

DeepSafe Tech Zoom App Implementation

<https://www.figma.com/file/e0HdmcRMk5fDF3M3AaljSw/DeepSafe-Pitch-Deck?type=design&node-id=0-1&mode=design&t=F07pNmbd4ac0riYW-0>

[Background & General Idea](#)

[Type of App](#)

[Regular OAuth App:](#)

[Server to Server OAuth App](#)

[Tech Stack](#)

[Resources](#)

[Technological Solutions](#)

[Code](#)

[Front End Pseudocode](#)

[Back End Pseudocode](#)

We are creating DeepFake AI services for financial institutions as the target market to enhance cybersecurity within the fintech sector. This is supported through several industry and user interviews who determined that individuals likely to be cyberattacked are individuals at financial institution organizations since cyber attackers are targeting money and other finite resources.

Background & General Idea

Our idea is to develop a video conferencing integration that provides flagging, host controls, and other features to add additional security to the current video conferencing meeting room, and prevent from being deepfake attacked or zoom boomed from individuals with malicious intent that could include impersonation, defaming, and other activities that affect an individual's personal safety during workplace video conferencing meetings.

We execute this through developing methods to detect and flag the use of virtual camera software during calls. This could involve checking for known virtual camera software signatures

or anomalies in the video feed that suggest manipulation. Core features are distinguished into 2 main categories:

Meeting Controls <ol style="list-style-type: none">1. Alert System: when the software detect that there are potential synthetic video inputs2. Host Controls: Improve or automate? process for host controls → removing the user from the meeting3. Liveness detection: for users in the waiting room for joining the meeting to complete spontaneous exercises (1st stage of filtering out synthetic video)<ol style="list-style-type: none">a. automatic vs human proctored	Company Features <ol style="list-style-type: none">1. Dashboard: representing the performance of the product<ol style="list-style-type: none">a. Link detected attacks to company's internal IT department2. Support Service: to contact out company
---	--

Priority List for feature implementation

1. Alert System - Core functionality

2. Host Controls
3. Dashboard
4. Liveness Detection Test
5. Support Service

This priority list represents the core features within the solution, including the more advanced features corresponding with more advanced levels of security.

We can implement this through a couple libraries:

- <https://developers.zoom.us/docs/zoom-apps/create/> for developing a Zoom App, which is essentially a zoom personalized integration
- How can we add to the user interface of zoom to include the flagging and verification system with pop ups

Process:

1. Host installs the DeepSafe zoom plugin extension
2. When participants are in the waiting room of the zoom meeting
 - a. Participants complete waiting room liveness testing, including spontaneous commands such as "Touch your nose, hold it for 1 second, turn your head to the side, hold it for 1 second".
3. All participants and host are in the zoom meeting
4. If one of the participants is detected to have a synthetic video input or a visually altered video stream, then:
 - a. Highlight the user with a problem detected's video stream box
 - b. Send a pop up to the host: "Mary Mourtazo may be committing suspicious behavior. Would you like to remove this user from the meeting?" Yes, No buttons
 - c. Confirmation with user about potential removal

- d. Send a pop up to the host: "Would you like to report this user" Yes, No buttons
- e. Message: "Thank you for protecting zoom"
- 5. From the malicious actor: "the host has removed you from the meeting due to suspicious activity"

Questions:

- How can we incorporate this tech stack into developing the zoom integration?
-

Type of App

ChatGPT response

As of my last update in January 2022, Zoom's developer documentation may have evolved, but I can provide a general understanding of the difference between a Server to Server OAuth app and a regular OAuth app based on typical OAuth principles.

Regular OAuth App:

- In a regular OAuth app, the authentication flow involves a user interacting with the OAuth provider (in this case, Zoom) to authorize the third-party application (client) to access their Zoom account on their behalf.
- The OAuth flow typically includes redirecting the user to Zoom's authorization server, where they log in, grant permissions, and are redirected back to the third-party application with an authorization code.
- The third-party application then exchanges this authorization code for an access token, which it can use to make API requests on behalf of the user.
- This type of OAuth flow is often used for scenarios where a user is directly involved in granting access to their account, such as integrating with user-specific data or actions.

Server to Server OAuth App

- A Server to Server OAuth app, also known as client credentials flow or two-legged OAuth, typically does not involve user interaction.
- Instead, it is used when a third-party application (client) needs to access resources or APIs on behalf of itself rather than on behalf of a user.
- In this flow, the third-party application authenticates directly with the OAuth provider using its own credentials (client ID and client secret) without requiring user authorization.
- Upon successful authentication, the OAuth provider issues an access token directly to the third-party application.
- This access token can then be used by the third-party application to make API requests to the OAuth provider's APIs on its own behalf.

