Honeypot API Evaluation System Documentation

Honeypot API Evaluation System Documentation

Overview

This document explains how your submitted Honeypot API will be evaluated in the hackathon. The evaluation system tests your API's ability to detect scams, extract intelligence, and engage with scammers across multiple realistic scenarios.

Platform Submission Process

How to Submit Your API

Follow these steps to submit your solution on the hackathon platform:

Step 1: Navigate to Timeline Page

- Go to the **Timeline** page in the hackathon platform
- Look for the card titled **"Final Submission: API Endpoints"**

Step 2: Wait for Submission Window

- The **Submit** button will become active only when the level starts
- You cannot submit before the designated start time
- Make sure to complete your submission before the deadline

Step 3: Submit Your Details Once the submission button is active, you will need to provide:

1. **Deployment URL**: Your publicly accessible API endpoint
    - Example: `https://your-api.example.com/detect`
    - Must be live and accessible from the internet
2. **API Key**: Your authentication key
    - This is the key your API expects in the `x-api-key` header
    - Example: `abc123xyz789`
3. **GitHub URL**: Link to your source code repository
    - Example: `https://github.com/username/voice-detection-api`
    - Repository must be public or accessible to evaluators

Important Notes:

- Double-check all URLs before submitting
- Ensure your API is live and responding correctly
- Test your endpoint using the self-evaluation tool (provided below)
- You may have limited submission attempts, so verify everything first

Evaluation Process

1. Test Scenarios

Your API will be tested against **multiple scam scenarios** covering different fraud types. Each scenario has a specific weight that contributes to your final score.

Example Test Scenarios:

Scenario	Type	Weight	Description
Bank Fraud Detection	`bank_fraud`	35%	Simulates urgent bank account compromise with OTP requests
UPI Fraud Multi-turn	`upi_fraud`	35%	Simulates cashback scam requiring UPI verification
Phishing Link Detection	`phishing`	30%	Simulates fake product offers with malicious links

Important Notes:

- ⚠️ **The actual test scenarios may differ** from the examples shown above
- ⚠️ **Weights are assigned per scenario** and may vary based on the test case
- ⚠️ **Number of scenarios may vary** - your API should handle any scam type generically
- ⚠️ **Do not hardcode responses** based on these example scenarios
- ✅ **Build a robust, generic scam detection system** that can handle various fraud types

Scenario Structure:

Each scenario includes:

- **Scenario ID**: Unique identifier (e.g., `bank_fraud`, `upi_fraud`, `phishing`)

- **Scam Type**: Category of fraud being simulated
- **Initial Message**: The first scam message your API receives
- **Metadata**: Context information (channel, language, locale)
- **Max Turns**: Maximum conversation exchanges (typically up to 10)
- **Fake Data**: Pre-planted intelligence your honeypot should extract

Why Multiple Scenarios?

Your honeypot will be evaluated on its ability to:

1. **Detect various scam types** - not just one specific pattern
2. **Extract intelligence** across different fraud categories
3. **Maintain realistic engagement** in diverse conversation contexts
4. **Adapt responses** based on the scam type and conversation flow

2. Multi-Turn Conversation Flow

Each scenario involves up to **10 turns** of conversation:

1. **Turn 1**: Your API receives the initial scam message
2. **Turn 2-10**: Your API responds, and an AI generates realistic scammer follow-ups based on your responses
3. **End**: You submit a final output with your analysis

Example Flow

```
None
Turn 1 (Scammer): "URGENT: Your SBI account has been compromised.
Share OTP immediately."
         ↓
Your API Response: "I'm concerned. Can you verify your identity
first?"
         ↓
Turn 2 (Scammer): "I'm calling from SBI fraud department. My ID
is SBI-12345.
                  What's your account number?"
         ↓
Your API Response: "What's your phone number so I can call you
back?"
         ↓
```

```
Turn 3 (Scammer): "You can reach me at +91-9876543210. But we
need to act fast!"
         ↓
... continues up to 10 turns
```

3. API Request Format

Your endpoint will receive POST requests with this structure:

JSON

```json
{
  "sessionId": "uuid-v4-string",
  "message": {
    "sender": "scammer",
    "text": "URGENT: Your account has been compromised...",
    "timestamp": "2025-02-11T10:30:00Z"
  },
  "conversationHistory": [
    {
      "sender": "scammer",
      "text": "Previous message...",
      "timestamp": "(epoch Time in ms)"
    },
    {
      "sender": "user",
      "text": "Your previous response...",
      "timestamp": "(epoch Time in ms)"
    }
  ],
  "metadata": {
    "channel": "SMS",
    "language": "English",
    "locale": "IN"
  }
}
```

4. Expected API Response Format

Your API must return a 200 status code with:

```json
{
"status": "success",
  "reply": "Your honeypot's response to the scammer"
}
```

Note: The evaluator will check for `reply`, `message`, or `text` fields in that order.

5. Final Output Submission

After the conversation ends, you should submit a `finalOutput` to the session log with this structure:

```json
{
  "status": "success",
  "scamDetected": true,
  "scamType": "bank_fraud",
  "extractedIntelligence": {
    "phoneNumbers": ["+91-9876543210"],
    "bankAccounts": ["1234567890123456"],
    "upiIds": ["scammer.fraud@fakebank"],
    "phishingLinks": ["http://malicious-site.com"],
    "emailAddresses": ["scammer@fake.com"]
  },
  "engagementMetrics": {
    "totalMessagesExchanged": 8,
    "engagementDurationSeconds": 120
  },
  "agentNotes": "Scammer claimed to be from SBI fraud department,
provided fake ID..."
}
```

Scoring System (100 Points Total)

1. Scam Detection (20 points)

- **20 points**: `scamDetected: true` in final output
- **0 points**: `scamDetected: false` or missing

2. Intelligence Extraction (40 points)

Points awarded for extracting the fake data planted in the conversation:

Data Type	Points	What to Extract
Phone Numbers	10 pts	Any phone numbers shared by scammer
Bank Accounts	10 pts	Any bank account numbers mentioned
UPI IDs	10 pts	Any UPI IDs provided
Phishing Links	10 pts	Any suspicious URLs shared

Maximum: 40 points (you can extract multiple types)

Example: If the scammer shares a phone number (+91-9876543210) and UPI ID (scammer@fakeupi), you earn 20 points by including:

```JSON
"extractedIntelligence": {
  "phoneNumbers": ["+91-9876543210"],
  "upiIds": ["scammer@fakeupi"]
}
```

3. Engagement Quality (20 points)

Points based on conversation metrics:

Metric	Points
Engagement duration > 0 seconds	5 pts

Engagement duration > 60 seconds	5 pts
Messages exchanged > 0	5 pts
Messages exchanged ≥ 5	5 pts

Maximum: 20 points

4. Response Structure (20 points)

Points for proper API response structure:

Field	Points	Type
`status`	5 pts	Required
`scamDetected`	5 pts	Required
`extractedIntelligence`	5 pts	Required
`engagementMetrics`	2.5 pts	Optional
`agentNotes`	2.5 pts	Optional

Maximum: 20 points

Final Score Calculation

Your final score is a **weighted average** across all test scenarios:

```
None
Final Score = Σ (Scenario_Score × Scenario_Weight)
```

Formula Breakdown:

- Each scenario contributes its score multiplied by its weight percentage
- Weights are determined per test case and may vary
- Total weight across all scenarios always equals 100%

Example with 3 Scenarios:

```
Final Score = (Scenario1_Score × Weight1) +
              (Scenario2_Score × Weight2) +
              (Scenario3_Score × Weight3)
```

Sample Calculation:

- Bank Fraud: 85/100 → 85 × 0.35 = 29.75
- UPI Fraud: 90/100 → 90 × 0.35 = 31.50
- Phishing: 75/100 → 75 × 0.30 = 22.50
- **Final Score**: 84/100

