49_Phase2_ESA_Report.docx

3,064 Words

6% Matches

1	Internet	github.com	<1%
2	Internet	huggingface.co	<1%
3	Internet	gitlab.com	<1%
4	Internet	vdocuments.com.br	<1%
5	Internet	archive.org	<1%
6	Internet	www.medevel.com	<1%
7	Internet	skannai.medium.com	<1%
8	Internet	codemax.app	<1%
9	Internet	medium.com	<1%
10	Internet	towardsdatascience.com	<1%

11	Internet	parkerintercom.com	<1%
12	Internet	programtalk.com	<1%

ABSTRACT

CCTV systems are an absolute necessity in modern surveillance, but their use is very vast covering everything from banking to transportation, healthcare to public safety. But these systems are becoming increasingly vulnerable to cyber threats, where replay attacks being one of the main threats. An alteration of video authentication can happen due to replay attacks, which involve capturing any valid stream and then re-recording it and putting it into the system to deceive the system and providing access to the camera, which could change the video and compromise the whole security of the system. This highlights the need for better security especially around communication protocols that which aren't strict enough

To tackle these challenges, we are proposing an AI-based detection and response system specifically designed to mitigate replay attacks in CCTV networks. By integrating machine learning and cryptographic security measures, the system aims to detect, prevent, and respond to replay attacks in real-time, due to which we can rely on CCTV systems without any fear.

The main components are:

1. AI-Powered Anomaly Detection: We will be utilizing ML models to analyse video streams and network traffic for anomalies that indicate replay attacks, such as inconsistencies in timestamps and frame sequences.

2. Secure Communication Protocols: We plan on implementing encryption and/or digital signatures in order to ensure secure data transmission and to verify that video streams are not tampered with, and that only legitimate data is processed.

3. Real-Time Automated Response: We will be working on a threat response module that takes appropriate action in near-real time upon detecting an attack, including alerting administrators, isolating compromised devices, and blocking malicious traffic.

4. Modular Design: We will try our best to design this system so that it can seamlessly integrate with existing CCTV systems, ensuring compatibility with a lot of hardware and software configurations.

This project will nvolve a comprehensive vulnerability assessment of current CCTV systems, training ML models on datasets simulating replay attacks, and testing in controlled environments. The goal is to make a scalable, effective and user-friendly system.

It is anticipated that effective implementation will yield better detection of replay attacks, thus resulting in enhanced security in CCTV systems, reduced susceptibility to cyber threats, and thus making surveillance technologies more reliable. Hence the project is a step forward towards intelligent, adaptable, and self-Secured surveillance technologies

1.INTRODUCTION

CCTV systems have become an integral part of security infrastructures, utilities that offer real-time surveillance and recording capabilities to deter threats and protect sensitive surroundings such as banks, data hubs, transportation hubs, and public areas. But these systems are vulnerable to replay attacks, a sophisticated type of cyberattack in which hackers intercept and capture real video footage or commands sent by the system, and retransmit the data at some point in the future. By replaying old footage or commands, attackers can deceive systems into displaying false information as live data, and thus bypass real-time monitoring mechanisms.

Replay attacks exploit the weaknesses in communication protocols that aren’t secured properly or old/legacy CCTV systems that don’t have modern safeguards such as strong encryption, authentication, or tamper detection mechanisms. This can lead to really bad consequences, including:

1. Bypassing Real-Time Surveillance:

Attackers may be able to replace live(legitimate) video streams with previously recorded footage (tampered data), masking their current activities or making the system believe that everyting is normal while intrusions or unauthorized actions happen. These compromises of system’s ability to detect and respond to real-time threats.

2. Creating Security Blind Spots:

By replaying old footage, hackers may acquire the ability to hide unauthorized access or any movements in sensitive areas such as secure bank vaults, server rooms, or restricted zones in airports and other such areas. These undetected intrusions can lead to theft, sabotage, or unauthorized access to critical information and devices.

3. Manipulating Evidence:

Surveillance footage is used many times as evidence in legal investigations and cases. Replay attacks can tamper with original recordings, leading to timelines that aren’t accurate, display of events that never occurred, or data can go missing, which potentially derail investigations and undermine judicial processes.

4. Exploiting Access Control Systems:

11 Quite a few of the modern access control systems are integrated with CCTV systems for enhancing the security, such as monitoring entries and exits. Replay attacks may exploit this integration by retransmitting any old footage of authorized entries, thus granting attackers unauthorized access without raising any alarms.

5. Targeting Legacy Systems:

Older CCTV cameras have outdated firmware which may lead to high exposure. Threat actors can easily attack these CCTV cameras and do replay attacks on the CCTV cameras and do replay attacks. They can exploit both physical and digital parts of the system. This shows that we must ensure that proper protocols are implemented and all the firmware’s are updated in order to prevent any kinds of replay attacks from threat actors in future.

2.PROBLEM STATEMENT

“Detection and Mitigation of Replay Attacks on CCTV Systems”

3.DATA

3.1 OVERVIEW

For this project we will be using the UCF-Crime dataset, it was developed by the UCF Center for research in Computer Vision(CSRV) which is a large scale benchmark for real world anomaly detection in surveillance videos. It was introduced in “CVPR 2018” by Waqas Sultani, Chen and Mubarak Shah, it provides a comprehensive dataset for evaluating security-based AI models.

The Key Features of this dataset are:

1. Scale and Composition

1. It has 1900 untrimmed surveillance videos (approx. 128 hrs of footage).
2. Mostly Real-world scenarios, covering diverse environments.

