بسم الله الرحمن الرحیم

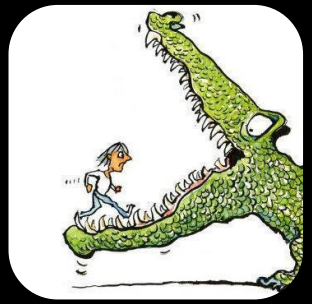
Deep Analysis , Precise Decisions , Rapid Responses
New Frontiers of Cybersecurity with AI

Solmaz Salimi
s.salimi@sharif.edu

October 2025

Who am I?

- (not) paid to watch programs and not find bugs!
- Seasoned CTF player with lots of stories!
- Directly listen to my brain noises `CQ CQ CQ EP2SOL`!



Solmaz Salimi

2010 Start working on security for my bachelor's degree final project

2012 Got familiar with CTFs, co-founded Asis CTF, the first Iran International CTF

2013 Start my Master at IUST, working on the Network Security and Botnet Detection

2015 Start my Ph.D. at Sharif University, working on Software Security and Vulnerability Detection

2022 Invited to work as a researcher at HexHive group, EPFL. Working on Kernel fuzzing

2023 Start my Postdoc with S3 group at Eurecom, working on System Security

2025 

A Look Back: The First Inflection Point

- Remember the mid-2010s? In 2016, DARPA hosted the Cyber Grand Challenge.
- **The big question**: Could a machine autonomously find, prove, and fix software vulnerabilities in real-time?
- Many were skeptical about the power of automated program analysis, symbolic execution, and fuzzing.

A Look Back: The First Inflection Point

- A massive acceleration in **automated vulnerability research** and the commercialization of fuzzing.
- It proved that **automated systems could reason about security** at machine speed.

Today: The AI Inflection Point

- Fast forward to 2022. DARPA announced the AI Cyber Challenge (AIxCC).
- The new question: Can we leverage the incredible power of modern AI to secure the nation's most critical software?
- The goal is to build a **new generation of AI-powered systems** that can secure codebases far beyond human scale.
- Just like the CGC sparked a revolution in automated bug-finding, the AIxCC signals that we are at another pivotal moment.

AI is no longer just a research idea; it's the next frontier for practical cybersecurity.

Why Now? The Problem of Scale & Complexity

Traditional, manual security methods are breaking. Signature-based and curated rules simply can't keep up with the data deluge.

- **Code**: Millions of binary functions across dozens of architectures.
- **Logs**: Petabytes of system, cloud, and network logs generated daily.
- **Malware**: Large, polymorphic families that constantly evolve.
- **Traffic**: Pervasive encryption that limits visibility.
- **Hardware**: Massive volumes of side-channel traces from embedded devices.

We need a new paradigm!

What Does AI Bring to the Table? 🤖

AI helps us move from matching simple patterns to understanding complex behavior. It introduces three key capabilities:

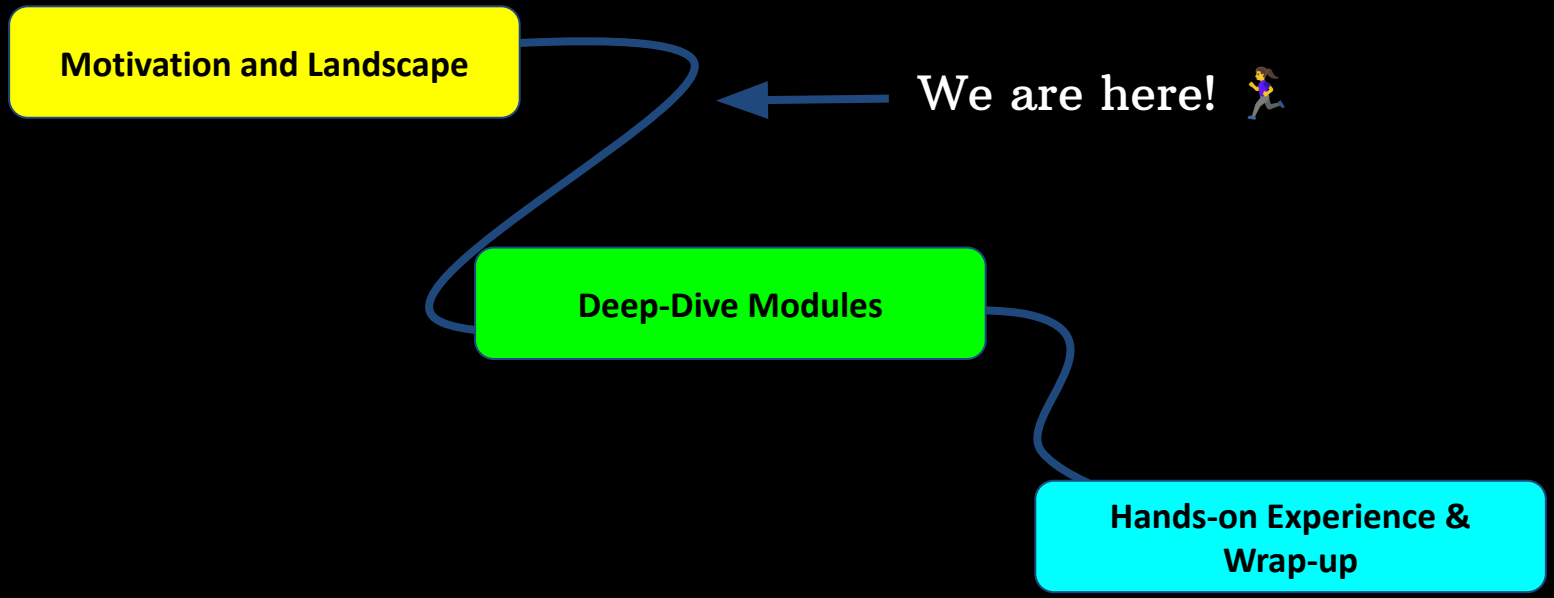
- **Semantic Learning**: AI models can learn the intent and behavior of programs, network traffic, and user actions, not just their syntax.
- **Adaptivity**: AI-driven systems can adapt to evolving malware, new attack surfaces, and changing system behavior without needing constant manual rule updates.
- **Generation:** Modern AI can automatically generate security content, from detection rules and pseudo-code to complex fuzzing inputs.

Early Wins: From Research to Reality 🌅🐦

This isn't just theory. AI is already delivering concrete results across the security landscape:

- Binary Similarity: Using AI to find similar functions across different CPU architectures for vulnerability hunting.
- Log Analysis: Treating logs as a language, allowing Transformers and LLMs to detect subtle anomalies that rules would miss.
- Protocol Fuzzing: Guiding fuzzers with LLMs to automatically infer grammars and state machines, finding deeper bugs.
- Malware Analysis: Assisting reverse engineers with Transformer models that provide context and summaries of malicious behavior.
- Side-Channel Analysis: Applying deep learning to analyze power traces, making hardware attacks more practical and accessible.

