Cyber Security Landscape in Today's Technology Scenario

Bernard Nongpoh

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Workshop on Cyber Security in the Era of Emerging Technologies

IIT Guwahati · 7 May 2026

About Me

Assistant Professor, Dept. of CSE, IIT Guwahati

Previously Staff Engineer at **Qualcomm Product Security Initiative, Hyderabad**

Research: Software & system security, program analysis, fuzzing

Current focus: **AI for security** and software security analysis

Software runs the world.

Software runs the world.

Finance. Healthcare. Defence. Infrastructure. Communication.

The Scale of Modern System Software

Linux Kernel: **36.9 million** lines of C, 4,037 contributors[1]

Chromium: **36.2 million** lines of C++, 2,109 contributors[2]

[1] openhub.net/p/linux [2] openhub.net/p/chrome [3] CISA CSAC Report, Dec 2023

The Scale of Modern System Software

Linux Kernel: **36.9 million** lines of C, 4,037 contributors[1]

Chromium: **36.2 million** lines of C++, 2,109 contributors[2]

More code → more complexity → **more bugs**

And some of those bugs are security vulnerabilities.

[1] openhub.net/p/linux [2] openhub.net/p/chrome [3] CISA CSAC Report, Dec 2023

The Scale of Modern System Software

Linux Kernel: **36.9 million** lines of C, 4,037 contributors[1]

Chromium: **36.2 million** lines of C++, 2,109 contributors[2]

More code → more complexity → **more bugs**

And some of those bugs are security vulnerabilities.

70% of security issues are memory safety bugs[3]

[1] openhub.net/p/linux [2] openhub.net/p/chrome [3] CISA CSAC Report, Dec 2023

Real-World Impact

Heartbleed (2014): OpenSSL buffer over-read. Leaked private keys worldwide.[1]

[1] heartbleed.com [2] CVE-2021-44228 [3] Times of India

Real-World Impact

Heartbleed (2014): OpenSSL buffer over-read. Leaked private keys worldwide.[1]

Log4Shell (2021): Java logging library. Remote code execution on millions of servers.[2]

[1] heartbleed.com [2] CVE-2021-44228 [3] Times of India

Real-World Impact

Heartbleed (2014): OpenSSL buffer over-read. Leaked private keys worldwide.[1]

Log4Shell (2021): Java logging library. Remote code execution on millions of servers.[2]

IIT Roorkee (2025): 30,000 student and alumni records exposed for years.[3]

[1] heartbleed.com [2] CVE-2021-44228 [3] Times of India

Real-World Impact

Heartbleed (2014): OpenSSL buffer over-read. Leaked private keys worldwide.[1]

Log4Shell (2021): Java logging library. Remote code execution on millions of servers.[2]

IIT Roorkee (2025): 30,000 student and alumni records exposed for years.[3]

One insecure line can compromise **millions** of systems.

Manual code review at this scale is practically impossible.

[1] heartbleed.com [2] CVE-2021-44228 [3] Times of India

What Is Software Security?

- Ensuring correct and safe behavior under **adversarial** conditions.

What Is Software Security?

- Ensuring correct and safe behavior under **adversarial** conditions.

- Common entry point for attacks:
  - **Software vulnerabilities**: flaws in code that attackers can exploit.

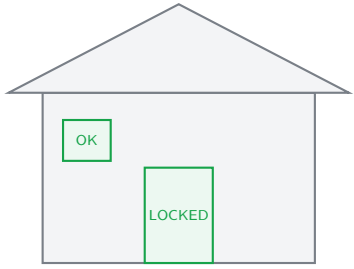
What Is Software Security?

- Ensuring correct and safe behavior under **adversarial** conditions.
- Common entry point for attacks:
  - **Software vulnerabilities**: flaws in code that attackers can exploit.
- Goal of software security:
  - Find and fix vulnerabilities **before attackers** discover them.

What Is Software Security?