2. Anomalous Events

1. It includes 13 crime categories such as: Assault, Arson, Robbery, Shooting, Theft.
2. Also contains normal activity videos for balancing training

3. Weak Supervision Approach

1. Labels are provided at the video level
2. No precise temporal annotations, simulating real-world uncertainty in anomaly detection.

The research contributions

1. Multiple Instance Learning (MIL)

It’s a deep MIL framework which treats each video as a bag and its segments as instances, enabling models to detect anomalies without requiring exact timestamps

2. **Ranking Loss with Constraints**

   The dataset introduced a ranking loss function incorporating sparsity and temporal smoothness to improve anomaly localization.

3. **Benchmark Standard**

   Used widely for evaluating anomaly detection models, ensuring robustness in real-world surveillance applications.

The UCF crime dataset is a cornerstone in video anomaly detection research which empowers security and AI professionals to enhance CCTV surveillance intelligence.

3.2 Datasets

The UCF crime dataset is widely recognized for real world anomaly detection in surveillance systems, however we are using this dataset in our project for the following tasks:

1. Simulating Replay attack
2. Creating Replay Dataset
3. Training the model to detect replay attacks

Unlike conventional datasets it provides untrimmed security footage, enabling the model to detect anomalies in unpredictable environments.

Breakdown anomalous versus normal activities

Anomalous events (Crimes And incidents)

1.Violent crimes: assault, abuse, fighting and shooting.

2.Property crimes: arson, burglary, shoplifting, vandalism

3.High Risk events: Explosion and road accident

4.Law Enforcement actions: Arrests and Abuse

Normal Activities

1.Foot Traffic in public spaces

2.Routine vehicle movement, non-suspicious pedestrian behaviour

3.Background scenes without criminal activity

Attackers would want to tamper with such footages in a way such that the crime is covered up. One of the was they could do so is a replay attack where they inject pre-recorded normal footage in the place of crime footage Our project aims to simulate this and also use this dataset to provide an approach to detect such replay attacks effectively.

3.3 Data Pre-processing

Raw surveillance videos contain noise, varying resolutions and unpredictable version patterns, that hinder effective anomaly detection. Pre-processing ensures standard input, enhancing the accuracy of detection modules.

The different steps involved in pre-processing are:

1. Frame standardization: Resized the frames to 640X480 for consistency and used gray scale frames to reduce complexity.
2. Motion extraction using optical flow: Applied farneback optical flow algorithm to analyse pixel wise motion vectors between consecutive frames and capture movement patterns than raw pixel intensity, making anomaly detection more robust.
3. Data storage optimization: Compression of extracted motion vectors in **.npz** files using numpy to ensure efficient storage and retrieval and allowed rapid access for real time detection.
4. Frame Sampling for processing efficiency: Periodically extracted frames to reduce redundancy and optimized computational load without losing essential anomaly cues

4.DESIGN DETAILS

4.1 Novelty

Our project focuses majorly on real time anomaly detection system for CCTV systems which integrates adaptive replay attack detection. So the key innovation lies in:

1. **Hierarchical temporal memory and hashing techniques** for anomaly verification.
2. Integration of anomaly detection with RTSP based replay detection (MediaMTX)

4.2 Innovativeness

Unlike any other frame based anomaly detection, our system focuses on:

1. Motion Based optical flow analysis to detect anomalies
2. Adopts weak supervision which allows detection without exact timestamp

4.3 Interoperability

1. It is compatible across surveillance infrastructures (RTSP, FFmpeg, OpenCV)
2. It works with various video formats, thus ensuring deployment flexibility.
3. It can integrate with existing alert systems, expanding functionality in security Environments.

4.4 Performance

1. Optimized frame processing using OpenCV and NumPy which minimizes latency.

2. Motion based detection which ensures fewer false alarms which is better than static frame-based methods.

4.5 Security

1. Usage of hashing-based frame integrity verification which ensures tamper detection
2. Alerts that can trigger automated lockdown mechanisms in surveillance zones.

4.6 Reliability

1. It is designed for real world noisy environments which improves resilience.
2. There is continuous monitoring and adaptation via hierarchical anomaly scoring(HTM)
3. There are redundant verification layers which ensure false positives are minimized.

4.7 Maintainability

1. Supports new model integration (HTM, deep learning variants)
2. It is a modular python-based framework which allows easy upgrades.
3. It is an API-based structure which simplifies debugging and enhancements.

4.8 Portability

1. It can run seamlessly on cloud, edge devices or local servers.
2. It can work on multiple OS environments (Windows, Linux…)
3. It has very lightweight dependencies to ensure minimal system strain.

4.9 Legacy to modernization

1. Transforms security operations from passive monitoring to a pro-active threat detection software.
2. Provides adaptive intelligence which replaces outdated rule-based methods.
3. It enhances traditional CCTV surveillance with AI-powered detection

4.10 Reusability

1. It has a modular design which allows component-based reuse across different AI-security applications.

2. this framework can be re-purposed for other anomaly detection scenarios (eg.: industrial security)

