Zoo, KittyCAD

Security Assessment

June 4, 2024

Prepared for:
Jessie Frazelle
Zoo

Prepared by: **Spencer Michaels and Max Ammann**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineering director was associated with this project:

David Pokora, Engineering Director, Application Security
david.pokora@trailofbits.com

The following consultants were associated with this project:

Spencer Michaels, Consultant
spencer.michaels@trailofbits.com

Max Ammann, Consultant
max.ammann@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
April 11, 2024	Pre-project kickoff call
April 16, 2024	Threat model discovery call #1
April 18, 2024	Threat model discovery call #2
April 22, 2024	Status call #1
April 26, 2024	Delivery of report draft
April 29, 2024	Report readout meeting
June 4, 2024	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Zoo engaged Trail of Bits to review the security of its CAD modeling platform and its associated infrastructure. Zoo provides a cloud-based CAD modeling app, streamed to user devices from remote engine instances, as well as an LLM interface that can generate models based on textual descriptions.

A team of two consultants conducted a lightweight threat modeling exercise from April 15 to April 19, 2024, followed by a code review from April 22 to April 26, 2024, for a total of four engineer-weeks of effort. This report covers the code review portion of the engagement. Our testing efforts focused on authentication and authorization, session handling, and user input handling in the rendering engine. With full access to source code, documentation, and a test instance, we performed static and dynamic testing of the Zoo system, using automated and manual processes.

Observations and Impact

Although we noted a number of findings, most are of low and informational severity, and less than a third are of high or medium severity (i.e., are readily exploitable and/or have substantial impact); the issues involve a wide spread of security control categories. On the whole, this points to a better-than-average security posture without major systematic issues, but also to a lack of defense-in-depth measures, especially in terms of security-aware development and testing practices that would prevent occasional mistakes from cropping up in various areas of concern throughout the codebase. In at least one case, a chain of individually small issues could be combined to execute an account takeover. We also identified several inputs that could crash the engine (TOB-ZOO-2, TOB-ZOO-3, TOB-ZOO-4). Such issues are difficult to avoid given a sufficiently large amount of native code but can be detected via targeted fuzzing.

In addition, we discovered that a series of relatively simple mistakes in Zoo's configuration of rate limiting, CORS checks, and cookie settings, combined with a lack of domain validation for authentication redirects, could be chained together to perform an account takeover (TOB-ZOO-8, TOB-ZOO-9, TOB-ZOO-10, TOB-ZOO-11). Many of these issues could have been prevented by better-established standards for secure development (e.g., security-first default cookie options, a high level of specificity about whether domains should be checked by suffix or exact string matching, and so on).

Finally, we found Zoo's documentation to be relatively sparse, aside from automatically generated API documentation. Given the company's small size and rapid pace of development, this is not unusual; however, care should be taken to flesh out the documentation as the codebase and infrastructure design solidifies.

Note that our review of the Zoo codebase was limited to only two engineer-weeks of effort, narrowly focusing on code paths and potential issues related to authentication, authorization, session management, and input handling. Therefore, we are limited in our ability to judge the security of the codebase as a whole; these observations are based on the limited scope within which we conducted our audit, which may or may not be representative of the codebase as a whole.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Zoo take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Consider adding more fuzz tests.** We identified the following fuzz targets:
    - Modeling commands: The engine parses JSON input that changes the rendered 3D landscape (e.g., imports a 3D model, changes the background color). A structured fuzzing approach using a tool such as the `arbitrary` crate might uncover bugs reachable by users.
    - Format library: The engine can import and export models and convert files between 3D models. File formats are often difficult to implement, as input is fully untrusted. The commercial HOOPS library might also be a good target for fuzzing.
- **Enable correctness and pedantic Rust lints.** We recommend enabling the following lints:
    - `panicking_unwrap`
    - `arithmetic_side_effects`
    - `string_slice`
- **Consider conducting an additional manual review of the format library.** The format library calls into the HOOPS library and calls a significant amount of unsafe Rust code. Further review might uncover memory safety issues.
- **Add code documentation.** Start with higher-level documentation (Rust modules) and continue with struct and function documentation.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	3
Low	6
Informational	4
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Authentication	2
Configuration	1
Data Exposure	2
Data Validation	3
Denial of Service	5

Project Goals

The engagement was scoped to provide a security assessment of the Zoo CAD modeling platform. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are Zoo's authentication and authorization methods, both native and externally supported (i.e., OAuth and SAML), correctly and securely implemented?
- Does Zoo's session handling mechanism appropriately associate sessions with users, time out unused sessions, prevent cross-session data leakage, and so on?
- Can model data be leaked between engine instances?
- Could an unauthenticated attacker compromise Zoo's WebRTC handler?
- Is Zoo's model import functionality appropriately protected against the possibility of malicious models being uploaded?
- Are modeling commands handled correctly?
- Is the WebRPC API used correctly?
- Is WebRTC used correctly?
- Is unsafe Rust used correctly?

Project Targets

The engagement involved a review and testing of the targets listed below. Additional repositories within the KittyCAD GitHub organization were consulted as needed during the audit but were not the primary focus.

api-deux

Repository	https://github.com/KittyCAD/api-deux
Version	`61f06dd724e32c8cd4b2bc61b5bf316f10a5099c`
Type	Rust
Platform	Dropshot

common

Repository	https://github.com/KittyCAD/common
Version	`e3dfab72f22e951f193695bda67aac46ea503e69`
Type	Rust
Platform	-

engine

Repository	https://github.com/KittyCAD/engine
Version	`a4ee620f3b3aded1c58d77d822aece913cd172c5`
Types	C++/Rust
Platform	Vulkan

infra

Repository	https://github.com/KittyCAD/infra
Version	`d965b6cc538afb4f64693cd7392c2282011183a2`
Types	C++/Rust
Platform	Vulkan

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manual code review and static analysis of critical code paths pertaining to the following:
    - Authentication and authorization, including native authentication, SAML, and OAuth
    - Session management
    - User input related to modeling commands in the engine
- Dynamic testing of critical endpoints in the Zoo API
- General correctness of the use of the WebRTC library

