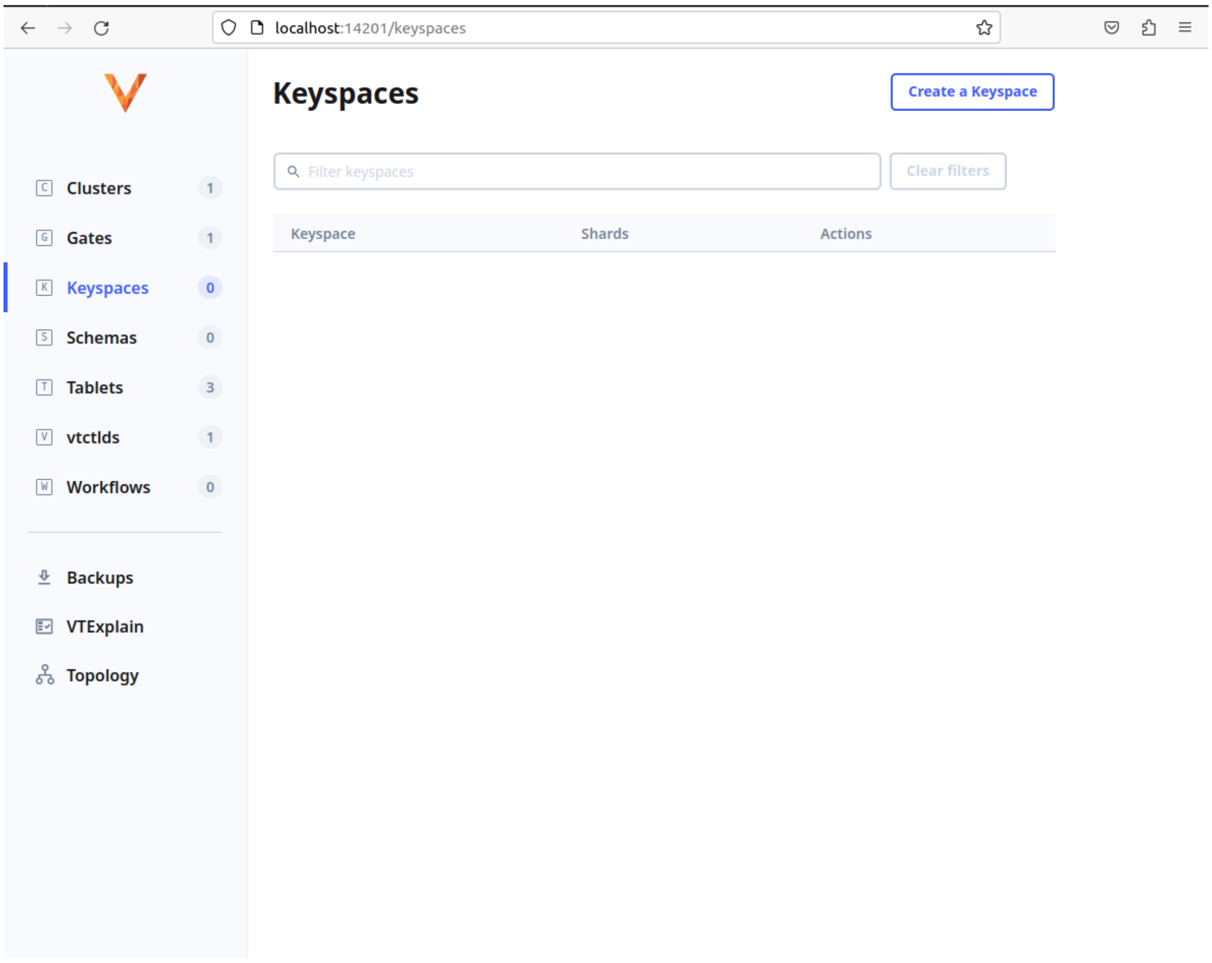
ADA-VIT-SA23-5: Users that can create keyspaces can deny access to already existing keyspaces

ID	ADA-VIT-SA23-5
Component	VTAdmin
Severity	Moderate
Fixed in: https://github.com/vitessio/vitess/pull/12843	

Users that can create keyspaces via the VTAdmin-web UI can specify a name that prevents the endpoint at `/keyspaces` from displaying any keyspaces.