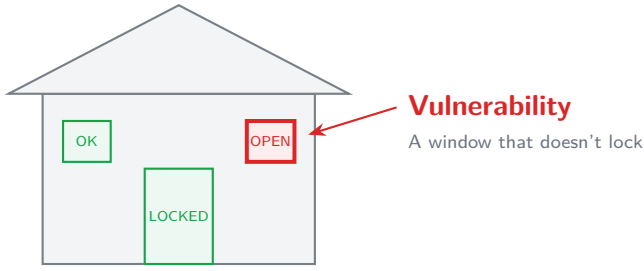
- Ensuring correct and safe behavior under **adversarial** conditions.
- Common entry point for attacks:
  - **Software vulnerabilities**: flaws in code that attackers can exploit.
- Goal of software security:
  - Find and fix vulnerabilities **before attackers** discover them.
- Security depends on assumptions:
  - Define the **threat model**
  - Attacker capabilities: local, remote, insider
  - Assets being protected

Vulnerability = A Flaw in Software



Your Software

Vulnerability = A Flaw in Software



A **weakness** in software that *could* be exploited to cause harm.

Unintentional. But attackers hunt for exactly these.

Common Vulnerability Classes

- **Buffer overflow**: writing data beyond allocated memory.

Common Vulnerability Classes

- **Buffer overflow**: writing data beyond allocated memory.
- **Race condition**: two threads clash over a shared resource.

Common Vulnerability Classes

- **Buffer overflow**: writing data beyond allocated memory.
- **Race condition**: two threads clash over a shared resource.
- **Logic error**: code does the wrong thing “correctly”.

Common Vulnerability Classes

- **Buffer overflow**: writing data beyond allocated memory.
- **Race condition**: two threads clash over a shared resource.
- **Logic error**: code does the wrong thing "correctly".
- Also includes: use-after-free, null pointer dereference, integer overflow, TOCTOU.
- Pattern-based tools catch the first two. **Logic errors** require *reasoning*.

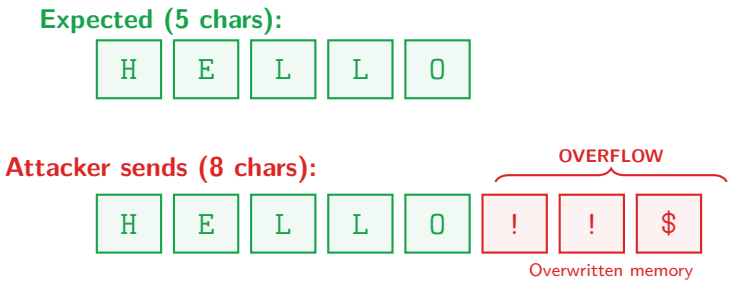
Buffer Overflow

The classic

Expected (5 chars):

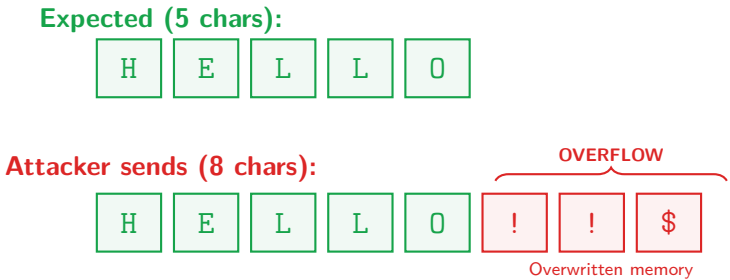
Buffer Overflow

The classic



Buffer Overflow

The classic



Data spills past its boundary → attacker can **inject code**.

`strcpy(buffer, input)` with no bounds check.

Vulnerability → Exploit → Impact

Vulnerability

The flaw

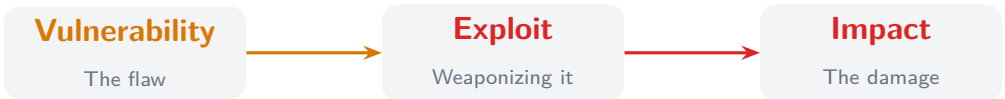
The vulnerability is the **unlocked window**.