Coverage Limitations

- Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. Given the short time frame of this code review, our audit focused on code paths related to the highest-priority items noted in our threat model of the system, as noted in the coverage list above. Other security concerns (e.g., compromising the model parsing engine) could not be covered due to time constraints.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We found that the engine could crash due to a division by zero (TOB-ZOO-1). This does not appear to indicate a systemic issue, as apart from that, no flaws with arithmetic calculations were discovered.	**Satisfactory**
Auditing	Logging and auditing measures were not the primary focus of this review. However, the code paths that were audited, as a representative sample, feature sufficient logging and error reporting measures.	**Satisfactory**
Authentication / Access Controls	In addition to its native authentication, Zoo supports external authentication providers via SAML and OAuth. Both native and provider-specific authentication methods are straightforwardly and consistently implemented, and we noted no security issues related to credential validation or access controls. However, we noticed a chain of bugs that can lead to a very effective phishing attack that could allow for a complete account takeover (TOB-ZOO-8, TOB-ZOO-9, TOB-ZOO-10, TOB-ZOO-11).	**Moderate**
Complexity Management	The Zoo codebase is well organized, with discrete functionality logically isolated into separate files and directories. Individual functions are relatively short and easy to follow.	**Satisfactory**
Configuration	Due to time constraints, the configuration of external components was not considered during the code review. However, we noticed that missing rate limiting for the email authentication has an impact on the security of the system (TOB-ZOO-8).	**Not Considered**

Cryptography and Key Management	The authentication-related components that were audited use industry-standard cryptographic mechanisms with no apparent misconfigurations. Due to time constraints, cryptography-related code that may exist in other parts of the system was not inspected.	**Satisfactory**
Data Handling	The use of Rust and a clean coding style avoid data-handling bugs on the API surface. However, we discovered a flaw in the verification of URL hosts that could enable cross-origin attacks (TOB-ZOO-10, TOB-ZOO-11). We found several inputs that can crash the engine (TOB-ZOO-2, TOB-ZOO-3, TOB-ZOO-4).	**Moderate**
Documentation	Zoo features basic high-level architecture documentation. We noticed a lack of documentation relating to code, modules, and functions.	**Moderate**
Maintenance	Due to time constraints, update mechanisms were not considered during the code review.	**Not Considered**
Memory Safety and Error Handling	While some of our testing touched on attacks that could cause memory corruption, it was not the primary focus of our review. We recommend implementing fuzz tests in order to comprehensively evaluate native code components' susceptibility to malicious input. We found an instance of incorrect use of unsafe Rust code, which does not pose a risk right now (TOB-ZOO-14).	**Further Investigation Required**

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	Risk of Rust panic due to division by zero	Denial of Service	**Informational**
2	Assertion on a positive FoV setting causes the engine to abort on zero- or negative-value settings	Denial of Service	**Low**
3	Engine aborts when rendering zero-area videos	Denial of Service	**Low**
4	Rendering videos with a low height and width divisible by 4 causes segmentation fault	Denial of Service	**Low**
5	UDP firewall can be circumvented through TURN WebRTC server	Access Controls	**Low**
6	GitHub App webhook panics if non-ASCII characters are sent in headers	Denial of Service	**Informational**
7	Infinite WebSocket authentication retry attempts	Authentication	**Low**
8	Missing rate limiting for authentication email endpoint	Configuration	**Low**
9	Insufficient validation for callback URLs in authentication email endpoint	Data Validation	**Medium**
10	CORS configuration allows malicious cross-origin requests	Data Validation	**Medium**
11	Cookie allows authentication on malicious origins	Authentication	**High**

12	Plaintext secrets in the infrastructure repository	Data Exposure	**Medium**
13	Personal email in source code	Data Exposure	**Informational**
14	Unsafe code wrapped in safe API	Data Validation	**Informational**

Detailed Findings

1. Risk of Rust panic due to division by zero

Severity: **Informational**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ZOO-1
Target: `engine/src/server/endpoints/modeling/engine.rs`	

Description

The worker thread responsible for generating frames and streaming them out through WebRTC will panic if a client requests a WebSocket with an FPS setting of `0`.

Using the websocat tool, it is possible to initiate a WebRTC session to an engine instance.

```
export URL="ws://localhost:8080/ws/modeling/commands"
websocat "$URL?video_res_width=512&video_res_height=512&fps=0"
```

Figure 1.1: A request that causes a panic

The panic is caused due to a division by zero in the following code.

```
/// Consume the engine, returning an iterable of Frames. Dropping
/// the consumer will drop the engine.
pub async fn into_iter(self) -> Result<Receiver<(Vec<u8>, Duration, SystemTime)>> {
        ...
        let duration = Duration::from_nanos(1_000_000_000 / fps as u64);
        let (tx, rx) = channel(1);

        tokio::task::Builder::new()
            .name("Engine::into_iter")
            .spawn(async move {
                ...
            })?;
        Ok(rx)
    }
}
```

Figure 1.2: The code that panics if FPS is 0 (engine/src/server/endpoints/modeling/engine.rs#130–192)

After the panic, the WebSocket connection stays open, but the engine reports errors for successive modeling commands.

```
2024-04-24T13:28:44.952404Z  WARN engine_api::server::endpoints::modeling: Engine
command failed CppErrors(CppErrors([CppError { code: InternalEngine, msg_internal:
"exception No existing render thread during call to RenderThread::current()!",
msg_external: "internal error" }]))
  at src/server/endpoints/modeling.rs:1092
```

Figure 1.3: The error message after a crash of the worker thread

Recommendations

Short term, modify the above-noted `into_iter()` function to gracefully return an error if FPS is `0`.

Long term, enable the Clippy lint `arithmetic_side_effects`.