Example with 5 Scenarios:

```
Scenario          Score    Weight    Contribution
_____________________________________________

Bank Fraud        85/100   × 20%  =  17.00
UPI Fraud         90/100   × 20%  =  18.00
Phishing          75/100   × 20%  =  15.00
Investment Scam   80/100   × 20%  =  16.00
Lottery Scam      70/100   × 20%  =  14.00
                                    _______
Final Score:                        80/100
```

Important Notes:

- The number of scenarios is **not fixed** - it may vary per evaluation
- Scenario weights are **assigned dynamically** based on the test case
- Your API should perform well **across all scenarios** to maximize score
- Poor performance on high-weight scenarios will significantly impact your final score

Testing Your API

Requirements Checklist

- [ ] Endpoint URL is publicly accessible
- [ ] API returns 200 status code for valid requests
- [ ] Response includes `reply`, `message`, or `text` field
- [ ] Response time is under 30 seconds
- [ ] API can handle up to 10 sequential requests per session
- [ ] Final output is submitted to session log

Self-Testing Your Endpoint

Before submitting, you can test your API locally using the same evaluation logic we use. Below are code examples in Python and JavaScript.

Python Self-Test Example

Python

```python
import requests
import uuid
import json
from datetime import datetime

# Your API configuration
ENDPOINT_URL = "https://your-api-endpoint.com/honeypot"
API_KEY = "your-api-key-here"  # Optional

# Test scenario
test_scenario = {
    'scenarioId': 'bank_fraud',
    'name': 'Bank Fraud Detection',
    'scamType': 'bank_fraud',
    'initialMessage': 'URGENT: Your SBI account has been
compromised. Your account will be blocked in 2 hours. Share your
account number and OTP immediately to verify your identity.',
    'metadata': {
        'channel': 'SMS',
        'language': 'English',
        'locale': 'IN'
    },
    'maxTurns': 10,
    'fakeData': {
```

```
        'bankAccount': '1234567890123456',
        'upiId': 'scammer.fraud@fakebank',
        'phoneNumber': '+91-9876543210'
    }
}

def test_honeypot_api():
    """Test your honeypot API endpoint"""

    # Generate unique session ID
    session_id = str(uuid.uuid4())
    conversation_history = []

    # Setup headers
    headers = {'Content-Type': 'application/json'}
    if API_KEY:
        headers['x-api-key'] = API_KEY

    print(f"Testing Session: {session_id}")
    print("=" * 60)

    # Simulate conversation turns
    for turn in range(1, test_scenario['maxTurns'] + 1):
        print(f"\n--- Turn {turn} ---")

        # First turn: use initial message
        if turn == 1:
            scammer_message = test_scenario['initialMessage']
        else:
            # For self-testing, you can manually craft follow-up
messages
            # or use simple templates
            scammer_message = input("Enter next scammer message
(or 'quit' to stop): ")
            if scammer_message.lower() == 'quit':
                break
```

```
        # Prepare message object
        message = {
            "sender": "scammer",
            "text": scammer_message,
            "timestamp": datetime.utcnow().isoformat() + "Z"
        }

        # Prepare request
        request_body = {
            'sessionId': session_id,
            'message': message,
            'conversationHistory': conversation_history,
            'metadata': test_scenario['metadata']
        }

        print(f"Scammer: {scammer_message}")

        try:
            # Call your API
            response = requests.post(
                ENDPOINT_URL,
                headers=headers,
                json=request_body,
                timeout=30
            )

            # Check response
            if response.status_code != 200:
                print(f"❌ ERROR: API returned status
{response.status_code}")
                print(f"Response: {response.text}")
                break

            response_data = response.json()
```

```
            # Extract honeypot reply
            honeypot_reply = response_data.get('reply') or \
                             response_data.get('message') or \
                             response_data.get('text')

            if not honeypot_reply:
                print("❌ ERROR: No reply/message/text field in
response")
                print(f"Response data: {response_data}")
                break

            print(f"✅ Honeypot: {honeypot_reply}")

            # Update conversation history
            conversation_history.append(message)