4.11 Application compatibility

1. It is compatible with cloud security infrastructures for remote anomaly monitoring.

2. It can scale across multiple surveillance cameras without compromising their efficiency.

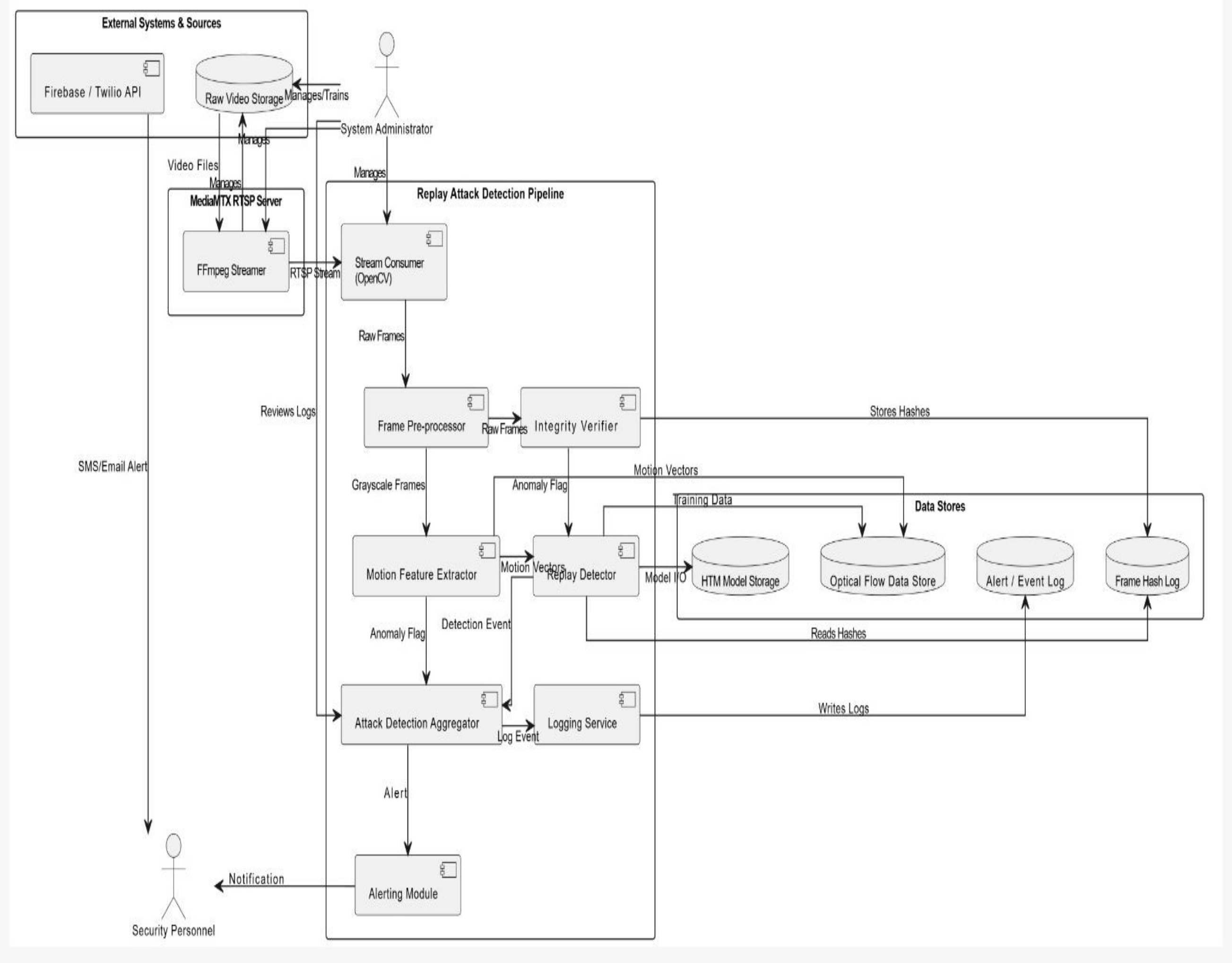
3. Integrates with RTSP-based CCTV networks.