- Server to Server OAuth is commonly used in scenarios where an application needs to access its own resources or perform tasks autonomously without user intervention.

In summary, the main difference lies in whether the OAuth flow involves user interaction (regular OAuth) or not (Server to Server OAuth), and whether the access token is obtained on behalf of a user (regular OAuth) or for the application itself (Server to Server OAuth). Depending on the use case and access requirements, one approach may be more appropriate than the other.

Tech Stack

Zoom Integration:

- Utilize the Zoom Apps platform to create a personalized integration.
- Develop the integration using Zoom's API for seamless communication with the Zoom platform.
- Implement UI modifications within the Zoom interface to incorporate flagging and verification features, such as adding buttons or indicators for flagged users.

Video Analysis and Detection:

- Use computer vision libraries like OpenCV for analyzing video streams and detecting anomalies.
- Implement algorithms to detect virtual camera software signatures and visual alterations in real-time video feeds.
- Integrate machine learning models for more advanced detection of deepfake content.

Backend Services:

- Develop backend services to handle user authentication, data storage, and communication between the Zoom integration and the detection algorithms.
- Use cloud platforms like AWS or Azure for scalability and reliability.

User Interface:

- Design user-friendly interfaces for both hosts and participants within the Zoom environment.
- Use web technologies such as HTML, CSS, and JavaScript for UI components.
- Implement pop-up notifications and interactive elements for alerting hosts and users about suspicious behavior.

Dashboard and Reporting:

- Develop a dashboard to provide insights into the performance of the security features.
- Integrate reporting mechanisms to link detected attacks to the company's internal IT department for further investigation.

Certainly! Below are some existing APIs, libraries, and resources that you can leverage to implement synthetic video detection, Zoom integration, and other technical aspects of your project:

1. Synthetic Video Detection API:

- Deepware Scanner API: Deepware Scanner provides an API for detecting deepfakes and manipulated videos. It offers both cloud-based and on-premise solutions for integrating into your application.
- Sensity API: Sensity provides an API for detecting deepfake videos and other forms of synthetic media. It offers real-time detection and analysis capabilities for video content.

2. Zoom API:

- Zoom Developer Platform: The Zoom Developer Platform offers APIs and SDKs for building custom Zoom integrations. You can use the Zoom API to manage meetings, users, and other aspects of the Zoom platform programmatically.
- Zoom App Marketplace: Explore the Zoom App Marketplace for existing integrations and plugins that may be useful for your project.

3. Technical Libraries for Video Analysis:

- OpenCV: OpenCV is a popular computer vision library that provides tools for image and video analysis. You can use OpenCV to detect anomalies and manipulations in video streams.
- TensorFlow / PyTorch: TensorFlow and PyTorch are deep learning frameworks that offer pre-trained models for detecting deepfake videos. You can fine-tune these models or train your own using custom datasets.
- FFmpeg: FFmpeg is a multimedia framework that provides tools for decoding, encoding, and processing video and audio files. You can use FFmpeg to extract frames from video streams and perform analysis.

4. Zoom Integration Resources:

- Zoom Developer Documentation: The Zoom Developer Documentation provides comprehensive guides, reference materials, and code samples for integrating with the Zoom platform. It covers topics such as authentication, API endpoints, and best practices for building Zoom Apps.
- Zoom API Reference: Refer to the Zoom API Reference for detailed information about available endpoints, request parameters, and response formats.
- Zoom Developer Forums: The Zoom Developer Forums are a valuable resource for getting help, sharing ideas, and troubleshooting issues related to Zoom integration development.

5. Additional Resources:

- GitHub: Explore GitHub repositories for open-source projects, code samples, and libraries related to synthetic video detection, Zoom integration, and computer vision.
- Technical Blogs and Tutorials: Look for technical blogs and tutorials that cover topics such as deepfake detection techniques, Zoom API integration, and video analysis algorithms.