2. Assertion on a positive FoV setting causes the engine to abort on zero- or negative-value settings

Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ZOO-2
Target: `engine/cpp/engine/scene/camera.cpp`	

Description

If a user requests a negative field of view (FoV), the engine crashes due to an assertion in the glm library. Sending the following modeling command to the engine causes an abort.

```
{
  "type": "modeling_cmd_req",
  "cmd_id": "49d96493-dda6-4116-adae-032dbceb3279",
  "cmd": {
    "type": "default_camera_perspective_settings",
    "center": {
      "x": 0,
      "y": 0,
      "z": 0
    },
    "up": {
      "x": 0,
      "y": 0,
      "z": 1
    },
    "vantage": {
      "x": -299,
      "y": -1103,
      "z": 182
    },
    "fov_y": -52.5,
    "z_near": 0.1,
    "z_far": 1000,
    "sequence": 783
  }
}
```

Figure 2.1: The modeling command that changes the perspective with a negative FoV

The assertion is reached through the `perspectiveFov` function.

```
template<typename T>
GLM_FUNC_QUALIFIER mat<4, 4, T, defaultp> perspectiveFovRH_ZO(T fov, T width, T
height, T zNear, T zFar)
```

```
{
        assert(width > static_cast<T>(0));
        assert(height > static_cast<T>(0));
        assert(fov > static_cast<T>(0));

        ...
}
```

Figure 2.2: The function that asserts that the FoV is positive (glm/glm/ext/matrix_clip_space.inl#351–369)

The above function is called from the code related to the camera perspective.

```
void Camera::setPerspective(float fovy, float zNear, float zFar)
{
    CHECK_RENDER_THREAD();
    projectionMatrix = glm::perspectiveFov(glm::radians(fovy), (float)screenWidth,
(float)screenHeight, zNear, zFar);
    fovY = fovy;
    nearClip = desiredNearClip = zNear;
    farClip = desiredFarClip = zFar;
    orthoMode = false;
    recomputeAttributes();
}
```

Figure 2.3: The engine code that sets the perspective without verifying that the FoV is positive (engine/cpp/engine/scene/camera.cpp#109–118)

The glm library is header-only and is built from source code. Therefore, the build settings of the engine apply to glm. We suspect that the engine that is used in production is also vulnerable because the connection to the WebRTC stream was lost after issuing the modeling command shown in figure 1.1.

Exploit Scenario

An attacker sends modeling commands with a negative FoV to the engine and causes a high load due to the restarting of engine instances.

Recommendations

Short term, have the code gracefully return an error if the FoV, rendering width, or height is `0` or negative.

Long term, make sure that libraries are built in production mode and assertions do not cause aborts.

3. Engine aborts when rendering zero-area videos	
Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ZOO-3
Target: `engine/cpp/engine/graphics/RenderThread.cpp`	

Description

Attempts to render video streams with a zero value for either the width or height cause the engine to abort. The following WebSocket request crashes the engine instance.

```
export URL="ws://localhost:8080/ws/modeling/commands"
websocat "$URL?video_res_width=512&video_res_height=0"
```

Figure 3.1: A request that causes an abort

Following the above request, the engine will exit with the following message.

```
terminate called after throwing an instance of 'std::runtime_error'
  what():  image creation failed! (VK_ERROR_INITIALIZATION_FAILED)
Aborted (core dumped)
```

Figure 3.2: The termination message

The following backtrace shows the crash in more detail.

```
#7  std::terminate() () from /lib/x86_64-linux-gnu/libstdc++.so.6
#8  __cxa_throw () from /lib/x86_64-linux-gnu/libstdc++.so.6
#9  checkVk (...) at graphics/core/utils.h:27
#10 Image::createImage (…, width=512, height=0, ...) at graphics/core/image.cpp:178
#11 Image::initRenderTarget (…, width=512, height=0, …) at graphics/core/image.cpp:85
#12 Framebuffer::initImages (…) at graphics/core/framebuffer.cpp:116
#13 Framebuffer::createFramebuffer (…) at graphics/core/framebuffer.cpp:85
#14 Framebuffer::Framebuffer (… , width=512, height=0) at graphics/core/framebuffer.cpp:28
#15 VideoEncoder::VideoEncoder (…, w=512, h=0, fps=60, …) at graphics/core/videoencoder.cpp:36
#16 std::make_unique<…> () at /usr/include/c++/11/bits/unique_ptr.h:962
#17 Renderer::Renderer (…) at graphics/core/renderer.cpp:68
#18 …
#19 Graphics::RenderThread::init (…) at graphics/RenderThread.cpp:43
#20 operator() (…) at graphics/RenderThread.cpp:97
```

Figure 3.3: The associated backtrace

The engine crashes because the runtime error raised in the checkVk function is not handled in the render thread lambda during initialization.

Additionally, we found that the following Rust unit test causes an abort.

```
#[tokio::test]
async fn tob_test() {
    let mut engine_streamer =
crate::stream::EngineStreamer::new(ValidatedEngineParams::try_from(EngineParams {
        video_res_height: 0, video_res_width: 0, webrtc: true,
        ..EngineParams::default()
    }).unwrap());
    engine_streamer.start().unwrap();
    let stream = engine_streamer.video_frames();
    assert!(had_frame(stream).await);
}
```

Figure 3.4: The unit test for the above runtime error

This is because the constructor for `EngineStreamer` does not return a Rust `Result`:

```
pub fn new(ValidatedEngineParams { video_res_width, video_res_height, … }:
ValidatedEngineParams,
) -> Self {
    Self {
        ...
        engine_streamer: cpp::new_engine_streamer(
            video_res_width,
            video_res_height,
            ...
        ),
    }
}
```

Figure 3.5: The constructor for `EngineStreamer`, which is missing a `Result` return value (`engine/bridge/src/stream.rs#193–214`)

Exploit Scenario

An attacker sends a WebSocket request to render a video stream with a height of `0`, causing the engine to abort. This causes a high load due to the restarting of engine instances.

Recommendations

Short term, have the code gracefully reject the WebSocket request if the width or height is `0`. Additionally, have the render thread lambda gracefully handle runtime errors.

Long term, make sure that all Rust functions that call into C++ code return a `Result`.