4.12 Resource Utilization

1. It has an optimized storage which are compressed into .npz data format

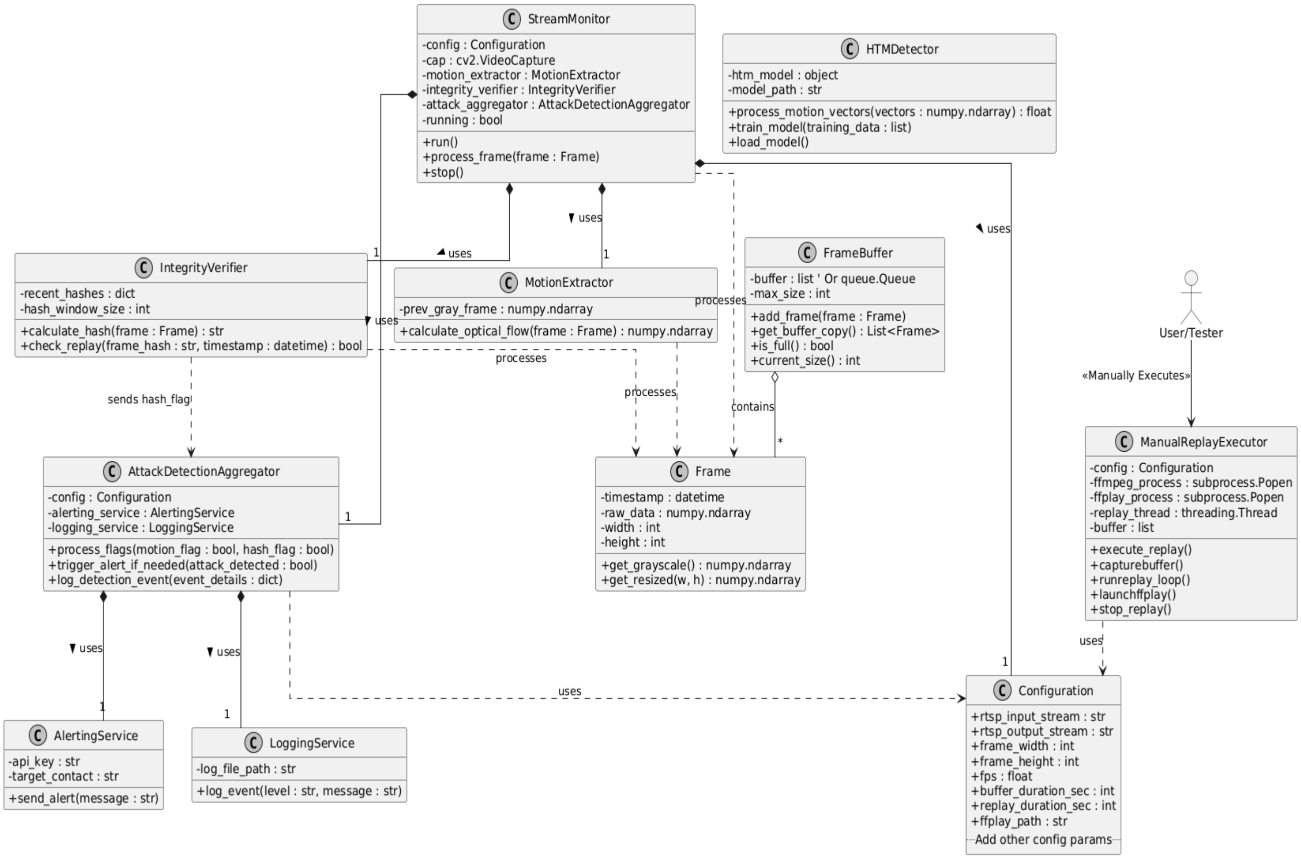
2. It has efficient processing with OpenCV and NumPy which reduces computational overhead.