Vulnerability → Exploit → Impact

The vulnerability is the **unlocked window**.

The exploit is the **burglar climbing through**.

Vulnerability → Exploit → Impact



The vulnerability is the **unlocked window**.

The exploit is the **burglar climbing through**.

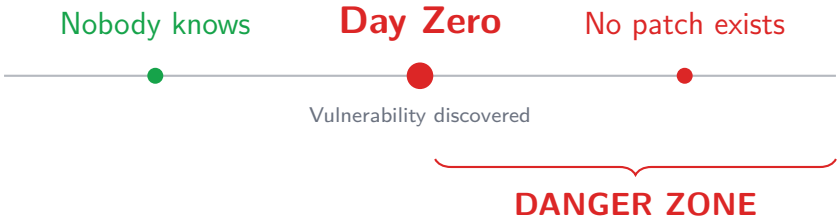
The impact is **everything they steal**.

Zero-Day Vulnerability

Nobody knows

Zero-Day Vulnerability

Zero-Day Vulnerability



A vulnerability **unknown to defenders**.

Zero time to prepare. Zero patches available.

Millions of lines. Thousands of contributors. Manual review is impossible.

Millions of lines. Thousands of contributors. Manual review is impossible.

How do we find bugs at scale?

Automated Code Analysis

Dynamic analysis: run the code, observe behavior at runtime

Fuzzing, unit testing, sanitizers (Valgrind, ASan)

Automated Code Analysis

Dynamic analysis: run the code, observe behavior at runtime

Fuzzing, unit testing, sanitizers (Valgrind, ASan)

Static analysis: inspect code without running it

Symbolic execution, abstract interpretation (Coverity, SVF, Infer)

Automated Code Analysis

Dynamic analysis: run the code, observe behavior at runtime

Fuzzing, unit testing, sanitizers (Valgrind, ASan)

Static analysis: inspect code without running it

Symbolic execution, abstract interpretation (Coverity, SVF, Infer)

Hybrid analysis: combine both

Each has trade-offs: false positives vs. missed bugs vs. scalability.

