Toward Formal Verification of Resource-Constrained Cyber Strategies across Macro, Meso, and Micro Scales
Work-in-Progress

Jonathan Goohs Jr. • Adam Pease

Naval Postgraduate School • Computer Science Department

Hot Topics in the Science of Security (HOTSOS) • April 14–16, 2026

Problem – Solution – Technical Approach - Discussion

The Problem: You Can't Patch Everything

- **Gordon-Loeb (2002):**
  - Optimal cybersecurity investment ≤ 37% of expected loss per vulnerability

- Defenders operate under hard budget and manpower constraints
  - **Not every vulnerability can be patched**

- **Existing tools answer what to prioritize:** CVSS risk scores, anomaly detection, compliance checklists

- **This work answers:**
  - **Is your chosen set of patches formally valid given all your constraints?**

The Problem: Classification ≠ Verification

Cyber defenders select strategies under resource constraints (budget, personnel, time windows) while adversaries do the same.

Existing approaches validate but cannot verify:

- Risk scoring (CVSS, FAIR): assigns numeric scores to individual vulnerabilities
- Anomaly detection: reactive, requires baseline on a system, and defeated by living-off-the-land techniques that blend into normal system activity
- Compliance checklists (NIST CSF, CMMC): checks presence of controls, not feasibility of combinations
- Game-theoretic optimization: computes optimal strategies but trusts the solver's numeric output

None produce a formal logical proof that a combination of defensive actions satisfies all meso-layer organizational constraints simultaneously.

The Problem: Equifax and CrowdStrike Breaches

Equifax 2017: Meso-Layer Failure

- The patch for CVE-2017-5638 (Apache Struts) was publicly available March 7, 2017.
- Equifax's scanner missed it, the breach ran undetected for 78 days.
- Result: 143 days from patch availability to breach discovery; 147 million records exposed.
  - We model this as:
    - **`patchRemediates(patch, vulnerability)`** = "this patch fixes this vulnerability”
    - **`holdsDuring(TimePosition, Formula)`** = a formula is true within a specific time window
      - The remediation must be in effect during the compliance window, in this case, when a vulnerability is known

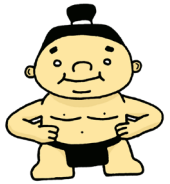
Source: U.S. House Committee on Oversight and Government Reform, 2018

CrowdStrike 2024: Pre-Condition Verification Failure

- A content configuration update was deployed to production without passing test environment validation.
- Multiple contributing factors: missing validation steps, inadequate test coverage, no staged rollout.
- Constraint our framework is designed to represent : deployment to **`ProductionEnvironment`** requires preceding **`programValidated`** in **`TestEnvironment.`**
  - Scope: we model the pre-condition violation, not the full organizational failure chain.

Source: CrowdStrike External Technical Root Cause Analysis, Channel File 291, Aug 6 2024

Technical Solution: Ontology



"A Dictionary for Computers to Read"

- Not just labels – computable definitions are needed
- Most things called ontologies are just labels and relationships
- A powerful logical language allows for capturing knowledge in a general and reusable way
  - Use logic more expressive than first order logic – modal and higher order logic

- **Suggested Upper Merged Ontology (SUMO) -** 25 years of formal ontology content development and an open-source tool set for debugging, logical inference, and deployment

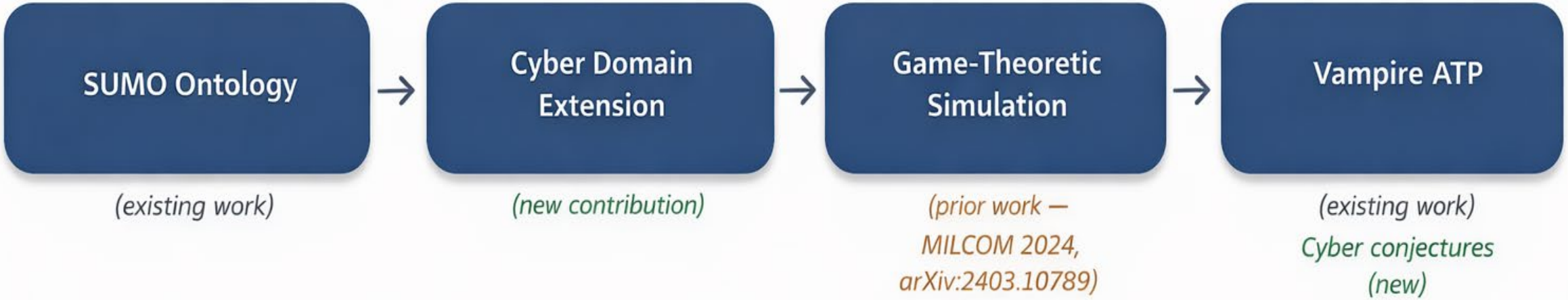
- **The cyber domain ontology extension adds domain-specific knowledge representation**