5.HIGH LEVEL SYSTEM DESIGN / SYSTEM ARCHITECTURE

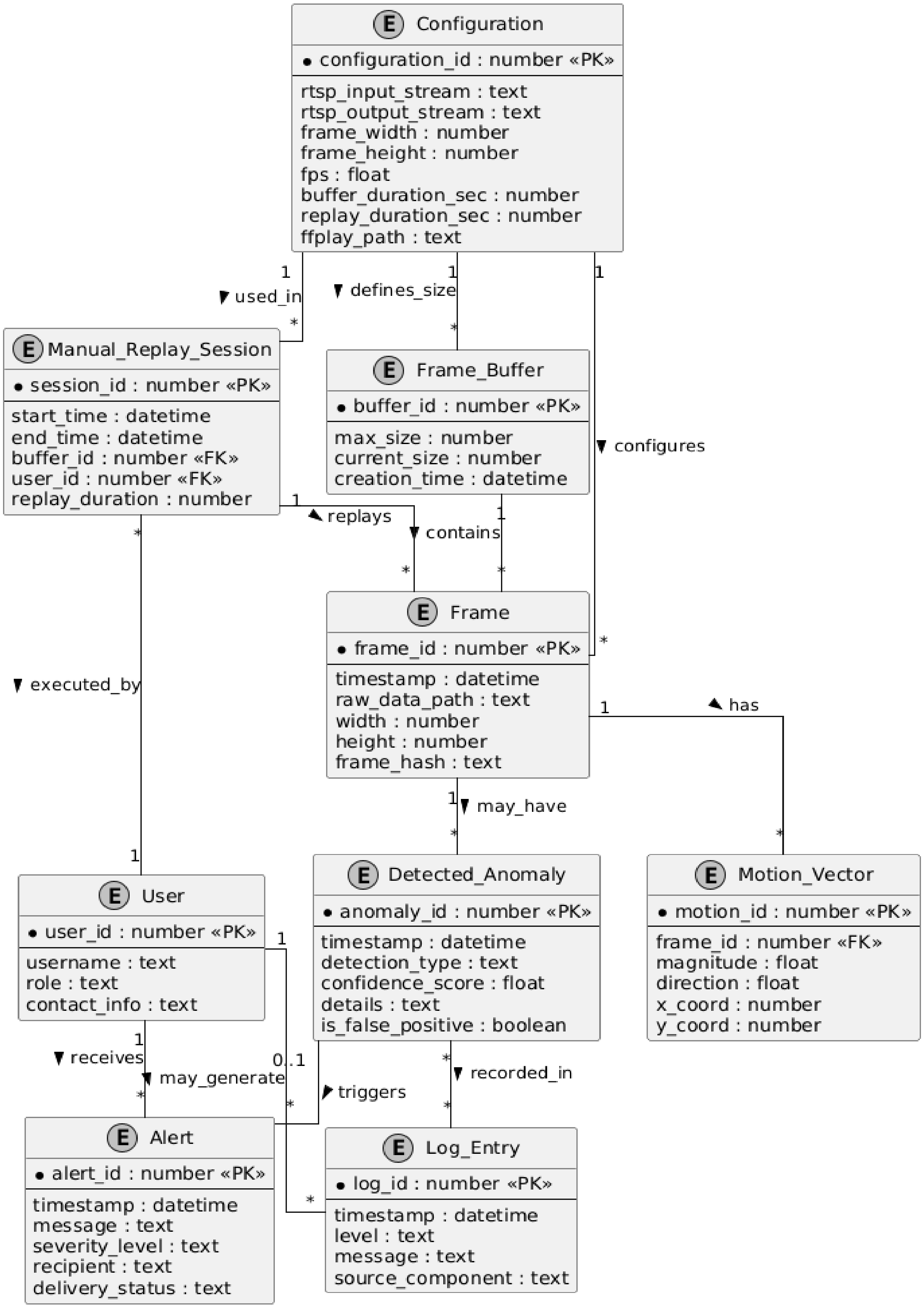


6.DESIGN DESCRIPTION

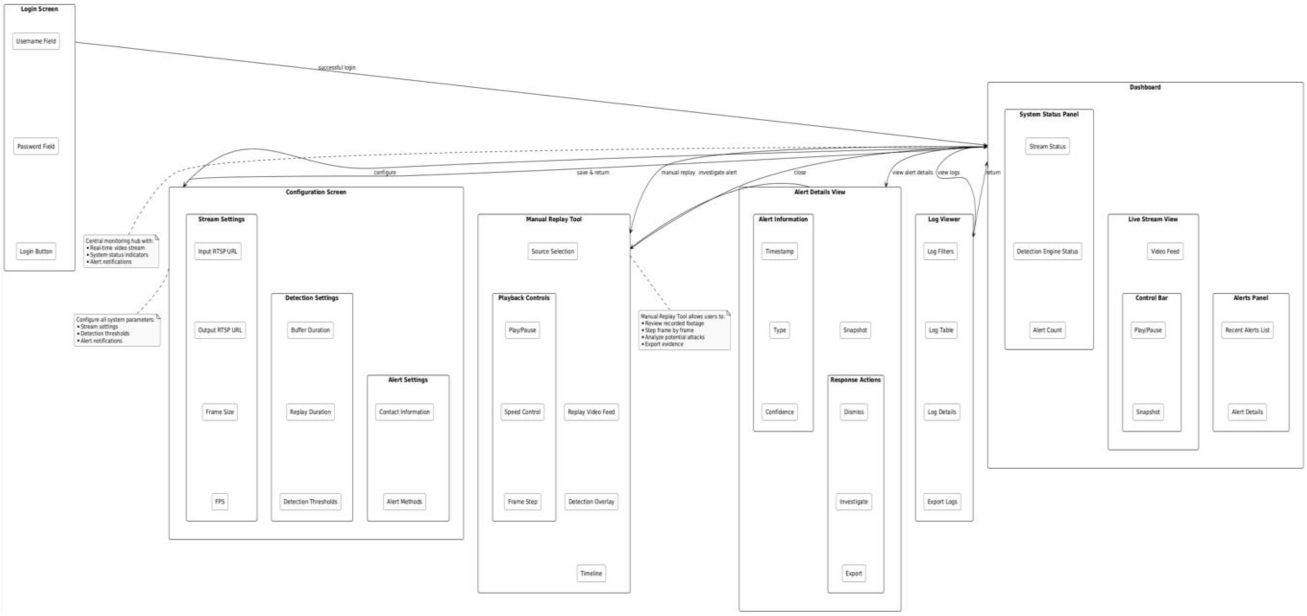
6.1 Master Class Diagram


StreamMonitor
-config : Configuration
-cap : cv2.VideoCapture
-motion_extractor : MotionExtractor
-integrity_verifier : IntegrityVerifier
-attack_aggregator : AttackDetectionAggregator
-running : bool
+run()
+process_frame(frame : Frame)
+stop()
HTMDetector
-htm_model : object
-model_path : str
+process_motion_vectors(vectors : numpy.ndarray) : float
+train_model(training_data : list)
+load_model()
uses
1
uses
1
uses
IntegrityVerifier
-recent_hashes : dict
-hash_window_size : int
+calculate_hash(frame : Frame) : str
+check_replay(frame_hash : str, timestamp : datetime) : bool
MotionExtractor
-prev_gray_frame : numpy.ndarray
+calculate_optical_flow(frame : Frame) : numpy.ndarray
FrameBuffer
-buffer : list ' Or queue.Queue
-max_size : int
+add_frame(frame : Frame)
+get_buffer_copy() : List<Frame>
+is_full() : bool
+current_size() : int
processes
uses
processes
processes
contains
*
sends hash_flag
User/Tester
«Manually Executes»
AttackDetectionAggregator
-config : Configuration
-alerting_service : AlertingService
-logging_service : LoggingService
+process_flags(motion_flag : bool, hash_flag : bool)
+trigger_alert_if_needed(attack_detected : bool)
+log_detection_event(event_details : dict)
1
Frame
-timestamp : datetime
-raw_data : numpy.ndarray
-width : int
-height : int
+get_grayscale() : numpy.ndarray
+get_resized(w, h) : numpy.ndarray
ManualReplayExecutor
-config : Configuration
-ffmpeg_process : subprocess.Popen
-ffplay_process : subprocess.Popen
-replay_thread : threading.Thread
-buffer : list
+execute_replay()
+capturebuffer()
+runreplay_loop()
+launchffplay()
+stop_replay()
uses
uses
uses
uses
1
1
1
AlertingService
-api_key : str
-target_contact : str
+send_alert(message : str)
LoggingService
-log_file_path : str
+log_event(level : str, message : str)
Configuration
+rtsp_input_stream : str
+rtsp_output_stream : str
+frame_width : int
+frame_height : int
+fps : float
+buffer_duration_sec : int
+replay_duration_sec : int
+ffplay_path : str
Add other config params