Fuzzing

Automated vulnerability discovery

Input Generator

Wu, **Nongpoh**, et al., "Fine-grained Coverage-based Fuzzing," ACM TOSEM 2024

Fuzzing

Automated vulnerability discovery

Wu, **Nongpoh**, et al., "Fine-grained Coverage-based Fuzzing," ACM TOSEM 2024

Fuzzing

Automated vulnerability discovery

Wu, **Nongpoh**, et al., "Fine-grained Coverage-based Fuzzing," ACM TOSEM 2024

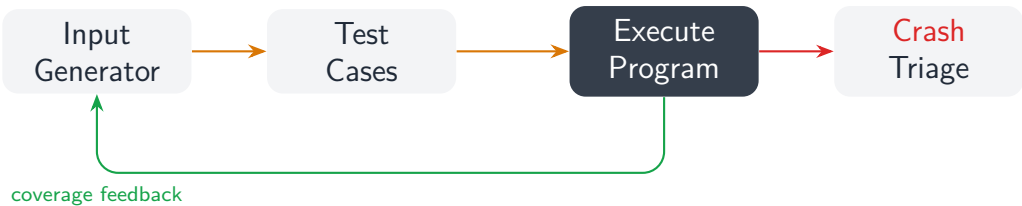
Fuzzing

Automated vulnerability discovery

Wu, **Nongpoh**, et al., "Fine-grained Coverage-based Fuzzing," ACM TOSEM 2024

Fuzzing

Automated vulnerability discovery



Wu, **Nongpoh**, et al., "Fine-grained Coverage-based Fuzzing," ACM TOSEM 2024

Limitations of Traditional Tools

Shallow coverage: fuzzers can't bypass complex sanity checks

Limitations of Traditional Tools

Shallow coverage: fuzzers can't bypass complex sanity checks

False positives: static analysis flags theoretical issues

Limitations of Traditional Tools

Shallow coverage: fuzzers can't bypass complex sanity checks

False positives: static analysis flags theoretical issues

No semantics: tools don't understand *intent* or business logic

Limitations of Traditional Tools

Shallow coverage: fuzzers can't bypass complex sanity checks

False positives: static analysis flags theoretical issues

No semantics: tools don't understand *intent* or business logic

Auth bypasses, logic errors, design flaws all require *reasoning*.

What if we had something that could reason about code like a human?

A new player. . .

A new player...

Large Language Models

A new player...

Large Language Models

Can they do what human security researchers do?

LLMs Can Find Real Zero-Days

- **Google Big Sleep** (2024): first LLM-found zero-day in SQLite[1]

[1] Project Zero Blog [2] blog.mozilla.org [3] InfoQ [4] red.anthropic.com [5] cybergym.io

LLMs Can Find Real Zero-Days

- **Google Big Sleep** (2024): first LLM-found zero-day in SQLite[1]
- **Anthropic + Mozilla** (Feb 2026): 22 vulns in Firefox, 14 high severity[2]

[1] Project Zero Blog [2] blog.mozilla.org [3] InfoQ [4] red.anthropic.com [5] cybergym.io

LLMs Can Find Real Zero-Days

- **Google Big Sleep** (2024): first LLM-found zero-day in SQLite[1]
- **Anthropic + Mozilla** (Feb 2026): 22 vulns in Firefox, 14 high severity[2]
- **Carlini, [un]prompted** (Mar 2026): Linux kernel heap overflow, **23 years old**[3]

[1] Project Zero Blog [2] blog.mozilla.org [3] InfoQ [4] red.anthropic.com [5] cybergym.io

LLMs Can Find Real Zero-Days

- **Google Big Sleep** (2024): first LLM-found zero-day in SQLite[1]
- **Anthropic + Mozilla** (Feb 2026): 22 vulns in Firefox, 14 high severity[2]
- **Carlini, [un]prompted** (Mar 2026): Linux kernel heap overflow, **23 years old**[3]
- **Claude Mythos Preview** (Apr 2026): zero-days in **major OSes & browsers**[4]

[1] Project Zero Blog [2] blog.mozilla.org [3] InfoQ [4] red.anthropic.com [5] cybergym.io

LLMs Can Find Real Zero-Days

- **Google Big Sleep** (2024): first LLM-found zero-day in SQLite[1]
- **Anthropic + Mozilla** (Feb 2026): 22 vulns in Firefox, 14 high severity[2]
- **Carlini, [un]prompted** (Mar 2026): Linux kernel heap overflow, **23 years old**[3]
- **Claude Mythos Preview** (Apr 2026): zero-days in **major OSes & browsers**[4]
- **CyberGym / UC Berkeley** (ICLR 2026): agents found **35 zero-days**[5]

[1] Project Zero Blog [2] blog.mozilla.org [3] InfoQ [4] red.anthropic.com [5] cybergym.io

How LLMs Differ from Traditional Tools

- **Static analyzers** report potential vulnerabilities
  - Often produce significant false positives at scale
  - Require substantial human triage and prioritization

How LLMs Differ from Traditional Tools

- **Static analyzers** report potential vulnerabilities
  - Often produce significant false positives at scale
  - Require substantial human triage and prioritization
- **Fuzzers** discover crashing inputs through automated testing.
  - Provide limited semantic understanding of exploitability

How LLMs Differ from Traditional Tools

- **Static analyzers** report potential vulnerabilities
  - Often produce significant false positives at scale
  - Require substantial human triage and prioritization
- **Fuzzers** discover crashing inputs through automated testing.
  - Provide limited semantic understanding of exploitability
- **LLM agents** can reason across codebases, generate hypotheses, invoke tools iteratively, and explain potential attack paths.