Resources

Technological Solutions

How easy it is to create deep fakes

https://www.youtube.com/watch?v=Bb0XvAQcntY&ab_channel=FaceTec

FaceTec

<https://www.facetec.com/>

Liveness detection:

<https://www.zoomlogin.com/FaceTec-Whitepaper-Liveness-Detection-Final-Frontier-1.pdf>

Go with the API for now, but:

1. Sign up for zoom developer account
2. Test process of doing it ourselves with a virtual camera
3. Looking at the QOS API data to see if there's a significant difference (quality of service)

<https://support.google.com/meet/thread/117874022/is-there-any-feature-which-allows-the-host-to-see-if-the-attendees-are-using-a-virtual-camera?hl=en>

- Information with the Meet Quality Tool via the Admin console.
- Quality Tool

Code

Participant Flagging

JavaScript

```
// Function to flag the participant's video stream
function flagParticipant(participantId) {
  // Send a request to the backend to flag the participant
  // You can use AJAX or fetch API to send the request

  // For demonstration purposes, let's assume we have an endpoint to send the
  flag
  const flagEndpoint = '/flagParticipant';
```

```

// Example of using fetch API to send a POST request to the backend
fetch(flagEndpoint, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ userId: participantId }) // Pass participant's
ID to backend
})
.then(response => {
  if (!response.ok) {
    throw new Error('Failed to flag participant');
  }
  // If the request is successful, display an alert
  alert("Participant flagged for suspicious behavior!");
})
.catch(error => {
  // Handle errors
  console.error('Error flagging participant:', error);
  alert("Failed to flag participant. Please try again later.");
});
}

// Function to handle meeting participant events using Zoom Web SDK
function handleParticipantEvents() {
  // Listen for participant joined event
  ZoomMtg.inMeetingServiceListener('user-added', function (data) {
    const participantId = data.userId;
    flagParticipant(participantId);
  });
}

// Initialize Zoom Web SDK and handle participant events
function initializeZoomSDK() {
  ZoomMtg.setZoomJSLib('https://source.zoom.us/1.9.1/lib', '/av');
  ZoomMtg.preLoadWasm();
  ZoomMtg.prepareJssdk();

  const meetingConfig = {
    apiKey: 'YOUR_ZOOM_API_KEY',
    meetingNumber: 'MEETING_NUMBER',
    userName: 'HOST_NAME',
    passWord: 'MEETING_PASSWORD',
    leaveUrl: 'YOUR_LEAVE_URL',
    role: 1 // 1 for host, 0 for participant
  };

  ZoomMtg.init({
    leaveUrl: meetingConfig.leaveUrl,
    isSupportAV: true,
    success: function () {
      ZoomMtg.join({
        meetingNumber: meetingConfig.meetingNumber,
        userName: meetingConfig.userName,
        signature: 'GENERATED_SIGNATURE',
        apiKey: meetingConfig.apiKey,

```

```

        passWord: meetingConfig.passWord,
        success: function (res) {
            console.log("Join meeting success: ", res);
            handleParticipantEvents();
        },
        error: function (res) {
            console.error("Join meeting failed: ", res);
        }
    });
},
error: function (res) {
    console.error("Zoom SDK initialization failed: ", res);
}
});
}
}

// Initialize Zoom SDK when the page loads
window.onload = function () {
    initializeZoomSDK();
};

```

Calling the Sensity API

<https://docs.sensity.ai/#section/Introduction>

```

JavaScript
// Function to call Sensity API for deepfake detection
async function detectDeepfake(videoClip) {
    // Sensity API endpoint for deepfake detection
    const sensityEndpoint = 'https://api.sensity.io/v1/deepfake/detect';

    // API key for accessing Sensity API
    const apiKey = 'YOUR_SENSITY_API_KEY';

    // Construct the request body with the video clip data
    const formData = new FormData();
    formData.append('video', videoClip);

    try {
        // Send a POST request to Sensity API
        const response = await fetch(sensityEndpoint, {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${apiKey}`
            },
            body: formData
        });

        if (!response.ok) {
            throw new Error('Failed to detect deepfake');
        }
    }
}

```



```

    // Parse the response JSON
    const result = await response.json();
    return result;
  } catch (error) {
    console.error('Error detecting deepfake:', error);
    throw error;
  }
}

// Function to handle recording and passing the video clip to Sensity API
function handleVideoRecording() {
  // Code for recording video clip through Zoom app integration
  // For demonstration purposes, let's assume we have the video clip data stored
  // in a variable named 'videoClip'

  // Call Sensity API for deepfake detection
  detectDeepfake(videoClip)
    .then(result => {
      // Process the detection result
      console.log('Deepfake detection result:', result);
    })
    .catch(error => {
      // Handle errors
      console.error('Error detecting deepfake:', error);
    });
}

// Call the function to handle video recording and deepfake detection
handleVideoRecording();

```

Front End Pseudocode

1. User Interface Initialization:
 - Create HTML elements for the meeting interface, including video containers and controls.
 - Add event listeners to UI elements for user interaction.
2. Start Meeting:
 - Implement functionality to initiate a meeting through the Zoom integration.
 - Display meeting details such as meeting ID and join link on the UI.
3. Record Video Clip:
 - Provide a button or control to allow the host to start recording the meeting.
 - Upon clicking the record button:
 - Trigger Zoom API to start recording the meeting.
 - Display a visual indication that recording is in progress.