If a user creates a keyspace from `/keyspaces/create` in VTAdmin-web and the name contains the character "/", then no keyspaces will be displayed at `/keyspaces`.

In this screenshot, Ada Logics have created a keyspace named `a/a`, and 6 keyspaces exist:



If we check the developer tools, we see we get an error:

```
HttpResponseNotOkError: [status 500] /api/keyspaces: unknown rpc error:
code = Unknown desc = node doesn't exist:
/vitess/global/keyspaces/KEYSPACE_NAME/Keyspace
```

If a single keyspace returns an error, VTAdmin-api does not return any keyspaces. The below code is responsible for fetching the existing keyspaces:
https://github.com/vitessio/vitess/blob/da1906d54eaca4447e039d90b96fb07251ae852c/go/vt/vtadmin/cluster/cluster.go#L1141

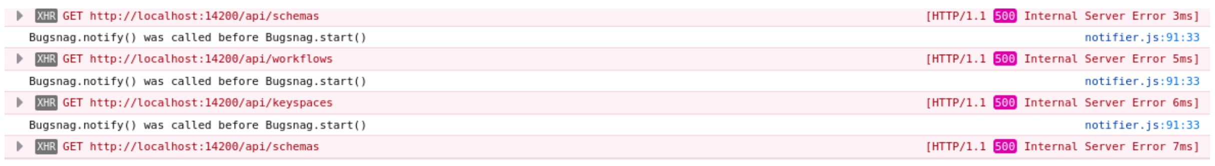
```
func (c *Cluster) GetKeyspaces(ctx context.Context) ([]*vtadminpb.Keyspace, error) {
        span, ctx := trace.NewSpan(ctx, "Cluster.GetKeyspaces")
        defer span.Finish()

        AnnotateSpan(c, span)

        if err := c.topoReadPool.Acquire(ctx); err != nil {
                return nil, fmt.Errorf("GetKeyspaces() failed to acquire topoReadPool:
%w", err)
        }

        resp, err := c.Vtctld.GetKeyspaces(ctx, &vtctldatapb.GetKeyspacesRequest{})
        c.topoReadPool.Release()

        if err != nil {
                return nil, err
        }

        var (
                m         sync.Mutex
                wg        sync.WaitGroup
                rec       concurrency.AllErrorRecorder
                keyspaces = make([]*vtadminpb.Keyspace, len(resp.Keyspaces))
        )

        for i, ks := range resp.Keyspaces {
                wg.Add(1)
                go func(i int, ks *vtctldatapb.Keyspace) {
                        defer wg.Done()

                        shards, err := c.FindAllShardsInKeyspace(ctx, ks.Name,
FindAllShardsInKeyspaceOptions{})
                        if err != nil {
                                rec.RecordError(err)
                                return
                        }

                        keyspace := &vtadminpb.Keyspace{
                                Cluster:  c.ToProto(),
                                Keyspace: ks,
                                Shards:   shards,
                        }

                        m.Lock()
                        defer m.Unlock()
                        keyspaces[i] = keyspace
                }(i, ks)
        }

        wg.Wait()
        if rec.HasErrors() {
                return nil, rec.Error()
        }

        return keyspaces, nil
}
```

In the first chunk of highlighted code, `rec` records an error of a single keyspace. In the second chunk of highlighted code, `GetKeySpaces` return `nil` and the error of the keyspace that had the error.

This is a security issue because a user can control whether VTAdmin-api returns the existing keyspaces thus enabling a denial-of-service attack vector. From the point of view of the threat model, this is a breach of security, because the user that creates the faulty keyspace has been granted permission to create keyspaces - not to prevent other users from viewing the existing keyspaces.

The `/topology` route shows the existing keyspaces correctly.