How LLMs Differ from Traditional Tools

- **Static analyzers** report potential vulnerabilities
  - Often produce significant false positives at scale
  - Require substantial human triage and prioritization
- **Fuzzers** discover crashing inputs through automated testing.
  - Provide limited semantic understanding of exploitability
- **LLM agents** can reason across codebases, generate hypotheses, invoke tools iteratively, and explain potential attack paths.

Agentic LLM-based vulnerability research is emerging as a new category of security tooling.

The Needle-in-the-Haystack Problem

"Find all vulnerabilities" produces breadth-first **hallucination**.

Devansh, "Needle in the Haystack: LLMs for Vulnerability Research," Mar 2026

The Needle-in-the-Haystack Problem

"Find all vulnerabilities" produces breadth-first **hallucination**.

No threat model: long lists of generic CWE possibilities, no prioritization

Devansh, "Needle in the Haystack: LLMs for Vulnerability Research," Mar 2026

The Needle-in-the-Haystack Problem

"Find all vulnerabilities" produces breadth-first **hallucination**.

No threat model: long lists of generic CWE possibilities, no prioritization

Context rot: more tokens in context $\rightarrow$ worse at finding "needles"

Devansh, "Needle in the Haystack: LLMs for Vulnerability Research," Mar 2026

The Needle-in-the-Haystack Problem

"Find all vulnerabilities" produces breadth-first **hallucination**.

No threat model: long lists of generic CWE possibilities, no prioritization

Context rot: more tokens in context $\rightarrow$ worse at finding "needles"

30+ real CVEs have been found using LLMs: but **none** via "find all vulnerabilities."

Devansh, "Needle in the Haystack: LLMs for Vulnerability Research," Mar 2026

The Needle-in-the-Haystack Problem

"Find all vulnerabilities" produces breadth-first **hallucination**.

No threat model: long lists of generic CWE possibilities, no prioritization

Context rot: more tokens in context $\rightarrow$ worse at finding "needles"

30+ real CVEs have been found using LLMs: but **none** via "find all vulnerabilities."

Every success used: threat model + thin slice + **tool-based verification**.

Devansh, "Needle in the Haystack: LLMs for Vulnerability Research," Mar 2026

LLM Alone vs. LLM + Analysis Tools

LLM Alone

Generic CWE checklists

Hallucinated exploits

No way to reproduce

30–60% false positives

LLM Alone vs. LLM + Analysis Tools

LLM Alone

Generic CWE checklists

Hallucinated exploits

No way to reproduce

30–60% false positives

VS

LLM + Analysis Tools

Guided by fuzzer coverage

Validated by sanitizers

Reproducible PoC exploits

$<5\%$ false positive rate

LLM Alone vs. LLM + Analysis Tools

LLM Alone

Generic CWE checklists

Hallucinated exploits

No way to reproduce

30–60% false positives

VS

LLM + Analysis Tools

Guided by fuzzer coverage

Validated by sanitizers

Reproducible PoC exploits

<5% false positive rate

Tools give the LLM **ground truth**. Without them, you get plausible fiction.

DARPA AI Cyber Challenge (AIxCC)

- **AIxCC**: DARPA competition on autonomous vulnerability discovery and patching.
- Goal: build AI systems that can find vulnerabilities, generate exploits, and repair software automatically.
- Systems operated continuously on real-world open-source software at machine speed.

Source: darpa.mil/research/programs/ai-cyber-challenge

AIxCC: Autonomous Security Pipelines

- AIxCC systems combined: LLM agents, program analysis, symbolic execution, fuzzing, automated patch validation.
- Final competition: autonomous systems competed to discover and patch vulnerabilities faster than other teams.

AIxCC marked a transition from isolated tools to **autonomous AI-driven security pipelines**.