Workshop Agenda



Deep-Dive Modules

Module 1: Software Security

Module 2: Runtime Monitoring

Module 3: Malware Analysis

Module 4: Side-Channel & ICS/IoT

Module 5: Agentic SecOps & Automation

Module 6: Guardrails & Security of Agentic AI

Module 1

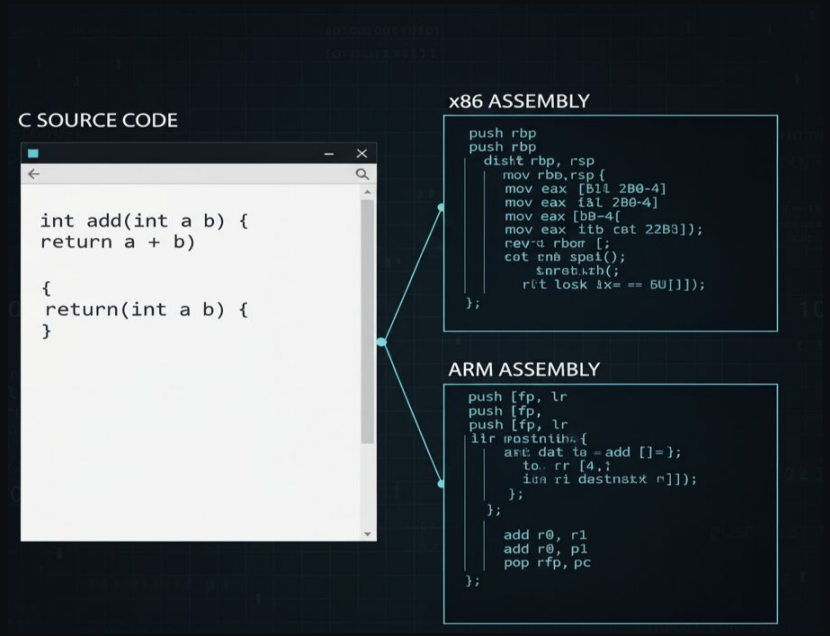
Software Security

Program Understanding, Vulnerability Discovery and LLM-Guided Fuzzing

The core problem

Syntactic analysis is hard

- The same logic looks completely different across compilers and architectures, making scaled analysis nearly impossible.

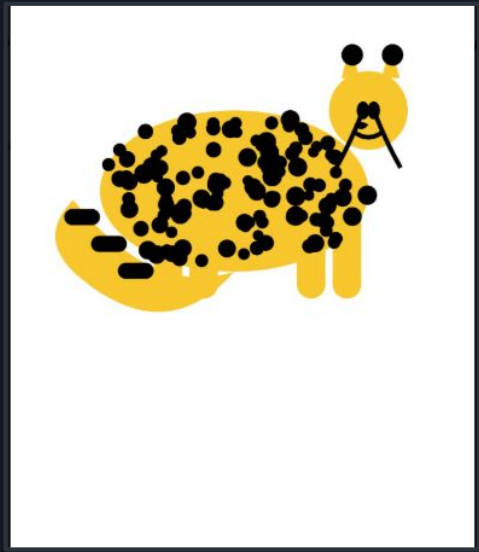


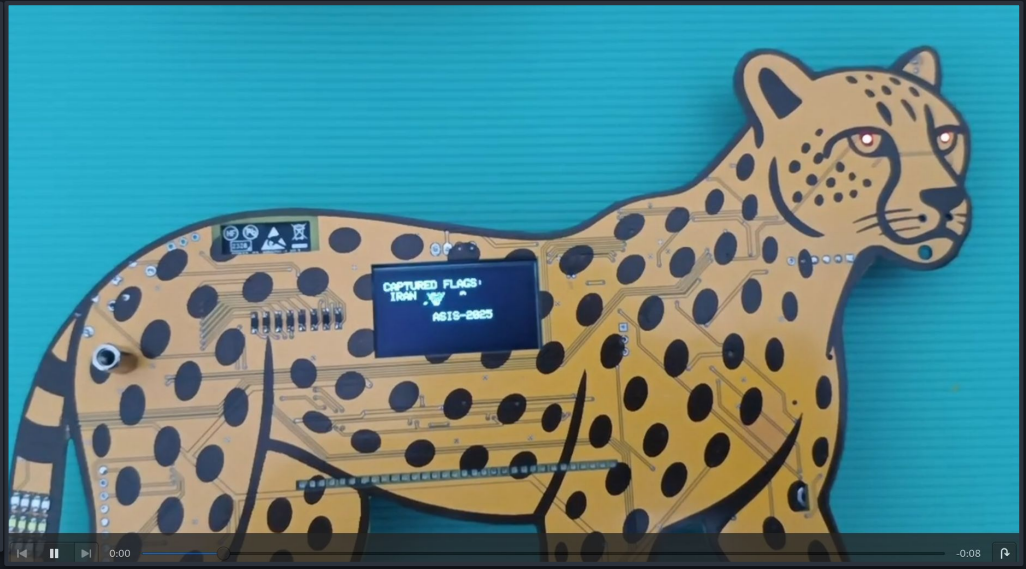
Decompilation in-the-loop

- IDA pro
- Ghidra
- LLM4Decompile

We can understand binaries, so every program is understable, we can even generate the original source code …

Decompilation in-the-loop





The Foundational Challenge: Program Understanding

Why Syntactic Analysis is Hard?

Program Understanding in security requires a deep comprehension of a program's logic, data flow, and control flow to identify potential weaknesses.

Traditional static analysis (SAST) struggles because pure syntactic analysis is difficult and often insufficient:

- **Language Complexity:** Modern languages like C++ involve complex features (templates, pointers, operator overloading) that are hard for automated tools to parse perfectly.
- **Scale** and **Volume:** Enterprise codebases contain millions of lines of code, making deep, exhaustive analysis computationally expensive.
- **Ambiguity:** Code can be ambiguous without the developer's original context or intent. A function might be insecure in one context but safe in another.
- **Limitations of Traditional Tools:**
  - **High False Positives:** SAST tools often flag safe code as vulnerable.
  - **High False Negatives:** They miss complex, logic-based vulnerabilities.

Traditional Vulnerability Discovery - Fuzzing

Fuzzing is an automated testing technique that provides invalid, unexpected, or random data as input to a program to find crashes and vulnerabilities.

Traditional Vulnerability Discovery - Fuzzing

```
$ ./testme --help
Usage: testme <int32_arg>

$ ./testme AAAA
Please enter an integer!

$ cat fuzzer.sh
while :
do
  input="$(dd if=/dev/urandom bs=4 count=1)"
  ./testme $input || echo $input >> crash_seeds
done
```

Traditional Vulnerability Discovery - Fuzzing

Common Fuzzing Techniques:

- **Black-box (Dumb) Fuzzing:** Generates random inputs with no knowledge of the program's internal structure. It's fast but inefficient.
- **Grey-box (Coverage-Guided) Fuzzing:** A major improvement (e.g., **AFL**, **libFuzzer**). It uses lightweight instrumentation to track which inputs exercise new code paths and prioritizes mutating them.