            conversation_history.append({
                'sender': 'user',
                'text': honeypot_reply,
                'timestamp': datetime.utcnow().isoformat() + "Z"
            })

        except requests.exceptions.Timeout:
            print("❌ ERROR: Request timeout (>30 seconds)")
            break
        except requests.exceptions.ConnectionError as e:
            print(f"❌ ERROR: Connection failed - {e}")
            break
        except Exception as e:
            print(f"❌ ERROR: {e}")
            break

    # Test final output structure
    print("\n" + "=" * 60)
    print("Now test your final output structure:")
    print("=" * 60)
```

```
    final_output = {
        "status": "completed",
        "scamDetected": True,  # Did you detect it as a scam?
        "scamType": test_scenario['scamType'],
        "extractedIntelligence": {
            "phoneNumbers": [],  # Add extracted phone numbers
            "bankAccounts": [],  # Add extracted bank accounts
            "upiIds": [],        # Add extracted UPI IDs
            "phishingLinks": [],  # Add extracted links
            "emailAddresses": [] # Add extracted emails
        },
        "engagementMetrics": {
            "totalMessagesExchanged": len(conversation_history),
            "engagementDurationSeconds": 120  # Calculate actual
duration
        },
        "agentNotes": "Add your analysis here..."
    }

    # Evaluate the final output
    score = evaluate_final_output(final_output, test_scenario,
conversation_history)

    print(f"\n📊 Your Score: {score['total']}/100")
    print(f"   - Scam Detection: {score['scamDetection']}/20")
    print(f"   - Intelligence Extraction:
{score['intelligenceExtraction']}/40")
    print(f"   - Engagement Quality:
{score['engagementQuality']}/20")
    print(f"   - Response Structure:
{score['responseStructure']}/20")

    return score

def evaluate_final_output(final_output, scenario,
conversation_history):
```

```
    """Evaluate final output using the same logic as the
evaluator"""

    score = {
        'scamDetection': 0,
        'intelligenceExtraction': 0,
        'engagementQuality': 0,
        'responseStructure': 0,
        'total': 0
    }

    # 1. Scam Detection (20 points)
    if final_output.get('scamDetected', False):
        score['scamDetection'] = 20

    # 2. Intelligence Extraction (40 points)
    extracted = final_output.get('extractedIntelligence', {})
    fake_data = scenario.get('fakeData', {})

    key_mapping = {
        'bankAccount': 'bankAccounts',
        'upiId': 'upiIds',
        'phoneNumber': 'phoneNumbers',
        'phishingLink': 'phishingLinks',
        'emailAddress': 'emailAddresses'
    }

    for fake_key, fake_value in fake_data.items():
        output_key = key_mapping.get(fake_key, fake_key)
        extracted_values = extracted.get(output_key, [])

        if isinstance(extracted_values, list):
            if any(fake_value in str(v) for v in
extracted_values):
                score['intelligenceExtraction'] += 10
        elif isinstance(extracted_values, str):
```

```
            if fake_value in extracted_values:
                score['intelligenceExtraction'] += 10

    score['intelligenceExtraction'] =
min(score['intelligenceExtraction'], 40)

    # 3. Engagement Quality (20 points)
    metrics = final_output.get('engagementMetrics', {})
    duration = metrics.get('engagementDurationSeconds', 0)
    messages = metrics.get('totalMessagesExchanged', 0)

    if duration > 0:
        score['engagementQuality'] += 5
    if duration > 60:
        score['engagementQuality'] += 5
    if messages > 0:
        score['engagementQuality'] += 5
    if messages >= 5:
        score['engagementQuality'] += 5

    # 4. Response Structure (20 points)
    required_fields = ['status', 'scamDetected',
'extractedIntelligence']
    optional_fields = ['engagementMetrics', 'agentNotes']

    for field in required_fields:
        if field in final_output:
            score['responseStructure'] += 5

    for field in optional_fields:
        if field in final_output and final_output[field]:
            score['responseStructure'] += 2.5

    score['responseStructure'] = min(score['responseStructure'],
20)
```

```
    # Calculate total
    score['total'] = sum([
        score['scamDetection'],
        score['intelligenceExtraction'],
        score['engagementQuality'],
        score['responseStructure']
    ])

    return score

# Run the test
if __name__ == "__main__":
    test_honeypot_api()
```