4. Rendering videos with a low height and width divisible by 4 causes segmentation fault

Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ZOO-4
Target: `engine/cpp/engine/graphics/RenderThread.cpp`	

Description

Similar to finding TOB-ZOO-3, attempts to render video streams with a low height and width that are divisible by 4 cause a segmentation fault. The following WebSocket request triggers the crash.

```
export URL="ws://localhost:8080/ws/modeling/commands"
websocat "$URL?video_res_width=4&video_res_height=4"
```

Figure 4.1: A request that causes a segmentation fault

Following the above request, the engine will exit with the following message.

```
CUDA (nvenc) GPU in use: Tesla T4
CreateEncoder : m_nvenc.nvEncInitializeEncoder(m_hEncoder, &m_initializeParams)
returned error 8 at cpp/engine/_thirdparty/nvenc/NvCodec/NvEncoder/NvEncoder.cpp:309
Segmentation fault (core dumped)
```

Figure 4.2: The associated termination message

The following backtrace shows the crash in more detail. It turns out that the NvC library causes the crash.

```
#0  NvCUStream::GetInputCUStream (this=0x0) at cpp/engine/_thirdparty/nvenc/NvCUStream.h:31
#1  AppEncCuda::encodeAndReturnFrame (…) at cpp/engine/_thirdparty/nvenc/AppEncCuda.cpp:154
#2  VideoEncoder::encodeAndStoreImage (…) at cpp/engine/graphics/core/videoencoder.cpp:253
#3  Renderer::renderFrame(…) (…) at cpp/engine/graphics/core/renderer.cpp:947
#4  Scene::Scene::render (…) at cpp/engine/scene/scene.cpp:1852
#5  Scene::LiveWindowMaintainer::render (…) at cpp/engine/scene/livewindowmaintainer.cpp:74
#6  operator() (…) at cpp/engine/graphics/RenderThread.cpp:113
```

Figure 4.3: The associated backtrace

The engine crashes because of a segmentation fault in the `encodeAndReturnFrame` function. The function is reached through the render thread loop.

Similarly to our findings for TOB-ZOO-3, we found that the following Rust unit test causes an abort. There is no way to catch the crash due to the segmentation fault.

```
#[tokio::test]
async fn tob_test() {
    let mut engine_streamer =
crate::stream::EngineStreamer::new(ValidatedEngineParams::try_from(EngineParams {
        video_res_height: 4, video_res_width: 4, webrtc: true,
        ..EngineParams::default()
    }).unwrap());
    engine_streamer.start().unwrap();
    let stream = engine_streamer.video_frames();
    assert!(had_frame(stream).await);
}
```

Figure 4.4: The unit test for the above runtime error

Exploit Scenario

An attacker sends a WebSocket request to render a video stream with a height of 4, causing a crash due to the described segmentation fault. This causes a high load due to the restarting of engine instances.

Recommendations

Short term, have the code gracefully reject the WebSocket request if the width or height is below a certain threshold. We found experimentally that 256 is a valid threshold. However, documentation should be consulted to find a proper lower bound. Additionally, have the render thread lambda gracefully handle runtime errors.

5. UDP firewall can be circumvented through TURN WebRTC server

Severity: **Low**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-ZOO-5
Target: `infra/kubernetes/stunner/base/udp-route.yaml`	

Description

The TURN server and network configuration allow access to Kubernetes internal UDP services in the 10.32.0.0/24 subnet. A TURN (Traversal Using Relays around NAT) server is used in the Zoo platform to facilitate the relay of network traffic between users and Kubernetes internal engine instances. The Kubernetes cluster is configured to allow access to the full 10.0.0.0/8 network:

```
# This is potentially very nasty, so we probably shouldn't keep it.
# But, for now we're trying to eliminate a class of errors we see on the WebRTC
side.
apiVersion: stunner.l7mp.io/v1alpha1
kind: StaticService
metadata:
  name: all-local-network
  namespace: stunner
spec:
  prefixes:
    - "10.0.0.0/8"
```

Figure 5.1: The UDP network configuration (`infra/kubernetes/stunner/base/udp-route.yaml/#19–29`)

Credentials for the TURN server can be acquired through a WebSocket connection.

Based on that, it is possible to connect to the AWS DNS server. The `turncat` binary can be downloaded from the STUNner releases.

Exploit Scenario

Zoo deploys new critical infrastructure that uses the UDP protocol. An attacker uses the above technique to scan Zoo's internal network and connect to the internal UDP service. Because the service is assumed to be internal, there is no authentication, so the attacker gains full access to the service.

Recommendations

Short term, adjust the Kubernetes StaticService to block access to AWS services like the DNS server.

Long term, deploy STUNner and engine instances to a separate VPC and disallow any non-engine connections from the TURN server.

6. GitHub App webhook panics if non-ASCII characters are sent in headers

Severity: **Informational**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ZOO-6
Target: `api-deux/src/server/endpoints.rs`	

Description

The HTTP request handler thread for GitHub App webhooks panics if the `X-GitHub-Event` or `X-Hub-Signature-256` header contains non-ASCII characters.

```
let req_headers = rqctx.request.headers();
// Parse the `X-GitHub-Event` header.
let event_type_string = req_headers
    .get("X-GitHub-Event")
    .unwrap_or(&http::header::HeaderValue::from_str("")?)
    .to_str()
    .unwrap()
    .to_string();
// Parse the `X-Hub-Signature-256` header.
// See:
https://docs.github.com/en/developers/webhooks-and-events/webhooks/securing-your-web
hooks
let webhook_signature_string = req_headers
    .get("X-Hub-Signature-256")
    .unwrap_or(&http::header::HeaderValue::from_str("")?)
    .to_str()
    .unwrap()
    .to_string();
```

Figure 6.1: The code that might panic if the `HeaderValue` contains non-ASCII characters (`api-deux/src/server/endpoints.rs#3533–3548`)

The crash can be reproduced using the following command.

```
curl -v -XPOST https://api.dev.zoo.dev/apps/github/webhook -H "X-GitHub-Event:
t😺est"
```

Figure 6.2: A `curl` request that causes a panic

The bug cannot be exploited, as crashes during request handling cause the current thread to be restarted.