Limitations of Traditional Fuzzing:

- **Getting Stuck:** Fuzzers struggle to get past "magic number" checks, checksums, or complex parsing logic (e.g., `if (input_header.checksum == 0xDEADBEEF)`).
- **Shallow Code Coverage:** They often repeatedly test simple, easy-to-reach code paths and fail to explore deeper, more complex program states.
- **Lack of Structure Awareness:** They are inefficient at generating highly structured inputs (e.g., valid PNG files, complex network protocols).

The AI Revolution - Enter Large Language Models (LLMs)

A Paradigm Shift in Program Understanding

Large Language Models (LLMs) like GPT, Gemini, and Codex are trained on massive datasets of source code, documentation, and natural language. This gives them an unprecedented ability to understand software.

How LLMs Surpass Syntactic Analysis:

- **Semantic Understanding:** LLMs move beyond syntax to grasp the *meaning* and *purpose* of code. They can infer developer intent from function names, comments, and code structure.
- **Context Awareness:** They can analyze code in the context of the entire project, including its documentation and related modules.
- **Pattern Recognition:** They excel at recognizing common coding patterns, both good (design patterns) and bad (vulnerability patterns).

LLMs provide a powerful new lens to overcome the limitations of traditional analysis, enabling a deeper and more intuitive understanding of complex codebases.

LLM-Guided Fuzzing - The Next Generation

Smarter Inputs, Deeper Bugs

LLM-Guided Fuzzing uses a Large Language Model to generate intelligent inputs that guide a fuzzer toward interesting and hard-to-reach parts of a program.

How It Works:

1. **Code Analysis:** The LLM reads the target program's source code.
2. **Input Generation:** It identifies complex data structures, protocols, and conditional checks. Based on this understanding, it generates high-quality, valid inputs (seeds) that are likely to satisfy these conditions.
3. **Intelligent Mutation:** Instead of just flipping random bits, the LLM can mutate these seeds in a semantically meaningful way (e.g., changing a specific field in a network packet header).
4. **Fuzzing Execution:** These intelligent inputs are fed to a traditional fuzzer (like `AFL++`), which can now explore deep code paths that were previously unreachable.

This approach combines the semantic understanding of LLMs with the raw speed and power of coverage-guided fuzzers.

Evidence of Progress and Key Publications

- **Key Improvements:**
  - **Increased Code Coverage:** LLM-guided fuzzers consistently achieve higher code coverage faster than their traditional counterparts.
  - **Novel Vulnerability Discovery:** They have successfully found new, real-world vulnerabilities (CVEs) in well-tested software that were missed by standard fuzzing campaigns.

- **Important Research and Publications:**
  - **Large Language Model guided Protocol Fuzzing → NDSS 2024**
  - **Fuzz4All: Universal Fuzzing with Large Language Models →ICSE 2024**
  - **Large Language Models for Fuzzing Parsers (Google, 2024):** Showcased the ability of LLMs to generate grammatically valid inputs for parsers, a classic challenge for fuzzers.

The Future of AI in Software Security

Towards Autonomous Vulnerability Discovery and Repair

LLM-guided fuzzing is just the beginning. The future of AI in software security is moving towards a more autonomous and proactive ecosystem.

- **Automated Program Repair:** The next frontier is not just finding bugs, but automatically fixing them. LLMs are already capable of suggesting or even generating code patches for vulnerabilities they discover.
- **AI-Powered Secure Development:** Imagine an AI assistant that acts as an expert security code reviewer in real-time within the IDE, flagging potential vulnerabilities and suggesting secure alternatives as a developer types.
- **Autonomous Security Agents:** Future systems may autonomously scan code, discover vulnerabilities, generate exploits to verify them, and deploy patches, all with minimal human intervention.

What's Still Hard ⚠️

- LLM oracle reliability & hallucinations require human verification.
- Code obfuscation and complex environments (concurrency, JIT) are still major hurdles.
- Ensuring the correctness and safety of AI-generated patches.
- Managing the computational cost of large-scale analysis.

Module 2

Runtime Monitoring

Logs/telemetry and network visibility under TLS/QUIC

Lots of Logs, Lots of encrypted traffic!

- **Runtime Monitoring:** The continuous process of collecting, analyzing, and responding to network and system events in real-time to identify malicious activities and performance issues.
- **The Main Challenge:** Today, over 90% of web traffic is encrypted with **TLS/QUIC** protocols. While essential for privacy, this encryption blinds traditional security tools like firewalls and Intrusion Detection Systems (IDS).
- **The Problem:** Attackers use these same encrypted channels to hide their activities:
  - Command and Control (C2) communication
  - Data Exfiltration
  - Downloading new malware

Modern Solution: Instead of inspecting packet content (Deep Packet Inspection), we must analyze traffic **Metadata** and **Behavior**.

Analyzing Encrypted Traffic Without Decryption

- How Do We See Invisible Traffic? With the rise of TLS 1.3 and QUIC, a large part of the handshake process is also encrypted. This makes certificate-based identification more difficult.

Solution: Focus on Metadata

- Even without breaking the encryption, every network connection provides valuable information that can be analyzed.

Fingerprinting - The Digital Fingerprint of Clients and Servers

- Every Application Has a Unique Fingerprint
- When initiating a TLS connection, every application (browser, malware, Python script) presents specific parameters to the server. The combination of these parameters creates a detectable digital Fingerprint.
- Introducing JA3 and JA3S

The Role of NDR Systems in Threat Detection

NDR: Hunting Threats in Network Traffic

How Does NDR Use Fingerprints?

Network Detection and Response (NDR) systems are specifically designed to monitor and analyze network traffic (both north-south traffic at the perimeter and east-west traffic between servers).

1. **Data Collection**
2. **Comparison with Known Threats**
3. **Anomaly Detection**

Artificial Intelligence (AI) - The Mastermind in NDR

The Role of AI in Fingerprint Analysis and Advanced Threat Detection

Beyond Simple Matching: The Power of Machine Learning

Comparing fingerprints against known lists is not enough to combat new threats and zero-day attacks. This is where **Artificial Intelligence (AI)** and **Machine Learning (ML)** play a crucial role.

AI Applications in NDR:

1. **Baseline Establishment**
2. **Intelligent Anomaly Detection**
3. **False Positive Reduction**

Log Analysis as Language (Transformers/LLMs for anomalies)

- DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning (CCS 2017)
- Interpretable Federated Transformer Log Learning for Cloud Threat Forensics (NDSS 2022)
- PROGRAPHER: An Anomaly Detection System based on Provenance Graph Embedding (Usenix 2023)

What's Still Hard ⚠️

- Concept drift and seasonality across tenants
- Label scarcity & noisy ground truth
- QUIC/ECH evolution breaking fragile fingerprints

Module 3

Malware Analysis

Deep learning & Transformers for static/dynamic signals

The Evolving Threat Landscape

Traditional malware analysis methods are struggling to keep up with the sophistication and volume of modern threats.