Source: darpa.mil/research/programs/ai-cyber-challenge

ATLANTIS

DARPA AIxCC Winner: Team Atlanta, Taesoo Kim[1]

Target

C/C++, Java

[1] ATLANTIS: AIxCC Finalist System Report, arXiv 2025

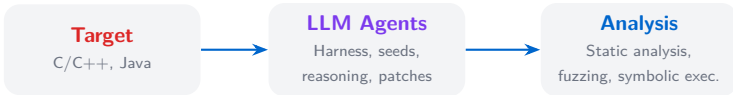
ATLANTIS

DARPA AIxCC Winner: Team Atlanta, Taesoo Kim[1]

[1] ATLANTIS: AIxCC Finalist System Report, arXiv 2025

ATLANTIS

DARPA AIxCC Winner: Team Atlanta, Taesoo Kim[1]



[1] ATLANTIS: AIxCC Finalist System Report, arXiv 2025

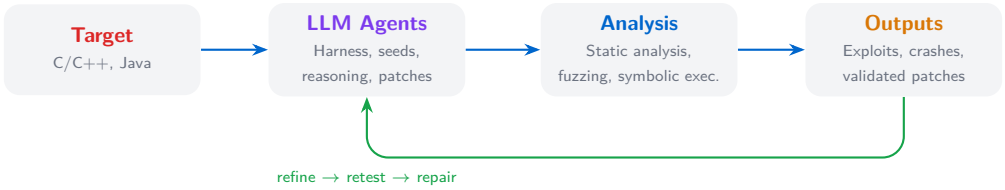
ATLANTIS

DARPA AIxCC Winner: Team Atlanta, Taesoo Kim[1]

[1] ATLANTIS: AIxCC Finalist System Report, arXiv 2025

ATLANTIS

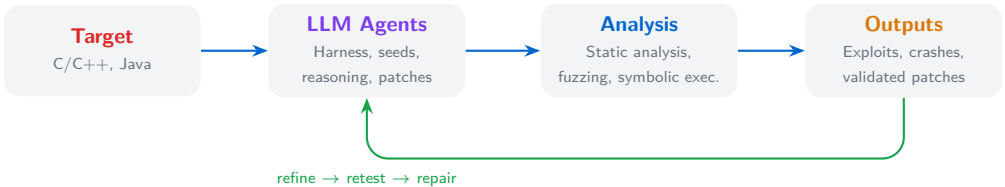
DARPA AIxCC Winner: Team Atlanta, Taesoo Kim[1]



[1] ATLANTIS: AIxCC Finalist System Report, arXiv 2025

ATLANTIS

DARPA AIxCC Winner: Team Atlanta, Taesoo Kim[1]



ATLANTIS combines **LLM reasoning** with automated program analysis and fuzzing.

LLMs generate harnesses, inputs, hypotheses, and patches; the analysis infrastructure validates every result.

[1] ATLANTIS: AIxCC Finalist System Report, arXiv 2025

The lesson from ATLANTIS:

LLM alone $\neq$ **reliable security analysis**

The lesson from ATLANTIS:

LLM alone $\neq$ **reliable security analysis**

LLM + Fuzzing + Program Analysis
= vulnerabilities discovered, validated, and repaired

The lesson from ATLANTIS:

LLM alone $\neq$ **reliable security analysis**

LLM + Fuzzing + Program Analysis
= vulnerabilities discovered, validated, and repaired

The LLM generates and refines hypotheses. The analysis infrastructure provides **ground truth**.