Recommendations

Short term, have the code handle the `Result` type gracefully and return a “400 Bad Request” error.

7. Infinite WebSocket authentication retry attempts

Severity: **Low**	Difficulty: **Low**
Type: Authentication	Finding ID: TOB-ZOO-7
Target: `api-deux/src/server/endpoints/modeling.rs`	

Description

The post-connection WebSocket authentication code allows infinite retries. The following code shows the relevant loop, which allows multiple attempts to authenticate. The outer `for` loop is executed only once. The last line of the function, which returns `None`, is not reachable. The reason for this is that the inner `while` loop iterates indefinitely over messages sent by the client.

```
async fn auth_via_websocket(
    rqctx: &RequestContext<Arc<Context>>,
    from_client: &mut SplitStream<WebSocketStream<hyper::upgrade::Upgraded>>,
) -> Option<crate::server::context::ReqContext> {
    for _ in 0..MAX_WEBSOCKET_AUTH_MSGS {
        while let Some(msg) = from_client.next().await {
            let Ok(WsMsg::Text(t)) = msg else { continue };
            let Ok(j) = serde_json::from_str::<WebsocketAuthBody>(&t) else {
                continue;
            };
            let Ok(extra_headers) = j.try_into() else { continue };
            let Ok(auth) = rqctx.context().authenticate_user_via(rqctx,
extra_headers).await else {
                continue;
            };
            return Some(auth);
        }
    }
    None
}
```

Figure 7.1: The loop that allows infinite authentication tries (`api-deux/src/server/endpoints/modeling.rs#554–572`)

As there is no general rate limiting for WebSocket messages, it is important to enforce a retry count for authentication. Zoo might expect that a rate limit on the HTTP route `/ws/modeling/commands` is enough to prevent brute-force attacks. Note, however, that an attacker attempting to brute force authentication does not necessarily need to retry random API tokens. An attacker may have access to a set of potentially valid credentials that she wants to verify.

Exploit Scenario

An attacker has access to a set of potentially valid API tokens. As there is no retry limit for inputting credentials, the attacker can open a WebSocket and try those tokens until she finds one that passes the authentication.

Recommendations

Short term, allow at most one authentication attempt. If that fails, the WebSocket connection should be closed.

8. Missing rate limiting for authentication email endpoint	
Severity: **Low**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-ZOO-8
Target: API-Deux authentication email endpoint	

Description

There is no sufficient rate limit on the `/auth/email` endpoint. The following `curl` command sends login emails to the target email address.

```
$ curl 'https://api.zoo.dev/auth/email' -H 'content-type: application/json' --data
'{"email":"maximilian.ammann@trailofbits.com","callback_url":"https://zoo.dev/"}'
```

Figure 8.1: An API request that sends login emails to the target email address

Emails are successfully sent (as indicated by success messages sent from the back end) regardless of how many times the endpoint is invoked (e.g., 20 times per minute).

Exploit Scenario

An attacker invokes the above API with a Zoo user's email address every minute continuously. The user gets annoyed by the repeated emails and complains about Zoo over social media, which results in reputational damage.

Recommendations

Short term, implement a rate limit per login email. For example, one message per hour per email address would be a reasonable limit. The rate limit should reset when the user logs in successfully. As this rate limit would contain logic, it might have to be implemented in API-Deux.

Long term, implement basic rate limiting in all endpoints.

9. Insufficient validation for callback URLs in authentication email endpoint

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ZOO-9
Target: API-Deux authentication email endpoint	

Description

The callback URL validation in the `/auth/email` endpoint is insufficient. It checks only that the URL ends with kittycad.io (see figure 9.1). Any attacker could register a domain ending with that string.

```
/// Return a valid callback URL to a kittycad domain or vercel if in dev, or None if
invalid.
pub(crate) fn valid_callback_url(url: Option<String>, env: Environment) ->
Option<String> {
    let url = match url {
        Some(url) => url,
        None => return None,
    };

    let url = match url::Url::parse(&url) {
        Ok(url) => url,
        Err(_) => return None,
    };

    let host = match url.host_str() {
        Some(host) => host,
        None => return None,
    };

    // Always allow callback urls to our domain.
    if host.ends_with("kittycad.io") || host.ends_with("zoo.dev") {
        return Some(url.to_string());
    }

    // If we are in dev, we allow vercel & local callback url
    if env != Environment::Production {
        return if host.ends_with("-kittycad.vercel.app") ||
host.contains("localhost") {
            Some(url.to_string())
        } else {
            None
        };
    }
```

```
        None
    }
```

Figure 9.1: Validation of the callback URL (api-deux/src/server/endpoints.rs/#3253–3285)

An attacker could send the following `curl` command to send a login email containing an attacker-controlled callback URL to the target email address.

```
$ curl 'https://api.zoo.dev/auth/email' -H 'content-type: application/json' --data
'{"email":"maximilian.ammann@trailofbits.com","callback_url":"https://attacker-kitty
cad.io"}'
```

Figure 9.2: The API request that sends an email containing an attacker-controlled callback URL

The user visiting the address, sent by Zoo, would be logged in and redirected to https://attacker-kittycad.io.