Key Challenges:

- **Polymorphism & Metamorphism:** Malware constantly changes its code (syntax) to evade signature-based detection, while keeping its malicious behavior (semantics) intact.
- **Obfuscation & Packing:** Attackers use advanced packing and encryption techniques to hide the malicious payload, making static analysis ineffective.
- **Massive Volume:** Millions of new malware samples are created daily, making manual analysis and signature generation impossible to scale.
- **Fileless Malware:** Threats that exist only in memory are invisible to traditional file-scanning antivirus engines.

We need automated, intelligent systems that can understand a program's *behavior* and *intent*, not just its static signature.

The AI Revolution in Malware Detection

From Reactive Signatures to Proactive Detection

The rise of Artificial Intelligence (AI) and Machine Learning (ML) marked a fundamental shift from a reactive to a proactive security posture.

Early AI/ML Approaches (e.g., SVM, Random Forests):

- Analyzed curated features extracted from files (e.g., API imports, strings, file header info).
- **Limitation:** Required extensive and brittle **manual feature engineering**. An expert had to decide what features were important, and attackers could easily learn to evade them.

The Deep Learning Breakthrough:

- Deep learning models can learn relevant features **automatically** directly from raw data.
- This eliminates the need for manual feature engineering and allows the model to discover complex, non-obvious patterns of maliciousness.

Deep Learning for Static Analysis (Code at Rest)

Deep learning models can analyze malicious code without ever running it.

Common Techniques:

1. **Visualizing Binaries as Images (CNNs):**
   - A binary file byte sequence is converted into a 2D grayscale image.
   - **Convolutional Neural Networks (CNNs)**, the same models used for image recognition, are trained to "see" the textural and structural patterns of malware.
   - **Advantage:** This method is highly resilient to simple code obfuscation, as the overall "texture" of the malware often remains similar.
2. **Analyzing Byte Sequences (RNNs/1D CNNs):**
   - The binary is treated as a one-dimensional sequence of bytes.
   - **Recurrent Neural Networks (RNNs)** or 1D CNNs can analyze this sequence to find malicious patterns.

Key Research:

- **"Malware Images: Visualization and Automatic Classification" (Nataraj et al., 2011):** A pioneering paper that introduced the concept of malware classification using image processing techniques.

Deep Learning for Dynamic Analysis (Code in Motion)

Dynamic analysis involves running the malware in a safe, isolated environment (sandbox) and observing its behavior.

How Deep Learning Helps:

- The behavior is recorded as a **sequence of events**: API calls, file system changes, registry modifications, and network requests.
- **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks are ideal for analyzing this sequential data.
- The model learns the "grammar" of malicious behavior, such as:
  - `CreateProcess` -> `VirtualAllocEx` -> `WriteProcessMemory` -> `CreateRemoteThread` (a common process injection pattern).

Advantage: This approach focuses on the malware's ultimate actions, making it effective against even the most heavily obfuscated or packed samples.

Enter Transformers - The Next Leap Forward

Transformers, originally designed for natural language processing (NLP), have proven to be exceptionally powerful for malware analysis.

Why Transformers? The Power of the Attention Mechanism:

- Unlike RNNs which process data sequentially, the **attention mechanism** allows a Transformer to weigh the importance of all parts of the input simultaneously.
- It can identify critical relationships between distant parts of a file or a long sequence of API calls.
- **Example:** It can connect a suspicious function call at the beginning of an execution trace to a data exfiltration attempt much later on.

Application: Transformers can be applied to both static analysis (treating bytes as "words") and dynamic analysis (treating API calls as "tokens" in a sentence).

Key Research:

- **"MalBERT: A Transformer-based approach for Malware Detection" (U-Sardar et al., 2021):** One of the early works demonstrating how a BERT-like model can be effectively pre-trained and fine-tuned for malware classification from raw byte sequences.

Key Shift #3

Robustness Matters

Conclusion and Future Outlook

The Future is Autonomous and Proactive

- **Summary:** AI, and particularly Deep Learning, has fundamentally shifted malware analysis from a manual, reactive process to an automated and predictive one. Transformers represent the current state-of-the-art, offering a deeper understanding of code and behavior.
- **Future Trends:**
  - **Graph Neural Networks (GNNs):** Modeling programs as graphs (e.g., control-flow graphs) and using GNNs to analyze them is a promising new direction.
  - **Multimodal Analysis:** Combining static, dynamic, and network signals into a single, powerful AI model.
  - **AI-Driven Response:** Moving beyond detection to automated response, where AI can generate signatures or even patch systems in real-time.

The cat-and-mouse game between attackers and defenders continues, but AI provides defenders with an unprecedented advantage to automate, scale, and predict.

What's Still Hard ⚠️

- Adversarial Attacks: Malware authors are now actively designing samples to fool AI detectors. By making subtle, calculated changes to a file, they can cause a model to misclassify it as benign.
- Explainability (XAI): Deep learning models are often "black boxes." It is extremely difficult to understand why a model flagged a file as malicious, which complicates incident response and forensics.
- Concept Drift: The malware landscape evolves constantly. AI models trained on yesterday's threats may become less effective against tomorrow's. This requires a robust infrastructure for continuous retraining and evaluation.
- High Computational Cost: Training and deploying large-scale Transformer models requires significant computational resources and expertise, making it inaccessible for some organizations.

Module 4

Side-Channel & OT

Hardware/ICS/IoT security with ML pipelines

The Invisible Battlefield - Understanding the Threats

Side-Channel Attacks (SCA):

- These attacks don't exploit software bugs. Instead, they exploit **physical information leakage** from a device as it operates.
- **Examples:** Power consumption (Power Analysis), electromagnetic (EM) emissions, timing information, and even acoustic signals can reveal secret data like cryptographic keys.

Operational Technology (OT) / ICS / IoT Security:

- **OT/ICS:** The hardware and software that monitor and control physical industrial processes (e.g., power grids, manufacturing plants).
- **Challenges:** These systems are often legacy, designed for reliability not security, and have real-time constraints. They are increasingly connected to IT networks, expanding their attack surface.
- **IoT Convergence:** Billions of IoT devices now bridge the physical and digital worlds, making them prime targets for both traditional cyber-attacks and physical attacks like SCAs.

The AI Revolution in Physical Security

Before AI / Traditional Methods:

- **SCA:** Relied on manual, expert-driven statistical analysis like **Differential Power Analysis (DPA)**. This required deep knowledge of the hardware and was slow, requiring thousands of measurements (traces).
- **OT/ICS Security:** Relied on network segmentation and rule-based systems (like IDS signatures), which were blind to zero-day attacks and anomalous operational behavior.

The AI Paradigm Shift:

- **Machine Learning (ML)**, and especially **Deep Learning**, automates the detection process by learning complex patterns directly from raw sensor and signal data.
- **Key Advantage:** AI eliminates the need for manual feature engineering. The model itself discovers what a "malicious" power trace or an "anomalous" sensor reading looks like, making detection faster, more scalable, and more effective.

ML Pipelines for Side-Channel Attack (SCA) Detection

Learning to "See" Leaking Secrets

Deep Learning models, particularly **Convolutional Neural Networks (CNNs)**, are highly effective at breaking cryptographic implementations.