4. Alert System:

- Implement a mechanism to detect deepfake or suspicious behavior during the meeting.
- If suspicious activity is detected:
 - Highlight the participant's video stream on the UI.
 - Display a pop-up notification to the host about the detected activity.
 - Provide options for the host to take action (e.g., remove participant, report).

5. Handle User Interaction:

- Listen for user actions such as clicking buttons or interacting with video streams.
- Implement corresponding functionality to respond to user actions (e.g., flagging a participant).

6. Error Handling:

- Implement error handling mechanisms to handle failures in API calls or unexpected events.
- Display error messages or notifications to users to inform them about any issues.

7. User Feedback:

- Provide feedback to users about the status of their actions (e.g., recording status, detection results).
- Display informative messages or notifications to keep users informed about the meeting's progress.

8. Styling and UI Enhancements:

- Apply CSS styles to the UI elements to enhance visual appeal and user experience.
- Ensure the UI layout is intuitive and user-friendly for seamless interaction during the meeting.

9. Continuous Monitoring:

- Implement mechanisms to continuously monitor the meeting for suspicious activities or deepfake incidents.
- Update the UI dynamically to reflect changes in meeting status or detection results.

10. Integration with Backend:

- Communicate with the backend server to exchange data and perform actions such as flagging participants or reporting incidents.
- Send API requests to the backend to trigger backend processes based on user interactions or detection results.

11. Testing and Debugging:

- Conduct thorough testing of the frontend components to ensure funct

Back End Pseudocode

1. Authentication and Authorization:

- Implement authentication mechanisms to verify the identity of users and ensure they have the necessary permissions.
- Authenticate requests from the frontend using tokens or session management.

2. Endpoint Setup:

- Define API endpoints for handling requests from the frontend and integrating with external services (e.g., Zoom, Sensity).
- Implement route handlers for each endpoint to process incoming requests.

3. Zoom Integration:

- Set up API routes to handle Zoom-related functionalities, such as starting meetings, recording meetings, and retrieving recorded video clips.
- Implement logic to interact with the Zoom API, authenticate requests, and handle responses.

4. Deepfake Detection:

- Define API routes for deepfake detection functionalities, including receiving video clips and initiating deepfake detection.
- Implement logic to communicate with the Sensity API, authenticate requests, and handle detection results.

5. Alert System:

- Implement mechanisms to handle alerts triggered by suspicious activities detected during meetings.
- Define routes and logic to process alerts, notify users, and take appropriate actions (e.g., removing participants, sending notifications).

6. Data Management:

- Set up database schemas and models to store information related to meetings, participants, recorded video clips, and detection results.
- Implement CRUD operations to manage data stored in the database.

7. Error Handling:

- Implement error handling middleware to catch and process errors that occur during request processing.
- Return appropriate error responses with meaningful error messages to the frontend.

8. Logging and Monitoring:

- Configure logging mechanisms to record important events, API requests, and system activities.
- Monitor system performance, track API usage, and analyze logs to identify issues and optimize performance.

9. Security:

- Implement security measures such as input validation, data sanitization, and encryption to protect against security threats and vulnerabilities.
- Ensure secure communication between the frontend and backend using HTTPS and secure authentication mechanisms.

10. Integration with Frontend:

- Define APIs and endpoints to facilitate communication between the frontend and backend.
- Implement logic to process requests from the frontend, interact with external services, and return appropriate responses.

11. Testing and Debugging:

- Write unit tests and integration tests to ensure the reliability and correctness of backend functionalities.
- Debug issues, errors, and inconsistencies in the backend code to ensure smooth operation of the application.

12. Continuous Integration and Deployment (CI/CD):

- Set up CI/CD pipelines to automate the build, testing, and deployment processes.
- Continuously deploy updates and enhancements to the backend codebase to improve functionality and address issues.