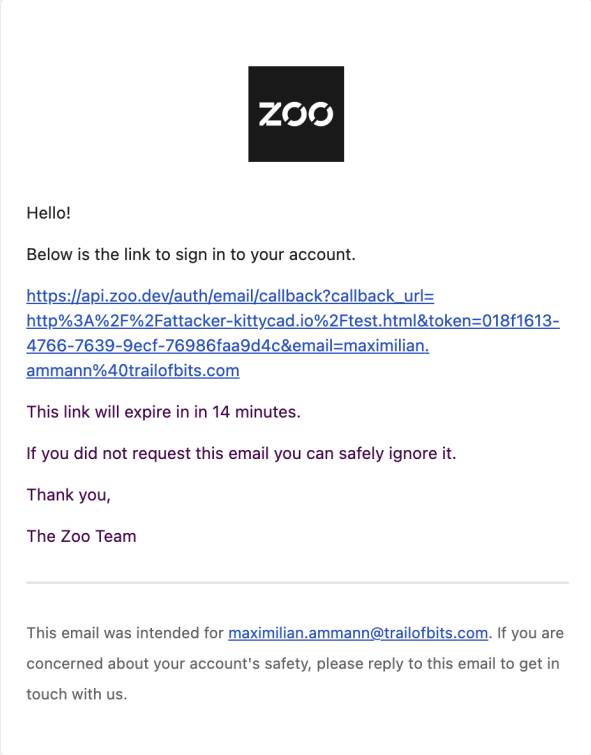

Figure 9.3: A login email sent to a user with a link that would lead to the attacker-controlled URL

The endpoint unit test is missing relevant test cases for production, including the use of attacker-controlled callback URLs in requests to the `/auth/email` endpoint. Note that the above function is also called from OAuth, SAML, and device authentication code.

Exploit Scenario

An attacker is spamming a Zoo user with authentication emails; he is able to do so because there is no rate limit on the endpoint (TOB-ZOO-8). After getting annoyed by the number of emails he is receiving, the user clicks on an authentication link, believing it is safe because it is in a trusted email originating from Zoo. The user is forwarded to a malicious website.

Recommendations

Short term, replace the current callback URL validation with an exact string comparison to allow only explicit domains. Avoid using prefix or postfix checks for any host, domain, or URL validation.

10. CORS configuration allows malicious cross-origin requests

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ZOO-10
Target: `common/src/server/http_response.rs`	

Description

The Zoo API allows cross-origin requests from any origin ending with `kittycad.io`, `zoo.dev`, or several other postfixes during production. A malicious party can register domains ending with any of the postfixes and perform cross-origin requests (e.g., using `fetch`). The following code sets the CORS headers.

```
pub async fn from_rqctx<T: Send + Sync + 'static + EnvContext>(rqctx:
&RequestContext<Arc<T>>) -> Self {
    ...
    if let Some(h) = h {
        let hostname = h.hostname();
        if hostname.ends_with("kittycad.io")
            || hostname.ends_with("zoo.dev")
            || hostname.ends_with("-kittycad.vercel.app")
            || hostname.ends_with("chat.openai.com")
            || (env != crate::types::Environment::Production &&
hostname.ends_with("localhost:3000"))
        {
            // We can return whatever origin they passed since it came from one our
our subdomains.
            // If it is a vercel domain, we assume it is a preview.
            return Self {
                access_control_allow_origin: h.to_string(),
                access_control_allow_credentials: "true".to_string(),
                access_control_allow_methods:
ACCESS_CONTROL_ALLOWED_METHODS.to_string(),
                access_control_allow_headers:
ACCESS_CONTROL_ALLOWED_HEADERS.to_string(),
            };
        }
    }
    ...
}
```

Figure 10.1: The back-end code that sets CORS headers (common/src/server/http_response.rs/#63–91)

Exploit Scenario

An attacker sends an authentication email with a link to https://attacker-kittycad.io to a Zoo user (TOB-ZOO-9). The website hosts malicious code that can perform cross-origin requests to https://api.zoo.dev.

Recommendations

Short term, adjust the back-end code that sets CORS headers to use exact string comparisons to allow only explicit domains. Avoid using prefix or postfix checks for any host, domain, or URL validation.

11. Cookie allows authentication on malicious origins

Severity: **High**	Difficulty: **Low**
Type: Authentication	Finding ID: TOB-ZOO-11
Target: `common/src/auth/session_cookie.rs:58`	

Description

The session cookie set by the back end is sent across origins. The following code defines the settings for session cookies.

```
pub fn session_cookie_header_value(token: &str, max_age: Duration, domain: &str) ->
String {
    // We don't set httponly because we want to be able to read the cookie in
    // the browser from javascript (ie. modeling-app).
    let mut cookie = format!(
        "{}={}; Path=/; Secure; SameSite=None; max-age={}",
        SESSION_COOKIE_COOKIE_NAME,
        token,
        max_age.num_seconds(),
    );
    if !domain.is_empty() {
        write!(cookie, "; Domain={domain}").unwrap_or_default();
    }

    cookie
}
```

Figure 11.1: Insecure cookie configuration (common/src/auth/session_cookie.rs/#54–68)

Because the `SameSite` attribute is set to `None`, origins that are different from the origin that receives a given `Set-Cookie` request are allowed to use the cookie.

Exploit Scenario

An attacker sends an authentication email with a link to https://attacker-kittycad.io to a Zoo user (TOB-ZOO-9). The malicious website hosts the following HTML document, which fetches the API tokens of the user.

```
<html><body><script>
    const result = fetch("https://api.zoo.dev/user/api-tokens", {
        credentials: "include",
}).then((r) => r.text().then((r) => console.dir(r)))
</script></body></html>
```

Figure 11.2: Malicious JavaScript that allows an attacker to gain access to API tokens

The attacker can then use the fetched tokens or send them to an external API. The attacker can also exploit the Zoo platform's weak CORS configuration (TOB-ZOO-10) to redirect the user back to Zoo's website so that the user does not notice the attack.

Recommendations

Short term, change the value of `SameSite` to `Lax` or `Strict`. Additionally, invalidate all existing sessions.

Long term, redact the contents of the `/user/api-tokens` endpoint to limit the impact of cross-origin attacks and XSS.

References

- MDN: `Set-Cookie` documentation

12. Plaintext secrets in the infrastructure repository

Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-ZOO-12
Target: `infra` repository	

Description
GitHub, Honeycomb, AWS, Azure, and Tailscale tokens and Slack webhooks are checked into the `infra` source code repository in Terraform `.tfstate` files. If attackers gain access to the application source code, they would have access to those secrets. Additionally, all employees and contractors with access to the repository also have access to the secrets. Secrets should never be stored in plaintext in source code repositories, as they can become valuable tools to attackers if the repositories are compromised.

Exploit Scenario
An attacker obtains a copy of the source code from a former employee. He is then able to extract the secrets and use them to exploit the Zoo infrastructure.