The ML Pipeline for SCA:

1. **Data Collection (Profiling):** Capture thousands of power or EM traces from a target device while it performs cryptographic operations with a known key.
2. **Model Training:** Train a CNN directly on the raw traces. The model learns to correlate subtle patterns in the signal with the specific cryptographic key bit being processed (e.g., classifying a trace as corresponding to a '0' or a '1').
3. **Attack Phase:** Use the trained model to predict the secret key, one bit at a time, from a new, small set of traces from a victim device.

Key Research & Impact:

- Deep learning has drastically reduced the number of traces needed for a successful attack, making SCAs far more practical.
- **Key Conferences:** Research in this area is heavily published at **CHES** (Conference on Cryptographic Hardware and Embedded Systems) and **USENIX Security**.

ML Pipelines for OT/ICS Anomaly Detection

Learning the "Heartbeat" of Industrial Processes

OT networks are highly predictable. AI excels at learning this normal behavior and flagging any deviation.

The ML Pipeline for OT Security:

1. **Baseline Creation:** An AI model (often an **LSTM** or **Autoencoder**) is trained on historical network and sensor data from the live OT environment. It learns the normal "rhythm" of operations; which sensors talk to which controllers, what their value ranges are, and in what sequence.
2. **Anomaly Detection:** The trained model monitors the live system in real-time. Any behavior that deviates significantly from the learned baseline is flagged as an anomaly.
3. **Alerting:** This could be an unknown device connecting, a sensor providing physically impossible values, or a controller receiving an unexpected command; all potential indicators of a compromise like Stuxnet.

Significant Progress and Real-World Success

How AI is Making a Measurable Difference

In Side-Channel Analysis:

- **Breaking Countermeasures:** Deep learning models have successfully defeated hardware countermeasures (like masking and shuffling) that were designed to protect against traditional statistical attacks.
- **Reduced Attack Cost:** The number of required traces has dropped from tens of thousands to just a few hundred in many cases, lowering the barrier to entry for attackers.

In OT/ICS/IoT Security:

- **Early Threat Detection:** ML-based systems have successfully detected both cyber-attacks and operational failures in real-world deployments before significant damage could occur.
- **Device Fingerprinting:** AI models can automatically identify and classify IoT and ICS devices on a network simply by observing their network traffic patterns, helping to identify unauthorized or rogue devices.
- **Botnet Detection:** Models can identify the synchronized, anomalous communication patterns characteristic of IoT botnets like Mirai.

Module 5

Agentic SecOps

CTI → detections, playbooks, orchestration, MCP-style connectors

The Traditional SecOps Challenge

The Problem: Human Scale vs. Machine Scale

Traditional Security Operations Centers (SOCs) are overwhelmed. Analysts face a constant battle against a high volume of threats with limited resources.

Key Challenges:

- **Alert Fatigue:** Security tools generate thousands of alerts daily, most of which are false positives. Analysts spend more time triaging than investigating.
- **Manual Processes:** Critical tasks like threat hunting, alert enrichment, and initial response are often manual, slow, and prone to human error.
- **Skills Gap:** There is a global shortage of highly skilled security analysts.
- **Slow Response Times:** High Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR) give attackers a significant advantage.

From Anomaly Detection to Autonomous Reasoning

The introduction of AI has been a game-changer for security operations, moving beyond simple automation.

Phase 1: Early AI/ML

- Focused on **anomaly detection** by learning a baseline of "normal" behavior.
- Helped reduce noise but still required significant human interpretation.

Phase 2: The Generative AI & LLM Breakthrough

- **Large Language Models (LLMs)** introduced the ability to **understand, reason, and generate** human-like text and code.
- This enabled a paradigm shift from just *detecting* anomalies to:
  - **Understanding** the context of an alert.
  - **Reasoning** about the steps needed to investigate and respond.
  - **Generating** new detection rules and response plans on the fly.

What is Agentic SecOps?

The AI Analyst: Autonomous Agents in the SOC

An AI Agent is an autonomous system that can perceive its environment, reason, create a plan, and execute actions to achieve a specific goal. In SecOps, the goal is to detect and neutralize threats.

The Agentic OODA Loop:

1. **Observe:** Ingests data from various sources (SIEM alerts, CTI feeds, EDR logs).
2. **Orient:** Enriches and contextualizes the data. Understands the "who, what, where, and when" of an event.
3. **Decide:** Determines the severity of the threat and formulates a multi-step response plan (a dynamic playbook).
4. **Act:** Executes the plan by interacting with other security tools via APIs (e.g., isolate host, block IP).

This is not just automation; it is **autonomous decision-making**.

The Agentic Workflow: CTI → Detections

Closing the Gap from Intelligence to Action

One of the biggest challenges in SecOps is operationalizing Cyber Threat Intelligence (CTI). AI agents automate this entire process.

The Workflow:

1. **Consume CTI:** The AI agent reads unstructured CTI reports (PDFs, blog posts, security advisories) and structured feeds (STIX/TAXII).
2. **Reason & Extract:** Using Natural Language Understanding (NLU), the agent extracts key information:
    - Indicators of Compromise (IoCs): IPs, domains, file hashes.
    - Tactics, Techniques, and Procedures (TTPs) mapped to the MITRE ATT&CK framework.
3. **Generate Detections:** The agent automatically translates the extracted TTPs and IoCs into detection rules for various platforms:
    - **SIEM queries** (e.g., Splunk, Microsoft Sentinel)
    - **YARA rules** for malware scanning
    - **Sigma rules** for generic log analysis

This transforms a manual process that took hours or days into an automated task that takes seconds.

The Agentic Workflow: Detections → Playbooks

Dynamic Response Generation

When an AI-generated detection rule fires, the agent begins the investigation and response phase.

The Workflow:

1. **Alert Ingestion:** Receives an alert from a SIEM or EDR.
2. **Autonomous Enrichment:** The agent automatically gathers context:
    - **User Info:** Who is the user? What is their role?
    - **Asset Info:** What is the device? Is it a critical server?
    - **Threat Intel:** Does the IP or hash match any known threats?
3. **Dynamic Playbook Generation:** Instead of using a rigid, pre-defined playbook, the agent **reasons** about the alert and creates a custom plan.
    - **Example:** For a suspicious login, the playbook might be: "1. Query EDR for process history. 2. Check user's recent geo-logins. 3. If anomalous, disable user account."

The Agentic Workflow: Orchestration & Connectors

Executing the Plan Across the Security Ecosystem

Once a plan is created, the agent must execute it.

- **Orchestration:** The agent acts as a central coordinator, managing the sequence of actions across multiple, disparate security tools.
- **MCP-style Connectors:**
  - The agent uses a library of standardized connectors (APIs) to interact with the entire security stack, regardless of the vendor.
  - This is the "hands and feet" of the agent, allowing it to take real action.
- **Example Actions:**
  - **EDR (CrowdStrike, SentinelOne):** Isolate host, retrieve file, run memory scan.
  - **Firewall (Palo Alto, Fortinet):** Block an IP address or domain.
  - **Identity Provider (Okta, Azure AD):** Disable a user account, force MFA re-authentication.
  - **Cloud (AWS, GCP):** Revoke temporary credentials, snapshot a VM.