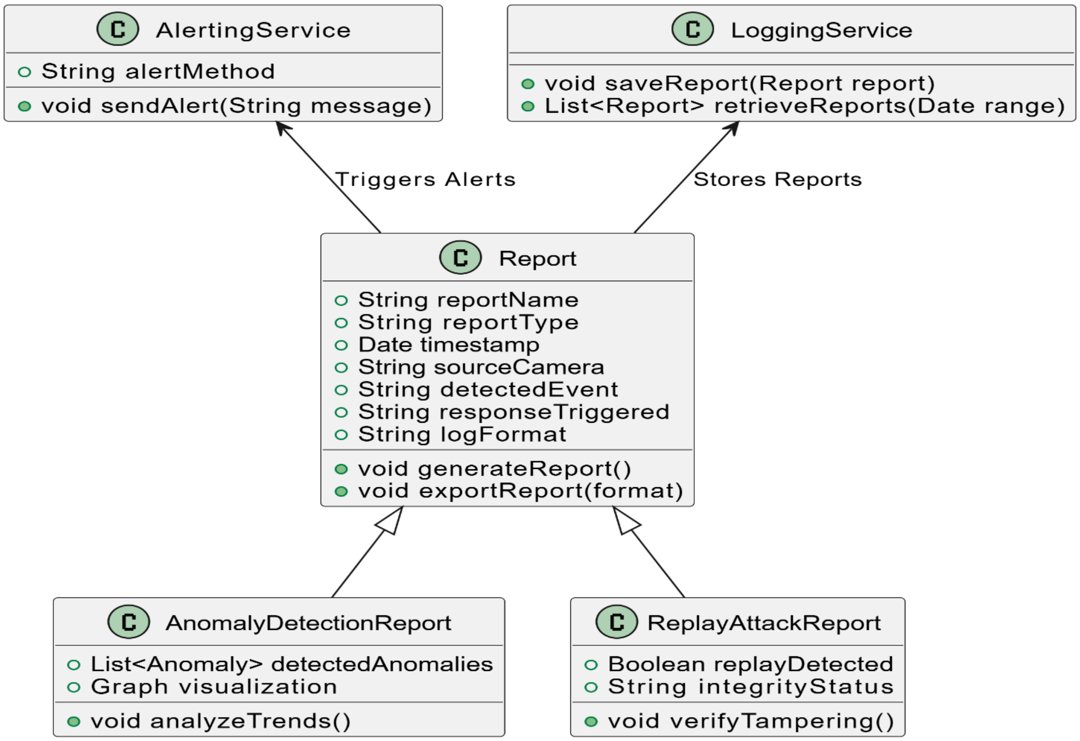
6.2 ER Diagram



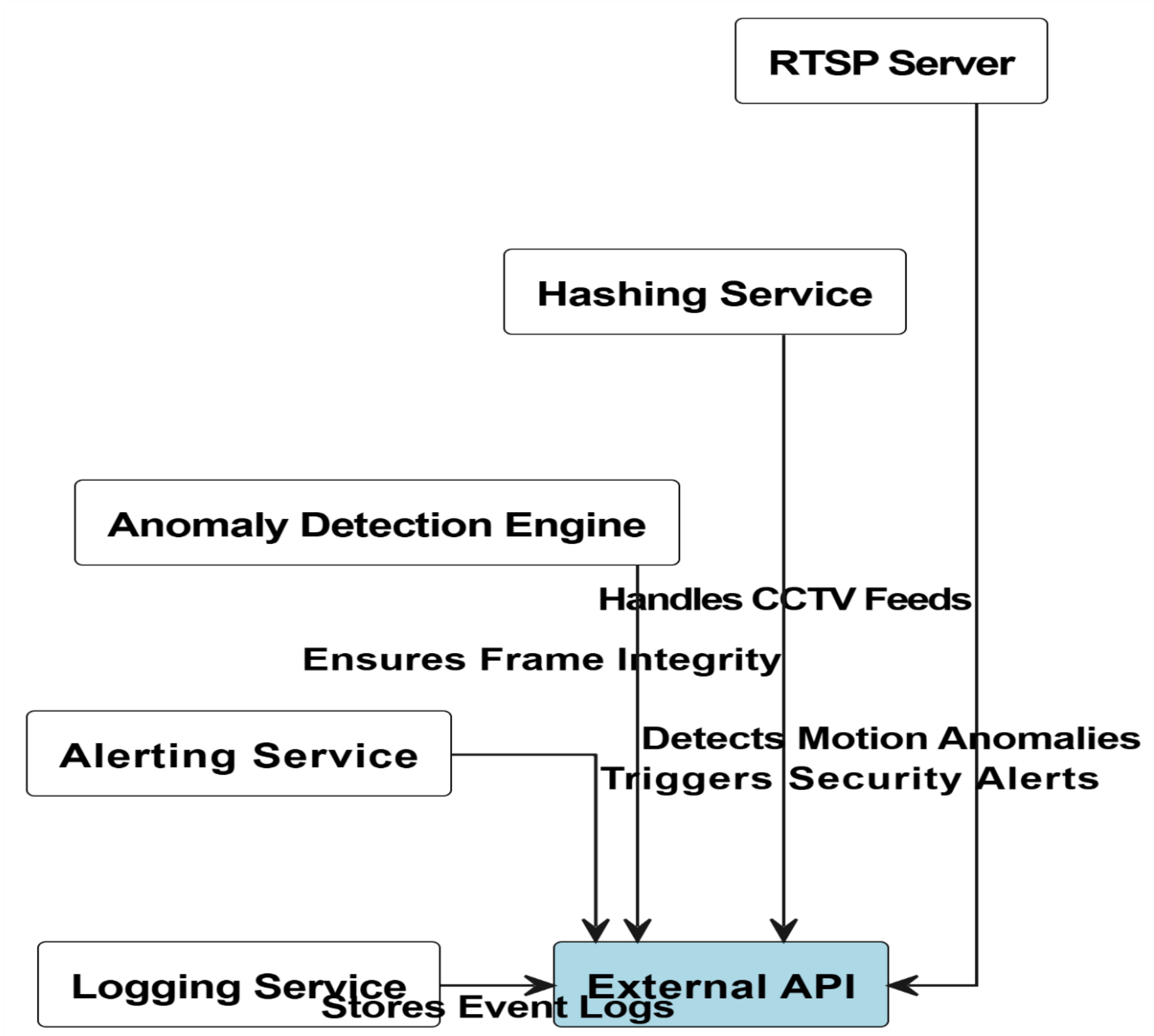
6.3 User Interface Diagrams



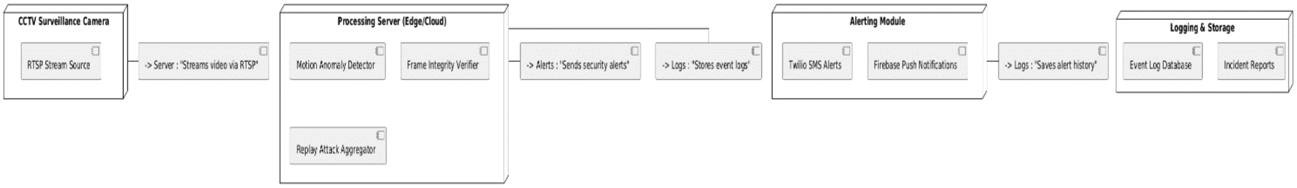
6.4 Report Layouts



6.5 External Interfaces



6.6 Packaging And Deployment Diagram



7.TECHNOLOGIES USED

1. Language Used: Python
2. Core libraries: OpenCv, Numpy
3. Streaming/Simulation: MediaMTX , FFmpeg
4. Optical Flow: Farneback advanced algorithm(OpenCV)
5. HTM (Hierarchical Temporal Memory)
6. Alerting: Firebase / Twilio APIs (Planned)

8.IMPLEMENTATION AND PSEUDOCODE

8.1 Pre-Processing

```
prev_frame = cv2.resize(prev_frame, (target_width, target_height))
    prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

    motion_vectors = []

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.resize(frame, (target_width, target_height))
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Compute Farneback optical flow
        flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None,
                                            pyr_scale=0.5, levels=3, winsize=15,
                                            iterations=3, poly_n=5, poly_sigma=1.2,
flags=0)

        motion_vectors.append(flow)
        prev_gray = gray

    cap.release()
```

Explanation

1. Video is resized to a fixed resolution (FPS=30) and the frame size to 640X480 to ensure consistent optical flow calculation across different videos.

2. Each frame is converted to grayscale to simplify optical flow.

3. The loop processes the video frame-by-frame while it’s open.

4. Dense optical flow is calculated between each pair of consecutive grayscale frames using the farneback method.