The `/workflows` and `/schemas` apis get denied too, when a user creates a keyspace containing the "/" char:



```
▶ XHR GET http://localhost:14200/api/schemas          [HTTP/1.1 500 Internal Server Error 3ms]
  Bugsnag.notify() was called before Bugsnag.start()                        notifier.js:91:33
▶ XHR GET http://localhost:14200/api/workflows        [HTTP/1.1 500 Internal Server Error 5ms]
  Bugsnag.notify() was called before Bugsnag.start()                        notifier.js:91:33
▶ XHR GET http://localhost:14200/api/keyspaces        [HTTP/1.1 500 Internal Server Error 6ms]
  Bugsnag.notify() was called before Bugsnag.start()                        notifier.js:91:33
▶ XHR GET http://localhost:14200/api/schemas          [HTTP/1.1 500 Internal Server Error 7ms]
```

This issue was assigned CVE-2023-29194.

ADA-VIT-SA23-6: VTAdmin-web ui is not authenticated by default

ID	ADA-VIT-SA23-6
Component	VTAdmin
Severity	Moderate
Fixed	No

VTAdmins web ui is not authenticated by default. As such, any default installation is insecure by default. The immediate impact is that the web ui is fully exposed to any user that has access to the domain and port hosting the Web ui. This can allow a threat actor to achieve elevated privileges by getting access to a system running VTAdmin-web. For example, if the deployment is exposed to the internet, any untrusted user could achieve the level of privileges of the web ui that they can locate. It is likely that VTAdmin-web ui is exposed on the internet given that the web ui and VTAdmin-api are designed to be deployed on different domains.

This is not a code vulnerability with high severity but a security issue related to VTAdmins default settings. Missing authentication is a security issue related to VTAdmins design.

We understand from internal discussions during the audit that Vitess wishes to allow as flexible a usage of the web ui as possible which authentication-by-default is counter-productive to. However, from the perspective of security, we recommend an authentication-by-default design that can be removed from deployments.

ADA-VIT-SA23-7: Critical 3rd-party dependency is archived

ID	ADA-VIT-SA23-7
Component	VTAdmin
Severity	Low
Fixed	No

VTAdmin uses the `github.com/gorilla/mux` library for routing incoming requests to VTAdmin-api. As of 9th December 2022, the gorilla/mux library has been archived and is now unmaintained. This does not mean that the library is insecure to use, but it does have implications for its security.
One implication is that gorilla/mux is unlikely to fix issues - both reliability issues and security vulnerabilities. Furthermore, the project is unlikely to even accept and triage security disclosures.
Another implication is that the project is unlikely to do its own ongoing security work. For example, Ada Logics attempted to involve the project in integrating continuous fuzzing by way of OSS-Fuzz in 2020: https://github.com/gorilla/mux/pull/575 via a pull request that has still not been merged.

As such, gorilla/mux has a low security posture that can affect VTAdmin. Since the library is designed to be exposed to untrusted input, security vulnerabilities could have a critical impact on VTAdmin.

ADA-VIT-SA23-8: VTAdmin not protected by a rate limiter

ID	ADA-VIT-SA23-8
Component	VTAdmin-api
Severity	Moderate
Fixed	No

Description
VTAdmin is not protected by a rate limiter which makes it susceptible to multiple attack vectors.

The underlying Vitess backend is guarded by a rate limiter, and the impact of this attack would be limited to stealing RBAC credentials or launching a DDoS attack.

PoC
We demonstrate the issue with the following PoC. The idea is that we should be able to execute all 100,000 requests without being blocked - which demonstrates lack of a rate limiter. The PoC checks the return value of the http response. The assumption is that if a rate limiter would prevent an attacker from sending 100,000 requests, VTAdmin would return an error or an empty response. The PoC therefore checks whether VTAdmin returns a valid hostname, and if not, then it breaks the loop and checks if it sent 100,000 requests.

```
import requests
import json

url = 'http://localhost:14200/api/vtctlds'

j = 0
for i in range(100000):
    x = requests.get(url)
    resp = json.loads(x.text)
    if resp["result"]["vtctlds"][0]["hostname"] != "localhost:15999":
        break
    j+=1

if j != 10000:
    print("We hit a limit")
else:
    print("We sent all 100000 requests")
```

This script sends all 100,000 requests to the server successfully demonstrating how easy it is to exploit the lack of rate limiting.