Significant Progress & Key Research

The Rise of the AI-Native SOC

Agentic SecOps is no longer theoretical. Major security vendors are building their next-generation platforms around this concept.

- **Real-World Impact:**
  - Dramatic reduction in alert triage time (from hours to minutes).
  - Significant improvement in MTTD and MTTR.
  - Automation of 80%+ of Tier-1 and Tier-2 analyst tasks, freeing up humans for high-level threat hunting.
- **Industry Examples & Sources:**
  - **Microsoft Security Copilot:** Integrates a powerful LLM across the entire Microsoft security suite.
  - **Palo Alto Networks Cortex XSIAM:** Billed as an "autonomous security platform" that replaces traditional SIEM.
  - **CrowdStrike Charlotte AI:** A generative AI assistant built into the Falcon platform to automate queries and investigations.
- **Key Concept:** The idea of an **"AI-native SOC"** is gaining traction, where AI is not just an add-on but the central operating system for security.

Trust, Reliability, and Security of the Agent Itself

- **Hallucinations & Reliability:** LLMs can make mistakes ("hallucinate"). An agent autonomously locking out the CEO or shutting down a critical production server based on a flawed inference is a major risk.
- **Human-in-the-Loop:** Finding the right balance is critical. Which actions can be fully automated, and which require human approval? This "human-in-the-loop" design is a key area of active research and development.
- **Security of the AI Agent:** A compromised AI agent would be the ultimate insider threat, holding the "keys to the kingdom." Securing the agent's models, prompts, and API keys is a paramount security challenge.
- **API & Tool Integration:** The vision of seamless "MCP-style connectors" is powerful but difficult to implement. Not all security tools have robust, well-documented APIs, and maintaining these integrations is a significant engineering effort.

So SoC?

- **Future Outlook:**
  - The role of the human security analyst will evolve from a hands-on "operator" to a "supervisor" or "AI fleet manager."
  - Humans will focus on:
    - Handling the complex edge cases that AI cannot.
    - Training and fine-tuning the AI models.
    - Performing high-level, creative threat hunting.
    - Designing the strategic goals for the AI security agents.

The ultimate goal is a resilient, self-healing security infrastructure where AI agents autonomously defend the enterprise.

Module 6

Security of AI Systems

Guardrails, sandboxing, red teaming, governance

Welcoming The New Attack Surface

Why Securing AI is a Fundamentally New Challenge

AI and Large Language Models (LLMs) introduce novel vulnerabilities beyond traditional software security. We are no longer just protecting code; we are protecting the model's integrity, data, and decision-making process.

The New Threat Landscape:

- **Prompt Injection:** Tricking an LLM into ignoring its original instructions and executing a malicious command. This is the #1 vulnerability according to OWASP for LLM Applications.
- **Data Poisoning:** Corrupting the model's training data to create backdoors, introduce biases, or degrade its performance.
- **Model Theft:** Stealing the proprietary weights and architecture of a trained model, which is often a company's most valuable intellectual property.
- **Insecure Output Handling:** When the AI's output (e.g., generated code) is passed directly to backend systems without sanitization, leading to vulnerabilities like XSS or SQL injection.

The First Line of Defense: Guardrails

Implementing Basic Rules and Filters

Guardrails are a set of safety controls and policies built around an AI model to control its inputs and outputs. They are the most basic and essential layer of AI security.

Types of Guardrails:

- **Input Validation:** Filtering prompts for malicious keywords, scripts, or known "jailbreak" patterns.
- **Topical Guardrails:** Restricting the model from engaging with harmful, unethical, or off-topic subjects (e.g., hate speech, illegal activities).
- **Output Sanitization:** Ensuring the model's responses do not contain sensitive information, harmful content, or executable code before being shown to the user or passed to another system.

Limitation: Guardrails are rule-based and can often be bypassed by creative or obfuscated prompts (adversarial prompting). They are necessary but not sufficient.

Containing the Damage: Sandboxing AI Agents

Limiting the "Blast Radius"

As AI agents gain the ability to use tools and execute code (**Agentic AI**), they must be treated as untrusted processes.

Sandboxing is the practice of isolating an AI agent in a restricted execution environment with limited permissions.

How It Works:

1. **Restricted Environment:** The agent runs in a container (e.g., Docker, gVisor) with no access to the host system or internal network.
2. **Limited Tool Access:** The agent is only granted API keys and permissions for the specific, low-privilege tools it needs to complete a task.
3. **Strict Monitoring:** All actions taken by the agent (API calls, code execution) are heavily logged and monitored for suspicious behavior.

The Goal: If an attacker successfully hijacks the agent via prompt injection, the sandbox ensures the damage is contained and does not spread to the wider infrastructure.

Proactive Defense: AI Red Teaming

Finding Vulnerabilities Before Attackers Do

AI Red Teaming is the process of simulating adversarial attacks to proactively discover a model's vulnerabilities, biases, and failure modes.

The Process:

- A dedicated team of humans (and now, other AIs) acts as attackers.
- They systematically try to "jailbreak" the model by crafting adversarial prompts to:
  - Bypass its guardrails.
  - Elicit harmful or biased responses.
  - Trick it into revealing confidential information from its training data.
  - Hijack its tool-using capabilities.

The findings are then used to improve the model's training, fine-tuning, and guardrails.

Using AI to Test AI

The emergence of powerful LLMs has revolutionized the red teaming process itself, moving it from a purely manual effort to a scalable, automated one.

How AI Accelerates Red Teaming:

- **Automated Jailbreak Generation:** An LLM can be prompted to generate thousands of creative and diverse adversarial prompts, testing a model's defenses far more exhaustively than a human team could.
- **Discovering Novel Attack Vectors:** By using one AI to attack another, red teams have discovered new and non-obvious ways to bypass safety filters.

Key Research & Sources:

- **"Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned" (DeepMind, 2022):** A foundational paper detailing the methods and effectiveness of using humans and models for red teaming.
- Many organizations, including Google, Meta, and OpenAI, now run continuous, automated red teaming pipelines where AI models constantly test each other for new vulnerabilities.

Establishing Order: AI Governance

Creating a Framework for Trust and Accountability

AI Governance is the framework of rules, policies, and processes for managing the risks and ensuring the responsible development and deployment of AI systems.

Key Components:

1. **Risk Management Frameworks:** Adopting structured frameworks to identify, assess, and mitigate AI-specific risks. The **NIST AI Risk Management Framework (AI RMF 1.0)** is a widely adopted standard.
2. **Model Inventory & Bill of Materials:** Maintaining a central registry of all AI models in use, their training data sources, and their known limitations.
3. **Access Control & Auditing:** Implementing strict controls on who can access, train, or deploy models. All AI-driven decisions and actions must be logged for forensic analysis.
4. **Ethical Oversight:** Establishing a review board to ensure AI systems are developed and used in an ethical, fair, and transparent manner.