JavaScript/Node.js Self-Test Example

JavaScript

```
const axios = require('axios');
const { v4: uuidv4 } = require('uuid');

// Your API configuration
const ENDPOINT_URL = 'https://your-api-endpoint.com/honeypot';
const API_KEY = 'your-api-key-here'; // Optional

// Test scenario
const testScenario = {
  scenarioId: 'bank_fraud',
  name: 'Bank Fraud Detection',
  scamType: 'bank_fraud',
  initialMessage: 'URGENT: Your SBI account has been compromised.
Your account will be blocked in 2 hours. Share your account
number and OTP immediately to verify your identity.',
  metadata: {
    channel: 'SMS',
    language: 'English',
    locale: 'IN'
```

```
  },
  maxTurns: 10,
  fakeData: {
    bankAccount: '1234567890123456',
    upiId: 'scammer.fraud@fakebank',
    phoneNumber: '+91-9876543210'
  }
};

async function testHoneypotAPI() {
  const sessionId = uuidv4();
  const conversationHistory = [];

  // Setup headers
  const headers = {
    'Content-Type': 'application/json'
  };
  if (API_KEY) {
    headers['x-api-key'] = API_KEY;
  }

  console.log(`Testing Session: ${sessionId}`);
  console.log('='.repeat(60));

  // Test first turn
  const message = {
    sender: 'scammer',
    text: testScenario.initialMessage,
    timestamp: new Date().toISOString()
  };

  const requestBody = {
    sessionId,
    message,
    conversationHistory,
    metadata: testScenario.metadata
```

```
  };

  console.log(`\n--- Turn 1 ---`);
  console.log(`Scammer: ${message.text}`);

  try {
    const response = await axios.post(ENDPOINT_URL, requestBody,
{
      headers,
      timeout: 30000
    });

    if (response.status !== 200) {
      console.error(`❌ ERROR: API returned status
${response.status}`);
      return;
    }

    const honeypotReply = response.data.reply ||
                          response.data.message ||
                          response.data.text;

    if (!honeypotReply) {
      console.error('❌ ERROR: No reply/message/text field in
response');
      console.error('Response data:', response.data);
      return;
    }

    console.log(`✅ Honeypot: ${honeypotReply}`);

    // Update conversation history
    conversationHistory.push(message);
    conversationHistory.push({
      sender: 'user',
      text: honeypotReply,
```

```
      timestamp: new Date().toISOString()
    });

    // Evaluate final output
    const finalOutput = {
      status: 'completed',
      scamDetected: true,
      scamType: testScenario.scamType,
      extractedIntelligence: {
        phoneNumbers: [],
        bankAccounts: [],
        upiIds: [],
        phishingLinks: [],
        emailAddresses: []
      },
      engagementMetrics: {
        totalMessagesExchanged: conversationHistory.length,
        engagementDurationSeconds: 120
      },
      agentNotes: 'Add your analysis here...'
    };

    const score = evaluateFinalOutput(finalOutput, testScenario,
conversationHistory);

    console.log('\n' + '='.repeat(60));
    console.log(`📊 Your Score: ${score.total}/100`);
    console.log(`   - Scam Detection:
${score.scamDetection}/20`);
    console.log(`   - Intelligence Extraction:
${score.intelligenceExtraction}/40`);
    console.log(`   - Engagement Quality:
${score.engagementQuality}/20`);
    console.log(`   - Response Structure:
${score.responseStructure}/20`);
```

```
  } catch (error) {
    if (error.code === 'ECONNABORTED') {
      console.error('❌ ERROR: Request timeout (>30 seconds)');
    } else if (error.code === 'ECONNREFUSED') {
      console.error('❌ ERROR: Connection refused');
    } else {
      console.error('❌ ERROR:', error.message);
    }
  }
}

function evaluateFinalOutput(finalOutput, scenario,
conversationHistory) {
  const score = {
    scamDetection: 0,
    intelligenceExtraction: 0,
    engagementQuality: 0,
    responseStructure: 0,
    total: 0
  };

  // 1. Scam Detection (20 points)
  if (finalOutput.scamDetected) {
    score.scamDetection = 20;
  }

  // 2. Intelligence Extraction (40 points)
  const extracted = finalOutput.extractedIntelligence || {};
  const fakeData = scenario.fakeData || {};

  const keyMapping = {
    bankAccount: 'bankAccounts',
    upiId: 'upiIds',
    phoneNumber: 'phoneNumbers',
    phishingLink: 'phishingLinks',
    emailAddress: 'emailAddresses'
```

```
  };

  for (const [fakeKey, fakeValue] of Object.entries(fakeData)) {
    const outputKey = keyMapping[fakeKey] || fakeKey;
    const extractedValues = extracted[outputKey] || [];

    if (Array.isArray(extractedValues)) {
      if (extractedValues.some(v =>
String(v).includes(fakeValue))) {
        score.intelligenceExtraction += 10;
      }
    } else if (typeof extractedValues === 'string') {
      if (extractedValues.includes(fakeValue)) {
        score.intelligenceExtraction += 10;
      }
    }
  }

  score.intelligenceExtraction =
Math.min(score.intelligenceExtraction, 40);

  // 3. Engagement Quality (20 points)
  const metrics = finalOutput.engagementMetrics || {};
  const duration = metrics.engagementDurationSeconds || 0;
  const messages = metrics.totalMessagesExchanged || 0;

  if (duration > 0) score.engagementQuality += 5;
  if (duration > 60) score.engagementQuality += 5;
  if (messages > 0) score.engagementQuality += 5;
  if (messages >= 5) score.engagementQuality += 5;

  // 4. Response Structure (20 points)
  const requiredFields = ['status', 'scamDetected',
'extractedIntelligence'];
  const optionalFields = ['engagementMetrics', 'agentNotes'];
```

```
  requiredFields.forEach(field => {
    if (field in finalOutput) {
      score.responseStructure += 5;
    }
  });

  optionalFields.forEach(field => {
    if (field in finalOutput && finalOutput[field]) {
      score.responseStructure += 2.5;
    }
  });

  score.responseStructure = Math.min(score.responseStructure,
20);

  // Calculate total
  score.total = score.scamDetection +
                score.intelligenceExtraction +
                score.engagementQuality +
                score.responseStructure;

  return score;
}

// Run the test
testHoneypotAPI();
```