ADA-VIT-SA23-9: Profiling endpoints exposed by default

ID	ADA-VIT-SA23-9
Component	Servenv
Severity	Low
Fixed	Partially

Vitess's servenv package exposes the HTTP handlers for profiling by default. We recommend exposing these handlers only if users choose to expose them, to prevent accidentally revealing sensitive information in a production deployment.

https://github.com/vitessio/vitess/blob/137cf9daf41112a553f617c66a56fd8b06fad20b/go/vt/servenv/servenv.go#L33

```
package servenv

import (
	// register the HTTP handlers for profiling
	_ "net/http/pprof"
	"net/url"
	"os"
	"os/signal"
	"runtime/debug"
	"strings"
	"sync"
	"syscall"
	"time"
```

Vitess is working on fixing this. It has been partially fixed in https://github.com/vitessio/vitess/pull/12987.

ADA-VIT-SA23-10: Unsanitized parameters in html could lead to XSS

ID	ADA-VIT-SA23-10
Component	Multiple
Severity	Low
Fixed in: • https://github.com/vitessio/vitess/pull/12939 • https://github.com/vitessio/vitess/pull/12940	

Vitess uses Go templating a number of places to generate HTML but does not escape the parameters to the template. Vitess could be exposed to front-end attacks such as cross-site scripting, if an attacker manages to pass valid javascript into the templates.

Ada Logics found the following parts of Vitess to be impacted:

https://github.com/vitessio/vitess/blob/867043971bd0aa969fe7e34ae8564330972e4d89/go/vt/topo/topoproto/shard.go#L54

```
func SourceShardAsHTML(source *topodatapb.Shard_SourceShard) template.HTML {
	result := fmt.Sprintf("<b>Uid</b>: %v</br>\n<b>Source</b>: %v/%v</br>\n",
source.Uid, source.Keyspace, source.Shard)
	if key.KeyRangeIsPartial(source.KeyRange) {
		result += fmt.Sprintf("<b>KeyRange</b>: %v-%v</br>\n",
			hex.EncodeToString(source.KeyRange.Start),
			hex.EncodeToString(source.KeyRange.End))
	}
	if len(source.Tables) > 0 {
		result += fmt.Sprintf("<b>Tables</b>: %v</br>\n",
			strings.Join(source.Tables, " "))
	}
	return template.HTML(result)
}
```

https://github.com/vitessio/vitess/blob/bd78c08ced8f6a3e55279d308a5a8402fd6780bc/go/vt/srvtopo/status.go#L126

```
func (st *SrvKeyspaceCacheStatus) StatusAsHTML() template.HTML {
	if st.Value == nil {
		return template.HTML("No Data")
	}

	result := "<b>Partitions:</b><br>"
	for _, keyspacePartition := range st.Value.Partitions {
```

```
                result += "&nbsp;<b>" + keyspacePartition.ServedType.String() + ":</b>"
                for _, shard := range keyspacePartition.ShardReferences {
                        result += "&nbsp;" + shard.Name
                }
                result += "<br>"
        }

        if len(st.Value.ServedFrom) > 0 {
                result += "<b>ServedFrom:</b><br>"
                for _, sf := range st.Value.ServedFrom {
                        result += "&nbsp;<b>" + sf.TabletType.String() + ":</b>&nbsp;" +
sf.Keyspace + "<br>"
                }
        }

        return template.HTML(result)
}
```

https://github.com/vitessio/vitess/blob/a49702d9f9782c14d96030c8d2771c8decb39948/go/vt/discovery/tablets_cache_status.go#L57

```
func (tcs *TabletsCacheStatus) StatusAsHTML() template.HTML {
        tLinks := make([]string, 0, 1)
        if tcs.TabletsStats != nil {
                sort.Sort(tcs.TabletsStats)
        }
        for _, ts := range tcs.TabletsStats {
                color := "green"
                extra := ""
                if ts.LastError != nil {
                        color = "red"
                        extra = fmt.Sprintf(" (%v)", ts.LastError)
                } else if !ts.Serving {
                        color = "red"
                        extra = " (Not Serving)"
                } else if ts.Target.TabletType == topodatapb.TabletType_PRIMARY {
                        extra = fmt.Sprintf(" (PrimaryTermStartTime: %v)",
ts.PrimaryTermStartTime)
                } else {
                        extra = fmt.Sprintf(" (RepLag: %v)",
ts.Stats.ReplicationLagSeconds)
                }
                name := topoproto.TabletAliasString(ts.Tablet.Alias)
                tLinks = append(tLinks, fmt.Sprintf(`<a href="%s"
style="color:%v">%v</a>%v`, ts.getTabletDebugURL(), color, name, extra))
        }
        return template.HTML(strings.Join(tLinks, "<br>"))
}
```