Maturing from an Academic Problem to an Engineering Discipline

The security of AI is no longer just a research topic; it is a critical focus for the entire industry.

- **Standardized Frameworks:**
  - **NIST AI Risk Management Framework (AI RMF):** Provides a comprehensive guide for governing AI risks.
  - **MITRE ATLAS (Adversarial Threat Landscape for AI Systems):** A knowledge base of adversarial tactics and techniques against AI systems, modeled after the successful ATT&CK framework.
  - **OWASP Top 10 for LLM Applications:** Provides a clear, actionable list of the most critical security risks for developers building with LLMs.
- **Industry Collaboration:** Major tech companies are collaborating on AI safety research and best practices through consortiums like the **AI Safety Institute**.

The Arms Race: Defenses vs. Evolving Attacks

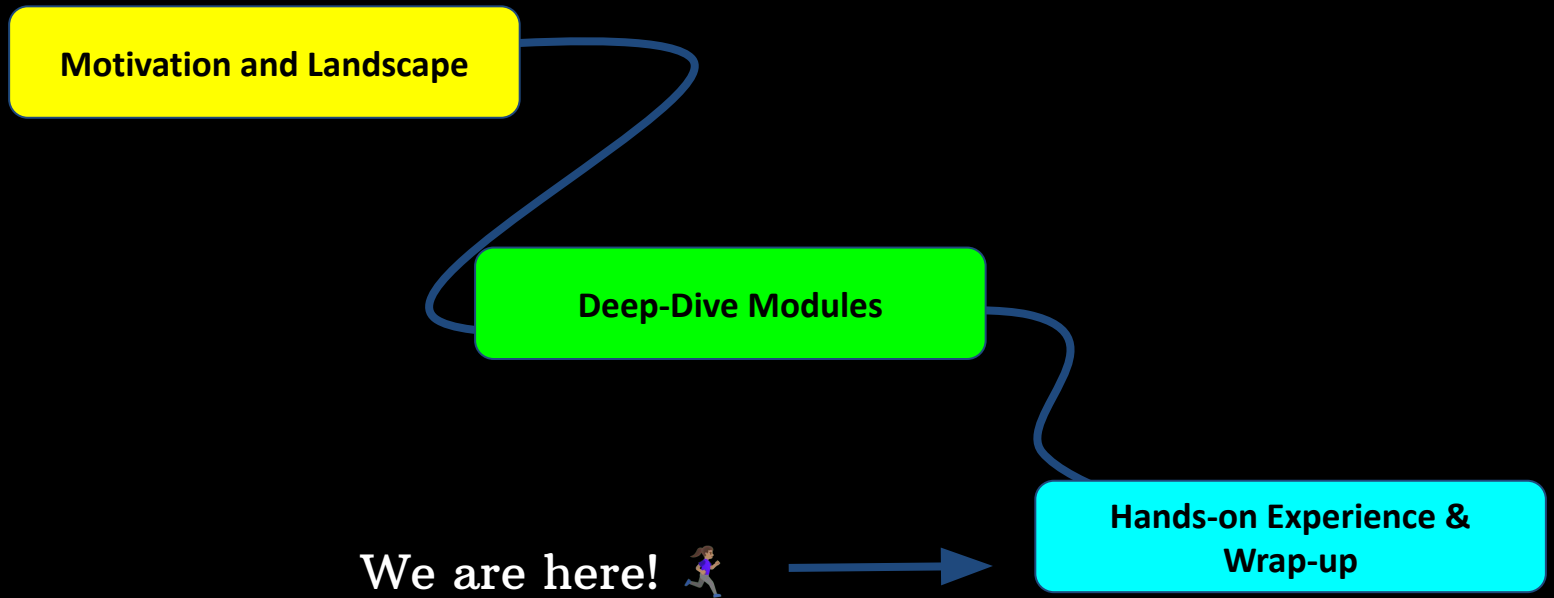
Despite significant progress, many fundamental challenges remain unsolved.

- **The Inherent Vulnerability of LLMs:** At their core, LLMs are designed to follow instructions, making them fundamentally susceptible to clever prompt injection. A perfect, universal defense may not be possible.
- **Scalability of Evaluation:** Manually red teaming every new or fine-tuned model is not scalable. Automated evaluation methods are still an active and complex area of research.
- **Model Theft and Watermarking:** While techniques exist to "watermark" a model's outputs to trace its origin, robustly protecting a model's weights from a determined attacker is extremely difficult.
- **AI Supply Chain Security:** How do you trust the pre-trained models and datasets you build upon? Research has shown that models can be trained with hidden "sleeper agent" backdoors that only activate with a specific trigger.
    - **Reference:** "Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training" (Anthropic, 2024).

Towards a "Zero Trust" Architecture for AI

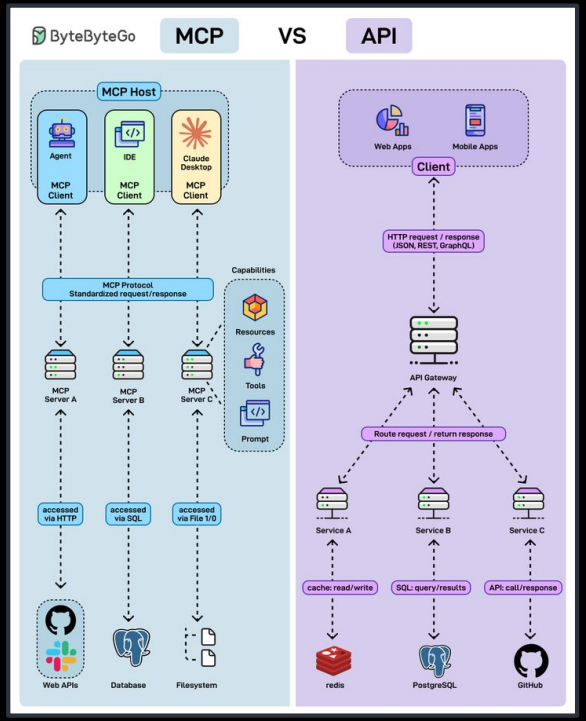
- **Summary:** Securing AI systems requires a new, multi-layered approach that combines proactive defenses (red teaming), real-time controls (guardrails, sandboxing), and a strong foundation of governance.
- **Future Outlook:**
  - **AI Security Posture Management (AI-SPM):** New tools will emerge to continuously scan, evaluate, and report on the security posture of an organization's entire fleet of AI models.
  - **Shift to a Secure AI Development Lifecycle (AI SDLC):** Security will be integrated into every stage of the model lifecycle, from data sourcing and training to deployment and monitoring.
  - **"Zero Trust" for Agents:** Every action taken by an AI agent will be treated as untrusted, requiring independent verification and operating under the principle of least privilege.
  - **Development of More Robust Models**