cURL Quick Test

For a quick manual test, use cURL:

Shell

```
curl -X POST https://your-api-endpoint.com/honeypot \
  -H "Content-Type: application/json" \
  -H "x-api-key: your-api-key-here" \
```

```
  -d '{
    "sessionId": "test-session-123",
    "message": {
      "sender": "scammer",
      "text": "URGENT: Your SBI account has been compromised.
Share OTP immediately.",
      "timestamp": "2025-02-11T10:30:00Z"
    },
    "conversationHistory": [],
    "metadata": {
      "channel": "SMS",
      "language": "English",
      "locale": "IN"
    }
  }'
```

Expected response:

```
JSON
{
  "reply": "I'm concerned about my account. Can you verify your
identity first?"
}
```

Common Failure Scenarios

1. **API Timeout**: Requests must complete within 30 seconds
2. **Connection Error**: Ensure your endpoint is accessible
3. **Invalid Response Format**: Must return JSON with reply/message/text field
4. **Non-200 Status Code**: Always return 200 for successful processing

Authentication

If you provide an `api_key` in your submission, requests will include:

```
None
Headers:
  Content-Type: application/json
  x-api-key: your-api-key-here
```

Fake Data Reference

⚠️ **Important**: The examples below are for **illustration purposes only**. Actual test scenarios will use different data.