https://github.com/vitessio/vitess/blob/47611bca3951ecdf442dda5c8fc12f4eb9cff29c/go/vt/callinfo/plugin_mysql.go#L56

```
func (mci *mysqlCallInfoImpl) HTML() template.HTML {
        return template.HTML("<b>MySQL User:</b> " + mci.user + " <b>Remote Addr:<b> " +
mci.remoteAddr)
}
```

https://github.com/vitessio/vitess/blob/47611bca3951ecdf442dda5c8fc12f4eb9cff29c/go/vt/callinfo/plugin_grpc.go#L67

```
func (gci *gRPCCallInfoImpl) HTML() template.HTML {
	return template.HTML("<b>Method:</b> " + gci.method + " <b>Remote Addr:</b> " +
gci.remoteAddr)
}
```

The Vitess maintainers triaged these cases extensively to assess whether user-controlled data could be passed to any of the templates to launch an XSS attack. Such an attack can be highly critical, since some of the templates are meant to be viewed by a Vitess admin. At the time of the audit, the Vitess maintainers found that the parameters passed to the templates were only user-controlled in one of the cases. This case was triaged heavily and the Vitess team found that an attack vector was not possible.

To guard against future issues with templating, Vitess now uses the https://github.com/google/safehtml library for html templating. In addition, Vitess now uses templates instead of raw string concatenation for non-user controlled input.

ADA-VIT-SA23-11: Zip bomb in k8stopo

ID	ADA-VIT-SA23-11
Component	k8stopo
Severity	Low
Fixed	No

Description
K8stopo may be susceptible to a zip bomb attack from lack of size checking when extracting a zip file. k8stopo reads the contents of an extracted zip archive entirely into memory on the highlighted line below. If the archive is accidentally or intentionally crafted in such a way that it is larger than the available memory, the zip archive could cause k8stopo to exhaust memory thereby resulting in denial of service.

https://github.com/vitessio/vitess/blob/395840969d183dbdb080eabf95b0bcd2ddefb885/go/vt/topo/k8stopo/file.go#L70

```go
func unpackValue(value []byte) ([]byte, error) {
	decoder := base64.NewDecoder(base64.StdEncoding, bytes.NewBuffer(value))

	zr, err := gzip.NewReader(decoder)
	if err != nil {
		return []byte{}, fmt.Errorf("unable to create new gzip reader: %s", err)
	}

	decoded := &bytes.Buffer{}
	if _, err := io.Copy(decoded, zr); err != nil {
		return []byte{}, fmt.Errorf("error coppying uncompressed data: %s", err)
	}

	if err := zr.Close(); err != nil {
		return []byte{}, fmt.Errorf("unable to close gzip reader: %s", err)
	}

	return decoded.Bytes(), nil
}
```

ADA-VIT-SA23-12: VTAdmin users that can create shards can deny access to other functions

ID	ADA-VIT-SA23-12
Component	VTAdmin
Severity	Moderate
Fixed in: https://github.com/vitessio/vitess/pull/12917	

Description
A user that can create shards in Vitess can also deny access to shards by creating a shard with a well-crafted name. This will cause Vitess to create the shard, and any access to shards will subsequently be denied from.

This is especially impactful for VTAdmin which has a granular permission control. As such, a user with low privileges - for example only create-privileges against shards - can deny all other users from fetching created shards.

The issue has been assigned CVE-2023-29195.

SLSA review

In this section we present our findings from our SLSA compliance review of Vitess.

SLSA is a framework for assessing artifact integrity and ensure a secure supply chain for downstream users.