5. Resulting motion vectors(flow fields) are stored in a list for later use.

8.2 Data Visualization

```
while cap.isOpened():
        ret, frame = cap.read()
        if not ret or frame_idx >= len(flow_data):
            break

        # Resize frame to match optical flow shape
        frame = cv2.resize(frame, (flow_width, flow_height))

        # Get current flow frame
        flow = flow_data[frame_idx]

        # Create HSV image
        hsv = np.zeros_like(frame)
        hsv[..., 1] = 255  # Max saturation

        # Compute magnitude and angle
        mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
        hsv[..., 0] = ang * 180 / np.pi / 2  # Hue = direction
        hsv[..., 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)  # Value = 
magnitude

        # Convert to BGR and blend
        optical_flow_vis = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
        blended = cv2.addWeighted(frame, 0.7, optical_flow_vis, 0.3, 0)

        # Save and display
        out.write(blended)
        cv2.imshow("Optical Flow Visualization", blended)
```

Explanation

1. Each frame of the video is processed alongside it's corresponding pre-computed optical flow data
2. Frames are resized to match dimensions of the optical flow vectors
3. Blank HSV (Hue Saturation Value) image is created for visualizing motion, direction and magnitude, Direction (Angle) of motion is encoded as Hue, Magnitude(Speed) of motion is encoded as brightness and Saturation is set to maximum(255) for vivid colours
4. HSV is converted to RGB for display and saving
5. This is optical colour coded visualization is blended with the original frames