Workshop Agenda



MCP

- The Model Context Protocol (MCP) allows applications to provide context for LLMs in a standardized way, separating the concerns of providing context from the actual LLM interaction.
  - MCP Server: Expose resources, prompts and tools
  - MCP Client: Can connect to the MCP Server



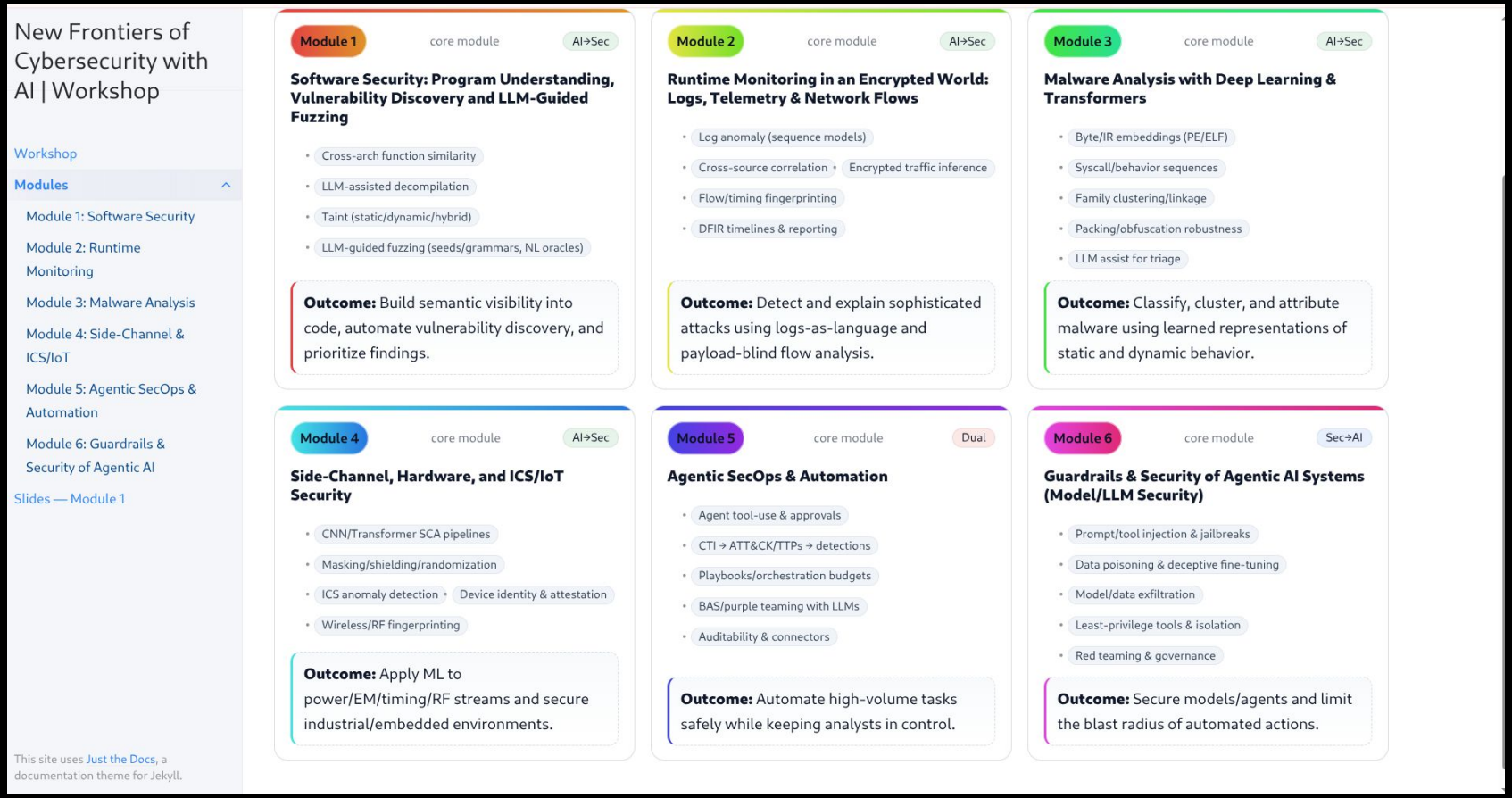
MCPGhidra

```
{
  "mcpServers": {
    "ghidra": {
      "command": "python",
      "args": [
        "/ABSOLUTE_PATH_TO/bridge_mcp_ghidra.py",
        "--ghidra-server",
        "http://127.0.0.1:8080/"
      ]
    }
  }
}
```

key takeaways!

- A lot :)

Wrap up!


New Frontiers of Cybersecurity with AI | Workshop
Workshop
Modules
Module 1: Software Security
Module 2: Runtime Monitoring
Module 3: Malware Analysis
Module 4: Side-Channel & ICS/IoT
Module 5: Agentic SecOps & Automation
Module 6: Guardrails & Security of Agentic AI
Slides — Module 1
This site uses Just the Docs, a documentation theme for Jekyll.
Module 1 core module AI→Sec
Software Security: Program Understanding, Vulnerability Discovery and LLM-Guided Fuzzing
Cross-arch function similarity
LLM-assisted decompilation
Taint (static/dynamic/hybrid)
LLM-guided fuzzing (seeds/grammars, NL oracles)
Outcome: Build semantic visibility into code, automate vulnerability discovery, and prioritize findings.
Module 2 core module AI→Sec
Runtime Monitoring in an Encrypted World: Logs, Telemetry & Network Flows
Log anomaly (sequence models)
Cross-source correlation
Encrypted traffic inference
Flow/timing fingerprinting
DFIR timelines & reporting
Outcome: Detect and explain sophisticated attacks using logs-as-language and payload-blind flow analysis.
Module 3 core module AI→Sec
Malware Analysis with Deep Learning & Transformers
Byte/IR embeddings (PE/ELF)
Syscall/behavior sequences
Family clustering/linkage
Packing/obfuscation robustness
LLM assist for triage
Outcome: Classify, cluster, and attribute malware using learned representations of static and dynamic behavior.
Module 4 core module AI→Sec
Side-Channel, Hardware, and ICS/IoT Security
CNN/Transformer SCA pipelines
Masking/shielding/randomization
ICS anomaly detection
Device identity & attestation
Wireless/RF fingerprinting
Outcome: Apply ML to power/EM/timing/RF streams and secure industrial/embedded environments.
Module 5 core module Dual
Agentic SecOps & Automation
Agent tool-use & approvals
CTI → ATT&CK/TTPs → detections
Playbooks/orchestration budgets
BAS/purple teaming with LLMs
Auditability & connectors
Outcome: Automate high-volume tasks safely while keeping analysts in control.
Module 6 core module Sec→AI
Guardrails & Security of Agentic AI Systems (Model/LLM Security)
Prompt/tool injection & jailbreaks
Data poisoning & deceptive fine-tuning
Model/data exfiltration
Least-privilege tools & isolation
Red teaming & governance
Outcome: Secure models/agents and limit the blast radius of automated actions.

Wrap up!

All materials will be available soon (promise soon🙂) at:
ret2ai.github.io

Stay in touch ✉️:
s.salimi@sharif.edu