In this part of the audit, we assessed Vitess's SLSA compliance by following SLSA's v0.1 requirements[3]. This version of the SLSA standard is currently in alpha and is likely to change.

Our assessment shows Vitess's current level of compliance.

Vitess manages its source code on Github which makes it version controlled and possible to verify the commit history. The source code is retained indefinitely and all commits are verified by two different maintainers.
The build is fully scripted and is invoked via Vitess's Makefile. The build runs in Github Actions which provisions the build environment for building Vitess and does not reuse it for other purposes. Github actions are not fully isolated, in that the build can access env var mounted secrets. The build is also not fully hermetic, since it runs with network access, which Vitess needs to pull in dependencies at build time.
Vitess lacks the provenance statement, and this is the area where Vitess can improve the most. Vitess can achieve level 1 SLSA compliance by:

- Making the provenance statement available with releases.
- Including the builder, artifacts and build instructions in the provenance.

The build instructions are the highest level of entry, which in Vitess' case is the command that invokes the Makefile.

Overview

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source - Version controlled		✓	✓	✓
Source - Verified history			✓	✓
Source - Retained indefinitely				✓
Source - Two-person reviewed				✓

[3] https://slsa.dev/spec/v0.1/requirements

Build - Scripted build	✓	✓	✓	✓
Build - Build service		✓	✓	✓
Build - Build as code			✓	✓
Build - Ephemeral environment			✓	✓
Build - Isolated			⛔	⛔
Build - Parameterless				✓
Build - Hermetic				⛔
Build - Reproducible				✓
Provenance - Available	⛔	⛔	⛔	⛔
Provenance - Authenticated		⛔	⛔	⛔
Provenance - Service generated		⛔	⛔	⛔
Provenance - Non-falsifiable			⛔	⛔
Provenance - Dependencies complete				⛔
Provenance - Identifies artifact	⛔	⛔	⛔	⛔
Provenance - Identifies builder	⛔	⛔	⛔	⛔
Provenance - Identifies build instructions	⛔	⛔	⛔	⛔
Provenance - Identifies source code		⛔	⛔	⛔
Provenance - Identifies entry point			⛔	⛔
Provenance - Includes all build parameters			⛔	⛔
Provenance - Includes all transitive dependencies				⛔
Provenance - Includes reproducible info				⛔
Provenance - Includes metadata	⛔	⛔	⛔	⛔
Common - Security	Not defined by SLSA requirements			
Common - Access				✓
Common - Superusers				✓

Conclusions

In this engagement, Ada Logics completed a security of Vitess's VTAdmin component. The scope was well-defined and set to be a 5-week engagement. The goals were to formalize a threat model of VTAdmin, conduct a manual code review of VTAdmin and the remaining Vitess codebase, assess and improve Vitess's fuzzing suite and finally carry out a SLSA compliance review.

Our overall assessment of VTAdmin is highly positive. VTAdmin follows secure design and code practices, and VTAdmin-web is written with React which is hardened to defend against many cases of Cross-Site Scripting. The backend, VTAdmin-api, is written in Go which is a memory-safe language.

The VTAdmin code is clean and well-structured, making it easy to understand and audit. This is important for both external auditors such as Ada Logics as well as for the Vitess team when triaging bug reports.

The auditing team found two vulnerabilities during the audit, and the Vitess team were fast to respond to these. The Vitess team also extensively triaged another issue reported by Ada Logics to determine its severity. This professional response to security disclosures is an important element of well-maintained security policy. The highest severity of any issue found was Moderate which is a testament to the security practices that Vitess follows with VTAdmin as well as the remaining code base.

Vitess's fuzzing suite is extensive, targets complex parts of the code base and runs continuously on OSS-Fuzz which are important elements of a solid fuzzing suite. Ada Logics added two fuzzers that test the root cause for the two CVEs. The fuzzers found edge cases that could trigger both vulnerabilities but had not been found and fixed initially, and the Vitess team subsequently fixed these.

Vitess showed great initiative with their SLSA compliance, having started work on generating the provenance attestation before the audit commenced.

Ada Logics would like to thank the Vitess team for a productive security audit with fruitful collaboration on the found issues. We would also like to thank OSTIF for facilitating the audit and the CNCF for funding the audit.