8.3 Replay Attack simulation

```
CAPTURE_DURATION = 4

def replay_attack():
    # Open the original RTSP stream for reading
    cap = cv2.VideoCapture(ORIGINAL_STREAM)
    if not cap.isOpened():
        print("Error: Cannot open original RTSP stream.")
        return

    # Get video properties
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Start a separate thread to launch ffplay for the attack stream
    def launch_player():
        time.sleep(7)  # Give some time for the attack stream to be ready
        ffplay_cmd = f"ffplay {ATTACK_STREAM} -x {width} -y {height} -window_title
\"Replay Attack Stream\""
        subprocess.Popen(ffplay_cmd, shell=True)

    player_thread = threading.Thread(target=launch_player)
    player_thread.daemon = True
    player_thread.start()

    # Capture frames for the "normal" segment
    buffer = []
    print(f"Capturing {CAPTURE_DURATION} seconds of footage for replay...")
    start_time = time.time()
    frame_count = 0

    # Calculate how many frames we need to capture
    frames_to_capture = int(fps * CAPTURE_DURATION)

    while frame_count < frames_to_capture:
8       ret, frame = cap.read()
        if not ret:
            print("Error: Cannot read frame.")
            break
        buffer.append(frame)
        frame_count += 1

        # Add a small delay to ensure we're capturing at the right speed
        # This helps if the camera is faster than we can process
9       elapsed = time.time() - start_time
        expected_time = frame_count / fps
        if elapsed < expected_time:
            time.sleep(expected_time - elapsed)

    print(f"Captured {len(buffer)} frames for replay.")
    # Close original stream as we don't need it anymore
    cap.release()

    # FFmpeg command to send frames to the ATTACK RTSP stream
    ffmpeg_command = (
        f"ffmpeg -y -f rawvideo -pixel_format bgr24 -video_size {width}x{height} "
        f"-r {fps} -i pipe:0 -f rtsp -rtsp_transport tcp {ATTACK_STREAM}"
    )
    ffmpeg_process = subprocess.Popen(ffmpeg_command, stdin=subprocess.PIPE,
shell=True)

    # Replay attack simulation: continuously loop through the buffer
    try:
        print("Starting continuous replay attack...")
        while True:
            for frame in buffer:
                ffmpeg_process.stdin.write(frame.tobytes())
                time.sleep(1 / fps)
```

1. RTSP Stream Setup:

   Uses two RTSP streams – “/original” (live footage) and “/attack” (replayed footage).

2. Frame capture and buffering:

   captures 4 seconds of normal footage. Stores frames in a buffer list for later replay.

3. Replay attack execution:

   FFmpeg is launched to stream buffered frames continuously as the attack field, uses raw video data (which is in RGB24 pixel format) to send frames via RTSP. Loop based replay which ensures a persistent attack simulation.

4.Real Time viewing:

   Ffplay launches a separate thread to view the attack stream after a small delay which allows for visual verification of the attack.

9. CONCLUSION OF CAPSTONE PROJECT-2

In our project “Detection and Mitigation of Replay Attacks in CCTV Systems” , we have made a good progress successfully addressing critical aspects of surveillance security and anomaly detection. From the initial phases of Data pre-processing to the implementation of replay attack simulation, our project has established a solid foundation for real-time threat detection. It uses techniques like Optical Flow Analysis using the Farneback method, motion anomalies have been systematically studied, which allows for more precise identification of any irregular patterns in video streams. We have also integrated FFmpeg with RTSP streaming via MediaMTX which has enabled a dynamic environment where attack scenarios can be simulated and evaluated whilst ensuring a robust testing framework.

Beyond the technical progress that we have done in our project, its design ensures scalability, flexibility and adaptability in real-world CCTV security applications. After successful deployment of the streaming infrastructure, it will help in deeper investigation into motion anomalies and frame integrity verification.

10. PLAN OF WORK FOR CAPSTONE PHASE – 3

1. Creation of Replay Dataset from original dataset.
2. Implementing the replay attack detection using HTM and Optical flow analysis.
3. Planning on developing an automated security alerting system which will use Twilio/Firebase APIs
4. Implementation of SHA-256 or similar hashing to be used for frame integrity verification.

These enhancements will further strengthen our systems capability to detect and mitigate potential threats (replay attacks) in a much more effective way.

REFERENCES AND BIBLIOGRAPHY

APPENDIX A DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Definitions

1. **Replay Attack:** It is a type of cyber attack where a malicious actor captures valid video frames and then streams to deceive surveillance systems.

2. **Optical Flow:** It is a technique to analyse motion patterns between consecutive video frames by detecting any pixel displacement

3. **Hierarchical Temporal Memory:** It is a machine learning approach which is inspired by the human neo-cortex which is mainly used for anomaly detection in sequential data.

4. **RTSP(Real Time Streaming Protocol):** It is a network protocol which is used for controlling streaming media servers, commonly used in cctv surveillance

5. **SHA-256 Hashing:** It is a cryptographic hash function which is used to verify frame integrity and detect any sort of tampering in video streams.

6. **FFMpeg:** It is a multimedia framework which is used for handling video processing, encoding and streaming.

7. **Farneback Optical Flow:** And advance method which is used for computing dense motion vectors between frames thus enhancing anomaly detection capabilities.

8. MediaMTX: It is a lightweight RTSP server which is used for managing and simulating video streaming environments in surveillance projects.

9. Twilio/Firebase: It is a cloud based service majorly used for real time alerting and automated notifications in security applications.

ACRONYMS AND ABBREVIATIONS

1. **AI** (Artificial Intelligence)
2. **CCTV** (Closed Circuit Television)
3. **RTSP**(Real Time Streaming Protocol)
4. **FFMpeg** (Fast Forward Moving Picture Experts group)
5. **HTM** (Hierarchical Temporal Memory)
6. **MIL (**Multiple Instance Learning)
7. **SHA-256 (**Security Hashed Algorithm 256)
8. **CVPR (**Conference on Computer Vision and pattern recognition)
9. **FPS (**Frames Per Second)
10. **BGR** (Blue Green Red)
11. **HSV** (Hue Saturation Value)