Recommendations
Short term, remove the hard-coded secrets from the source code and rotate their values. Avoid checking `.tfstate` files into source code repositories; instead, add them to the `.gitignore` file. The TruffleHog tool can be used to find verified access tokens. TruffleHog tests any found access tokens by using them to try to authenticate against cloud services.

```
trufflehog --only-verified git https://github.com/KittyCAD/infra
```

Figure 12.1: An example command to scan for secrets using TuffleHog

Remove the `--only-verified` flag if Trufflehog is used to also find secrets that it cannot verify automatically, such as RSA private keys.

Long term, use HashiCorp Vault or GitHub to manage secrets.

References
- GitHub: Removing sensitive data from a repository

13. Personal email in source code

Severity: **Informational**	Difficulty: **N/A**
Type: Data Exposure	Finding ID: TOB-ZOO-13
Target: `api-deux/src/server/context.rs`	

Description

In the `authenticate_user_via` function in `api-deux/src/server/context.rs`, nonemployees are barred from accessing environments in `Preview` mode. An exception is made for what appears to be a Gmail account.

This account's security properties are unknown, and its presence in Zoo's publicly available code could facilitate phishing attacks.

```
if self.environment == Environment::Preview
    && !(user.is_zoo_employee() || user.email == *"ad<…redacted…>27@gmail.com")
{
    return Err(common::server::handle_anyhow_err_and_finish(
        common::error::Error::invalid_request("You must be a Zoo employee to access
the preview environment")
            .into(),
        &a,
        &rqctx.context().db,
    )
    .await);
}
```

Figure 13.1: The check that allows only employees and a single personal email account to access the `Preview` environment (`api-deux/src/server/context.rs:372–384`)

Recommendations

Short term, remove the extra check for this email address, highlighted in figure 13.1.

Long term, ensure that employees do not use personal email accounts within the organization's code or infrastructure.

14. Unsafe code wrapped in safe API

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ZOO-14
Target: `engine/bridge/src/stream.rs`	

Description

The Rust implementation of a `Frame` shown in figure 14.1 can cause undefined behavior.

```
#[derive(Debug)]
#[repr(C)]
pub struct Frame {
    /// The pointer to the frame.
    pub ptr: *const u8,
    /// The offset.
    pub offset: u32,
    /// The size.
    pub size: u32,
    /// Timestamp
    pub timestamp: u64,
}

unsafe impl Send for Frame {}
unsafe impl Sync for Frame {}

impl Frame {
    /// Convert the frame to bytes.
    pub fn to_bytes(&self) -> Result<bytes::Bytes> {
        let frame_bytes: &[u8] =
            unsafe { std::slice::from_raw_parts(self.ptr.offset(self.offset as
isize), self.size as usize) };

        //copy the frame bytes until we can find a way to tie the bytes obj to the
frame
        Ok(bytes::Bytes::copy_from_slice(frame_bytes))

        //Ok(bytes::Bytes::from(frame_bytes)
    }
}
```

Figure 14.1: The unsafe Rust code that can cause undefined behavior (engine/bridge/src/stream.rs#72–99)

For instance, the following safe Rust program causes a segmentation fault.

```
Frame {
    ptr: std::ptr::null(),
    offset: 0,
    size: 8,
    timestamp: 0
}.to_bytes().unwrap();
```

Figure 14.2: Safe Rust code that panics

Recommendations

Short term, mark functions that could receive raw pointers—such as `to_bytes`, which receives a raw pointer in the `Frame` struct—as unsafe and add a `# Safety` comment.

Long term, review the codebase for uses of unsafe Rust and make sure that it is not possible to cause memory corruptions from safe Rust code. Make sure to always write an extensive `# Safety` comment for any such uses.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category does not apply to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

The following section contains code quality recommendations that do not have any immediate security implications.

1. Do not use the `system` function. If the engine initialization fails, additional debug information is logged using the `vulkaninfo` command line tool. To invoke the tool, the `system` function is called, which creates a shell to execute the command. Use of this function is generally discouraged, as it behaves differently on each platform and might allow an attacker to inject commands. We recommend replacing it with `execve`.

```
if(r != VK_SUCCESS)
{
//nail down some issues we're seeing in the cloud
#if LinuxBuild()
logger << Logger::error("Instance creation failed, logging additinal
information..\n");
system("vulkaninfo --summary");
#endif

//report the engine failure as we normally would
checkVk(r, "Instance creation");
}
```

Figure C.1: A use of the `system` function (engine/cpp/engine/graphics/core/instance.cpp/#170–180)

2. Do not have the rendering engine rely on global state. In various parts of the codebase, the rendering engine relies on the presence of global state and follows the singleton pattern. The use of global state could be dangerous, as it makes it more difficult to reason about the surrounding code. The following code shows an example.

```
if(auto v1 =
std::dynamic_pointer_cast<Scene::Model::Brep::Vertex>(Scene::Scene::current().getSce
neEntity(entity_uuid1, true)))
```

Figure C.2: An example use of global state in the engine (engine/cpp/endpoints/endpoints.cpp#1595)

3. Implement Send and Sync for `cpp::EngineStreamer` instead of `EngineStreamer`. It is possible to declare a C++ struct as thread-safe by unsafely implementing the Send and Sync traits. In the following code, the traits are implemented for the wrapper type instead of the C++ type. Implementing the traits directly on `cpp:EngineStreamer` would be more idiomatic. Additionally, verify whether the C++ `EngineStreamer` is really thread-safe.

```
pub struct EngineStreamer {
```

```
    engine_streamer: UniquePtr<cpp::EngineStreamer>,
}

// This should "work" since we are only using the streamer in a single thread.
unsafe impl Send for EngineStreamer {}
```

Figure C.3: The unsafe implementation of Send and Sync on the Rust `EngineStreamer` type (`engine/bridge/src/stream.rs/#102-108`)

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On June 18, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Zoo team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

Zoo provided us with GitHub pull requests (PRs) with changes addressing the findings in this report. Most issues have been resolved, except for a few whose fixes are either incomplete or postponed. Some fixes are applied only to the production environment. We believe that because the development environment is a publicly facing service, the fixes should also be applied there. Nonetheless, we marked such findings as resolved.