- **What SUMO already provides (existing work):**
- ~25,000 concepts, 100,000 hand-written logical formulas, ~7,800 rules across all domains
- Integration with modern first- and higher-order logic theorem provers: Vampire, Eprover, LEO-III …
  - Formal **explanation** of conclusions as mathematical proofs
- **Existing relevant concepts with definitions:**
  - **`holdsDuring`**, **`BeginFn`**/**`EndFn`** (temporal reasoning), **`ComputerProcess`**, **`ComputerNetwork`**

`https://www.ontologyportal.org`

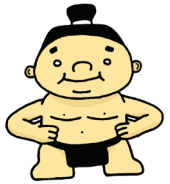
Technical Approach: Three-Phase Verification Pipeline

Pipeline Integration
(new work)



Key insight: A strategy that is game-theoretically optimal in Phase II may still be rejected in Phase III if it violates a meso-layer organizational constraint. These phases answer different questions.

Phase I: Building the Cyber Knowledge Base

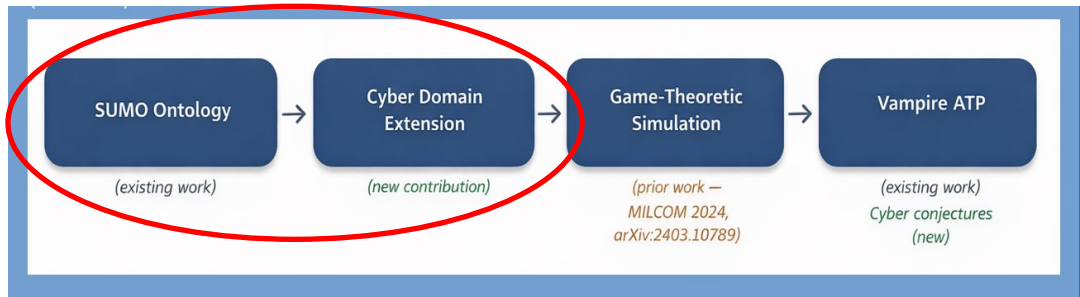


What cyber domain ontology extension adds (our contribution): 100+ terms, 200+ inference rules

- New concepts: **`SoftwareBug`**, **`Vulnerability`**, **`CyberExploit`**, **`CyberAttack`**, **`SoftwarePatch`**, **`DefensiveStrategy`**, **`patchCost`**, **`exploitCost`**, **`hazardScore`**, **`computerCapacity`**, **`bugInProgram`**
- Behavioral rules: causal chains (bug to vulnerability to exploit), temporal state transitions, constraint axioms for budget and maintenance windows

Update from WiP Submission - Knapsack axioms (previously pseudocode) are now written as formulas in logic

- AdversarialKnapsack class, operationalBudget predicate, knapsack constraint rule, and the dueling interaction rule (written; under final validation)
- Formal definition of **`CyberExploit`** success: observable state change vs. SUMO Failure attributes



Phase II: Knapsack Progression (Textbook → Adversarial)

Cyber resource allocation has a natural mathematical structure: the knapsack problem.

- We build from the textbook case to the adversarial case used in this work.
- Adversarial Planning as a competitive bin-packing problem

Weighted Knapsack: one player, one budget

- max $\Sigma\ \theta_i\mu_i$ subject to $\Sigma\ \theta_i w_i \le W$, $\theta_i \in \{0,1\}$
  - Select N items with weights w_i and hazard values μ_i to maximize value under budget W.

Dueling Knapsack: two players, shared vulnerabilities — (prior work)

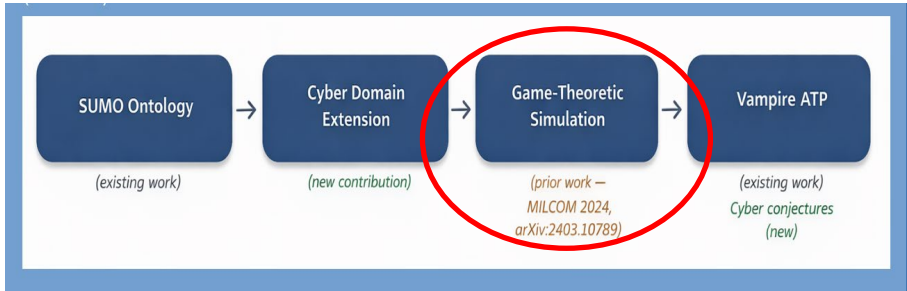
- Attacker: max $\Sigma\ \xi_i(1-\zeta_i)\mu_i$ s.t. $\Sigma\ \xi_i w_i^a \le W_a$
- Defender: max $\Sigma\ \zeta_i\xi_i\mu_i$ s.t. $\Sigma\ \zeta_i w_i^d \le W^d$
  - The $(1-\zeta_i)$ term: if the defender patches vulnerability i, the attacker gains nothing from exploiting it.
  - This is the formal encoding of the defender-attacker interaction.