The Shrinking Window

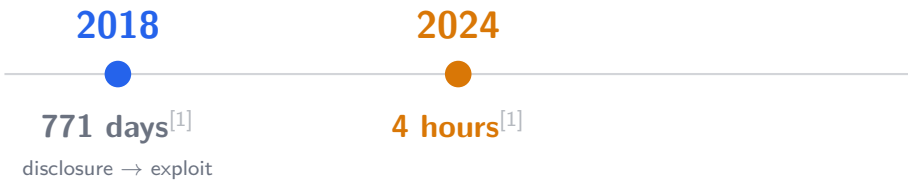
2018

771 days[1]

disclosure → exploit

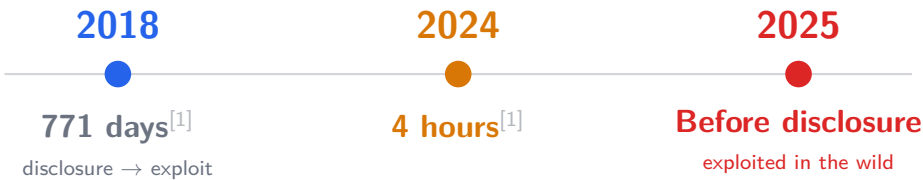
[1] zerodayclock.com

The Shrinking Window



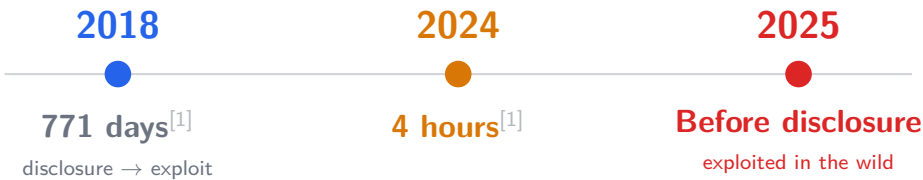
[1] zerodayclock.com

The Shrinking Window



[1] zerodayclock.com

The Shrinking Window



"AI can reverse-engineer a patch and generate a working exploit in minutes."

: Sergej Epp, Zero Day Clock

[1] zerodayclock.com

The Global Cybersecurity Landscape

89% increase in AI-enabled attacks (2025)[1]

[1] CrowdStrike GTR 2025 [2] Rest of World, May 2026

The Global Cybersecurity Landscape

89% increase in AI-enabled attacks (2025)[1]

- Advanced defensive AI systems remain concentrated among a small number of organizations.

[1] CrowdStrike GTR 2025 [2] Rest of World, May 2026

The Global Cybersecurity Landscape

89% increase in AI-enabled attacks (2025)[1]

- Advanced defensive AI systems remain concentrated among a small number of organizations.
- Many governments, financial institutions, and critical infrastructure operators lack access to state-of-the-art AI security tooling.

[1] CrowdStrike GTR 2025 [2] Rest of World, May 2026

The Global Cybersecurity Landscape

89% increase in AI-enabled attacks (2025)[1]

- Advanced defensive AI systems remain concentrated among a small number of organizations.
- Many governments, financial institutions, and critical infrastructure operators lack access to state-of-the-art AI security tooling.
- The gap is especially visible across parts of the **Asia-Pacific** region, including **India**.

[1] CrowdStrike GTR 2025 [2] Rest of World, May 2026

Key Takeaways

- LLMs can reason about code and identify logic flaws that traditional tools may miss.

Key Takeaways

- LLMs can reason about code and identify logic flaws that traditional tools may miss.
- LLMs alone may hallucinate findings that are plausible but unverifiable.

Key Takeaways

- LLMs can reason about code and identify logic flaws that traditional tools may miss.
- LLMs alone may hallucinate findings that are plausible but unverifiable.
- **LLM + fuzzing + program analysis** enables discovery of verified and exploitable vulnerabilities.

Key Takeaways

- LLMs can reason about code and identify logic flaws that traditional tools may miss.
- LLMs alone may hallucinate findings that are plausible but unverifiable.
- **LLM + fuzzing + program analysis** enables discovery of verified and exploitable vulnerabilities.
- Hybrid systems such as ATLANTIS demonstrated autonomous vulnerability discovery and repair at AIxCC scale.

Thank You

Questions?