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:

- **Deployment URL:** Your publicly accessible API endpoint
    - Example: https://your-api.example.com/detect
    - Must be live and accessible from the internet
- **API Key:** Your authentication key
    - This is the key your API expects in the x-api-key header
    - Example: abc123xyz789
- **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:

- **Session ID:** Unique identifier
- **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:

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

---

2. Multi-Turn Conversation Flow

Each scenario involves up to 10 turns of conversation:

- Turn 1: Your API receives the initial scam message
- Turns 2–10: Your API responds, and an AI generates realistic scammer follow-ups based on your responses
- 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
{
  "sessionId": "abc123-session-id",
  "scamDetected": true,
  "totalMessagesExchanged": 18,
  "engagementDurationSeconds":120,
  "extractedIntelligence": {
    "phoneNumbers": ["+91-9876543210"],
    "bankAccounts": ["1234567890123456"],
    "upiIds": ["scammer.fraud@fakebank"],
    "phishingLinks": ["http://malicious-site.com"],
    "emailAddresses": ["scammer@fake.com"]
  },
```

```
  "agentNotes": "Scammer claimed to be from SBI fraud department,
provided fake ID..."
}
```

Scoring System (100 Points Total)

1. Scam Detection — 20 Points

Condition	Points
scamDetected: true in final output	20 points
scamDetected: false or missing	0 points

---

2. Extracted Intelligence — 30 Points

Points awarded for extracting the fake data planted in the conversation. Points per item are calculated dynamically:

Points per item = 30 ÷ Total fake data fields in scenario

Data Type	What to Extract
Phone Numbers	Any phone numbers shared by scammer
Bank Accounts	Any bank account numbers mentioned
UPI IDs	Any UPI IDs provided
Phishing Links	Any suspicious URLs shared
Email Addresses	Any email addresses shared
Case IDs	Any case/reference IDs mentioned
Policy Numbers	Any policy numbers shared
Order Numbers	Any order IDs mentioned

Example: If the scenario has 3 fake data fields, each correctly extracted field earns 10 points (30 ÷ 3). Extract all 3 → full 30 points.

```json
"extractedIntelligence": {
  "phoneNumbers": ["+91-9876543210"],
  "upiIds": ["scammer@fakeupi"],
  "phishingLinks": ["http://fake-bank-kyc.com"]
}
```

---

3. Conversation Quality — 30 Points

Sub-Category	Max Points	How to Score
Turn Count	8 pts	≥8 turns = 8pts, ≥6 = 6pts, ≥4 = 3pts
Questions Asked	4 pts	≥5 questions = 4pts, ≥3 = 2pts, ≥1 = 1pt
Relevant Questions	3 pts	≥3 investigative = 3pts, ≥2 = 2pts, ≥1 = 1pt
Red Flag Identification	8 pts	≥5 flags = 8pts, ≥3 = 5pts, ≥1 = 2pts
Information Elicitation	7 pts	Each elicitation attempt earns 1.5pts (max 7)

Tips to Maximize Conversation Quality:

- Keep the scammer engaged for as many turns as possible (aim for 8+)
- Ask investigative questions about identity, company, address, website
- Reference red flags like urgency, OTP requests, fees, or suspicious links
- Actively probe for scammer contact details and organizational info

---

4. Engagement Quality — 10 Points

Metric	Points
Engagement duration > 0 seconds	1 pt
Engagement duration > 60 seconds	2 pts
Engagement duration > 180 seconds	1 pt
Messages exchanged > 0	2 pts
Messages exchanged ≥ 5	3 pts
Messages exchanged ≥ 10	1 pt

```JSON
"engagementMetrics": {
  "engagementDurationSeconds": 240,
  "totalMessagesExchanged": 12
}
```

---

5. Response Structure — 10 Points

Field	Points	Type
sessionId	2 pts	Required
scamDetected	2 pts	Required
extractedIntelligence	2 pts	Required
totalMessagesExchanged and engagementDurationSeconds	1 pt	Optional
agentNotes	1 pt	Optional
scamType	1 pt	Optional
confidenceLevel	1 pt	Optional

⚠️ Missing required fields incur a -1 point penalty each.

Final Score Calculation

Component	Weight	Max Points
Scenario Performance	90%	90 pts
Code Quality (GitHub)	10%	10 pts

Formula:

```
None
Scenario Score = Σ (Scenario_Score × Scenario_Weight / 100)
Final Score    = (Scenario Score × 0.9) + Code Quality Score
```

Example Calculation:

Scenario	Score	Weight	Contribution
Bank Fraud	85/100	35%	29.75
UPI Fraud	90/100	35%	31.50
Phishing	75/100	30%	22.50
Weighted Scenario Score			**83.75 → 84**

```
None
Scenario Portion = 84 × 0.9  = 75.6
Code Quality     = 8 out of 10
Final Score      = 75.6 + 8  = 84 / 100
```

Important Notes:

- Scenario weights always sum to 100%
- Code quality is evaluated from your submitted GitHub URL
- If no GitHub URL is provided, code quality score defaults to 0
- The number of scenarios is not fixed and may vary per evaluation

- Scenario weights are assigned dynamically per test case
- Poor performance on high-weight scenarios will significantly impact your final score

---

Common Failure Scenarios

- **API Timeout:** Requests must complete within 30 seconds
- **Connection Error:** Ensure your endpoint is accessible
- **Invalid Response Format:** Must return JSON with reply/message/text field
- **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.

Example: Bank Fraud Scenario

```
None
Bank Account: 1234567890123456
UPI ID: scammer.fraud@fakebank
Phone: +91-9876543210
```

Example: UPI Fraud Scenario

```
None
UPI ID: cashback.scam@fakeupi
Phone: +91-8765432109
```

Example: Phishing Scenario

```
None
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

- **Build Generic Detection Logic:** Don't hardcode responses for specific scenarios — detect scams based on patterns, keywords, and behavior
- **Ask Identifying Questions:** Request phone numbers, account details, verification codes to extract intelligence
- **Maintain Engagement:** Keep the scammer talking for longer conversations to maximize engagement score
- **Extract All Intelligence:** Capture every piece of information shared (numbers, emails, links, IDs)
- **Proper Structure:** Follow the exact JSON format for final output

- **Handle Edge Cases:** Be prepared for various scammer tactics and responses across different fraud types
- **Use AI/LLM Wisely:** Leverage language models for natural conversation, not for hardcoded test detection
- **Test Thoroughly:** Use the self-test scripts to validate your implementation before submission
- **Document Your Approach:** Clear README helps in code review if your submission is selected
- **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

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

Submission Requirements

What to Submit

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, requirements/dependencies file, API documentation
- Example: https://github.com/username/honeypot-api

GitHub Repository Requirements

```
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:

High Scores with Suspicious Patterns

- Scores above 95% on all scenarios
- Perfect intelligence extraction without realistic conversation flow
- Identical responses across different scenarios

Anomalous API Behavior

- Extremely fast responses (<100ms) with complex outputs
- Inconsistent behavior across test runs
- Response patterns that suggest hardcoded answers

Random Audit

- A random sample of submissions (approximately 10–15%)
- To ensure fairness and maintain competition integrity

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

```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.