In summary, of the 14 issues described in this report, Zoo has resolved 11 issues and has and has not resolved three issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Risk of Rust panic due to division by zero	**Resolved**
2	Assertion on a positive FoV setting causes the engine to abort on zero- or negative-value settings	**Resolved**
3	Engine aborts when rendering zero-area videos	**Resolved**
4	Rendering videos with a low height and width divisible by 4 causes segmentation fault	**Resolved**
5	UDP firewall can be circumvented through TURN WebRTC server	**Resolved**
6	GitHub App webhook panics if non-ASCII characters are sent in headers	**Resolved**
7	Infinite WebSocket authentication retry attempts	**Resolved**
8	Missing rate limiting for authentication email endpoint	**Resolved**

9	Insufficient validation for callback URLs in authentication email endpoint	Resolved
10	CORS configuration allows malicious cross-origin requests	Unresolved
11	Cookie allows authentication on malicious origins	Resolved
12	Plaintext secrets in the infrastructure repository	Unresolved
13	Personal email in source code	Unresolved
14	Unsafe code wrapped in safe API	Resolved

Detailed Fix Review Results

TOB-ZOO-1: Risk of Rust panic due to division by zero
Resolved in PR #2153. The `NonZeroU32` type is now used to prevent a value of `0` for the `fps` field.

TOB-ZOO-2: Assertion on a positive FoV setting causes the engine to abort on zero- or negative-value settings
Resolved in PR #2057. The `setPerspective` function now checks that the FoV, screen width, and height are strictly larger than zero.

TOB-ZOO-3: Engine aborts when rendering zero-area videos
Resolved in PR #2153. The `NonZeroU32` type is now used to prevent a value of `0` for the `video_res_width` and `video_res_height` fields.

TOB-ZOO-4: Rendering videos with a low height and width divisible by 4 causes segmentation fault
Resolved in PR #2253. The minimum width and height of a video is now 256 pixels. Enforcing this lower bound prevents the engine from crashing.

TOB-ZOO-5: UDP firewall can be circumvented through TURN WebRTC server
Resolved in PR #1645. The configuration of the STUNner server has been updated so that only specific internal subnets are reachable from the internet.

TOB-ZOO-6: GitHub App webhook panics if non-ASCII characters are sent in headers
Resolved in PR #1662. The code now properly handles errors instead of using the `unwrap` function on result types.

TOB-ZOO-7: Infinite WebSocket authentication retry attempts
Resolved in PR #1663. The `while` loop has been replaced with an `if` condition. Because of that, the `auth_via_websocket` function is now processing at most two authentication requests.

TOB-ZOO-8: Missing rate limiting for authentication email endpoint
Resolved. PR #841 adds a check that limits a single authentication request per hour. PR #1616 adds more strict rate limiting through CloudFlare for the email endpoint. Rate limiting affects only the production deployment, not the development environment.

TOB-ZOO-9: Insufficient validation for callback URLs in authentication email endpoint
Resolved in PR #1662. For production setups, the domain check in `is_company_domain` now explicitly matches zoo.dev and subdomains of zoo.dev and kittycad.io. For test setups, the original concern of the finding still exists, as the domain would need to end with "-kittycad.vercel.app" or include the string "localhost," which can both be achieved by an attacker. To further lower the attack surface, we recommend further restricting the callback URL in test environments.

We found that the `is_company_domain` function does not check that the scheme/protocol is HTTPS. While we did not mention this in the initial finding, checking for the protocol would further improve the resistance against person-in-the-middle attacks if an attacker sets a plaintext HTTP redirect URL. Because both zoo.dev and kittycad.io enable HSTS only, first-time visitors would be vulnerable.

TOB-ZOO-10: CORS configuration allows malicious cross-origin requests
Unresolved. As of PR #1686, the current Nginx configuration restricts the origins to zoo.dev, kittycad.io, and subdomains thereof. However, the origin https://chat.openai.com, subdomains of vercel.app, and localhost origins are still allowed. Specifically, the arbitrary subdomains of verel.app pose a risk, as any attacker can host content on Vercel.

The Zoo team plans to move the CORS configuration from Nginx to the application layer. Due to implementation issues, CORS remains to be handled by an Nginx configuration. The Zoo team provided the following comment:

> *Updating this thread for folks coming along. We found that our handling in api-deux was not quite ready because of how it handles errors (it doesn't send cors headers there). And this was causing a bunch of issues for our frontends, basically just deleting sessions because it thought they were invalid.*
>
> *So, we're back to nginx handling cors until we can patch api-deux to correctly send headers on errors which is why this is still open.*

TOB-ZOO-11: Cookie allows authentication on malicious origins
Resolved in PR #764. The `SameSite` cookie is now set to `Strict` in production. To further lower the attack surface, we recommend also using the `Strict` property for the development environment, as it is reachable over the public internet: https://dev.zoo.dev/.

TOB-ZOO-12: Plaintext secrets in the infrastructure repository
Unresolved. Valid Honeycomb API keys and Slack Webhook URLs remain in the `infra` repository.

TOB-ZOO-13: Personal email in source code
Unresolved. The mentioned email has been removed as of commit 7a3278c. However, new ones have been added:

```
if self.environment == Environment::Preview
    && !(user.is_zoo_employee()
        || user.email.ends_with("<…redacted…>")
        || user.email.ends_with("<…redacted…>")
        || user.email.ends_with("<…redacted…>"))
    || user.email == "<…redacted…>"
```

Figure D.1: External emails in the source code (api-deux/src/server/context.rs#374–379)

TOB-ZOO-14: Unsafe code wrapped in safe API
Resolved in PR #2062. The changes restrict how a `Frame` can be initialized. It is no longer possible to directly initialize a `Frame` from safe Rust code and invoke undefined behavior. A comment was added to acknowledge that the function might exhibit undefined behavior if the C++ part of the engine misbehaves.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.