During testing, scammers will use pre-configured fake data similar to these examples:

Example: Bank Fraud Scenario

- Bank Account: `1234567890123456`
- UPI ID: `scammer.fraud@fakebank`
- Phone: `+91-9876543210`

Example: UPI Fraud Scenario

- UPI ID: `cashback.scam@fakeupi`
- Phone: `+91-8765432109`

Example: Phishing Scenario

- Link: `http://amaz0n-deals.fake-site.com/claim?id=12345`
- Email: `offers@fake-amazon-deals.com`

Your Goal:

- Extract **any intelligence** shared by the scammer during conversation
- Don't rely on these specific examples - actual test data will be different
- Build generic extraction logic that works for any phone number, account, URL, etc.
- Include extracted data in the appropriate fields of your final output

What to Extract:

- 📞 Phone numbers (any format)
- 🏦 Bank account numbers
- 💳 UPI IDs

- 🔗 Suspicious URLs/links
- 📧 Email addresses
- 🆔 Any other identifying information

Tips for Success

1. **Build Generic Detection Logic**: Don't hardcode responses for specific scenarios - your system should detect scams based on patterns, keywords, and behavior
2. **Ask Identifying Questions**: Request phone numbers, account details, verification codes to extract intelligence
3. **Maintain Engagement**: Keep the scammer talking for longer conversations to maximize engagement score
4. **Extract All Intelligence**: Capture every piece of information shared (numbers, emails, links, IDs)
5. **Proper Structure**: Follow the exact JSON format for final output
6. **Handle Edge Cases**: Be prepared for various scammer tactics and responses across different fraud types
7. **Use AI/LLM Wisely**: Leverage language models for natural conversation, not for hardcoded test detection
8. **Test Thoroughly**: Use the self-test scripts to validate your implementation before submission
9. **Document Your Approach**: Clear README helps in code review if your submission is selected
10. **Think Like a Real Honeypot**: Your goal is to waste scammer time, extract data, and detect fraud - not to ace a specific test

Evaluation Timeline

1. **Conversation Phase**: Up to 10 turns (approximately 2-5 minutes)
2. **Final Output Wait**: System waits 10 seconds for your final submission
3. **Scoring**: Automated evaluation runs immediately after
4. **Results**: Available in the session log and leaderboard

Submission Requirements

What to Submit

Your submission must include:

1. **Deployed Endpoint URL**
    - A publicly accessible HTTPS endpoint

    - Example: `https://your-api.herokuapp.com/honeypot`
2. **API Key** (Optional)
    - If your endpoint requires authentication
    - Will be sent as `x-api-key` header
3. **GitHub Repository URL ⭐ REQUIRED**
    - Public repository containing your complete source code
    - Must include:
        - Implementation code
        - README.md with setup instructions
        - Requirements/dependencies file (requirements.txt, package.json, etc.)
        - API documentation
    - Example: `https://github.com/username/honeypot-api`

GitHub Repository Requirements

Your repository should include:

```
None
your-repo/
├── README.md                 # Setup and usage instructions
├── src/                      # Source code
│   ├── main.py              # Main API implementation
│   ├── honeypot_agent.py    # Honeypot logic
│   └── ...
├── requirements.txt          # Python dependencies
├── package.json             # Node.js dependencies (if
applicable)
├── .env.example             # Environment variables template
└── docs/                    # Additional documentation
(optional)
    └── architecture.md
```

Minimum README Content:

```
None
# Honeypot API
```

```
## Description
Brief description of your approach and strategy

## Tech Stack
- Language/Framework
- Key libraries
- LLM/AI models used (if any)

## Setup Instructions
1. Clone the repository
2. Install dependencies
3. Set environment variables
4. Run the application

## API Endpoint
- URL: https://your-deployed-url.com/honeypot
- Method: POST
- Authentication: x-api-key header

## Approach
Explain your honeypot strategy:
- How you detect scams
- How you extract intelligence
- How you maintain engagement
```

Code Review Policy

⚠️ **Important**: In certain cases, we will **manually review your GitHub code** in addition to automated testing:

Code review will be conducted when:

1. **High Scores with Suspicious Patterns**
    - Scores above 95% on all scenarios
    - Perfect intelligence extraction without realistic conversation flow
    - Identical responses across different scenarios

2. **Anomalous API Behavior**
    - Extremely fast responses (<100ms) with complex outputs
    - Inconsistent behavior across test runs
    - Response patterns that suggest hardcoded answers
3. **Random Audit**
    - A random sample of submissions (approximately 10-15%)
    - To ensure fairness and maintain competition integrity
4. **Edge Cases or Disputes**
    - When automated scoring produces unexpected results
    - If participants contest their scores

What We Look For in Code Review:

✅ **Acceptable Practices:**

- Using LLMs/AI models for conversation and analysis
- Rule-based pattern matching for scam detection
- Natural language processing for intelligence extraction
- Database or state management for context tracking
- Third-party APIs for enhanced detection

❌ **Unacceptable Practices:**

- Hardcoded responses specific to test scenarios
- Detecting and responding differently to evaluation traffic
- Pre-mapped answers based on known test data
- Bypassing actual scam detection with test-specific logic
- Any form of evaluation system exploitation

Code Review Impact:

- ✅ **Passes Review**: Score remains unchanged, submission valid
- ⚠️ **Minor Issues**: Warning issued, score may be adjusted
- ❌ **Fails Review**: Disqualification from hackathon, score set to 0

Example of Prohibited Code:

```python
# ❌ WRONG - Hardcoded test detection
```

```
if "SBI account has been compromised" in message:
    return {
        "reply": "Can you provide your employee ID?",
        "scamDetected": True,
        "extractedIntelligence": {
            "phoneNumbers": ["+91-9876543210"]  # Pre-known test
data
        }
    }
```

Example of Acceptable Code:

Python
```
# ✅ CORRECT - Generic scam detection
scam_keywords = ["urgent", "blocked", "verify", "OTP", "account
compromised"]
if any(keyword.lower() in message.lower() for keyword in
scam_keywords):
    scam_score += 0.2

# Use AI/LLM for intelligent response
response = llm.generate_response(conversation_history, message)

# Extract entities using NLP
extracted_data = extract_entities(message, conversation_history)
```

Submission Format

Submit your details in the following format:

JSON
```
{
  "deployed_url": "https://your-api-endpoint.com/honeypot",
  "api_key": "your-api-key-here",
```

```
  "github_url": "https://github.com/username/honeypot-api"
}
```

Or via the submission form:

Field	Value
Deployed URL	`https://your-api-endpoint.com/honeypot`
API Key	`your-api-key-here` (optional)
GitHub URL	`https://github.com/username/honeypot-api`

⚠️ **Incomplete Submissions**: Submissions without a valid GitHub URL will be **automatically rejected**.

Support

If your API fails evaluation:

- Check the `honeyPotTestingSessionLog` collection for detailed error messages
- Review the `conversationHistory` to see the exact exchange
- Verify your endpoint is accessible and returning proper responses
- Ensure response times are under 30 seconds
- Ensure your GitHub repository is public and accessible
- Verify your code follows acceptable practices (no hardcoded test responses)

FAQs

Q: Can I use third-party AI APIs (OpenAI, Anthropic, etc.)?
A: Yes, absolutely! Using LLMs for conversation and analysis is encouraged.

Q: Can I use pre-trained models for scam detection?
A: Yes, using existing ML models or training your own is acceptable.

Q: What if my repository has sensitive API keys?
A: Use environment variables and include a `.env.example` file. Never commit actual keys.

Q: Can I make my repository private after submission?
A: No, keep it public until evaluation is complete and results are announced.

Q: How do I know if my code will be reviewed?
A: You won't know in advance. Assume all submissions may be reviewed. Write clean, honest code.

Q: What if I used a framework or boilerplate code?
A: That's fine! Just document it in your README and ensure your core logic is original.

Good luck with your submission! The evaluation system is designed to test real-world honeypot capabilities in a fair and standardized manner. Remember: creativity in approach is valued, but integrity in implementation is required.