Adversarial Knapsack: a network of interacting knapsack problems — (prior work)

- Multiple nodes, asymmetric costs, interacting vulnerability sets across the network.
- Generalizes the dueling formulation to full network-scale resource competition.

Why this model?

- Finite budgets. Discrete binary choices (patch or exploit). Shared vulnerabilities.



Phase II: Game-Theoretic Simulation

- **The simulation generates the candidate strategies that Phase III will formally verify.**
- **Simulation engine complete (prior published work: MILCOM 2024, arXiv:2403.10789).**

- **Connect Phase II output to Vampire: under development.**
  Prior work: Goohs et al. (MILCOM 2024); Goohs et al. (arXiv:2403.10789)

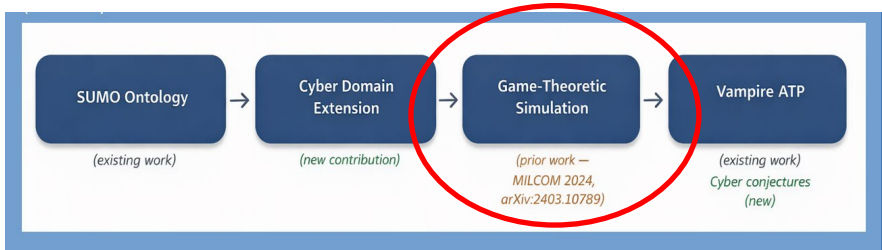
Method: Fictitious Play (iterated best-response)

- Each player observes the opponent's prior strategy and solves their own weighted knapsack.
- Reduces 2^n joint decision variables to N binary variables per iteration.
- Iterate until convergence; output is an approximate Nash equilibrium.

Simulation parameters (validated in prior work):

- Networks of 100 machines, 70 vulnerabilities (0-10 per node)
- Hazard scores uniform [6, 100]; patch costs [0, 100]; exploit costs [100, 700]
- Budget per player: 800; tested at networks with thousands of nodes

Intended output: candidate strategies expressed as logical assertions for input to Phase III



Phase III: Formal Verification (Vampire theorem prover)

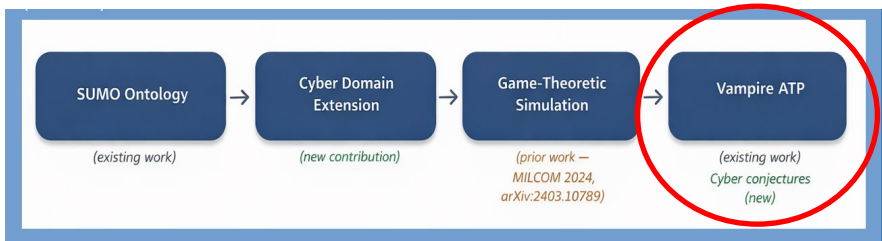
The pipeline filters candidate strategies by proof-by-refutation enumeration.

Proof-by-refutation (four steps):

1. Load Cyber constraint axioms: budget, time, capacity
3. Conjecture: Does a constraint violation exist?
4. Vampire Automated Theorem Prover provides mathematical proof of constraint violations

Key advantage: the output is a logical proof, not a confidence score

- Every rejected strategy comes with an inspectable chain of logical steps showing the violated constraints



Concrete Example: Model, Strategy, Scenario

Key terms before the math:

1. Model: the network state (hosts, vulnerabilities, resource costs)
2. Strategy: a binary decision vector $\theta_i \in \{0,1\}$ per vulnerability (1=patch, 0=skip)
3. Scenario: a strategy paired with an adversary's choices and a time step

Model: 2 hosts, 4 vulnerabilities

Host A: vulnerability 1 (μ_1=9, w^d=40), vulnerability 2 (μ_2=6, w^d=30)
Host B: vulnerability 3 (μ_3=8, w^d=50), vulnerability 4 (μ_4=3, w^d=20)
Defender budget W^d = 100. Maintenance window: Tuesday 0200-0600.

Strategy: θ = [1, 1, 1, 0] (patch vulns 1, 2, 3; skip vuln 4)
Total patch cost: 40 + 30 + 50 = 120 > W^d = 100
This strategy violates the budget constraint.

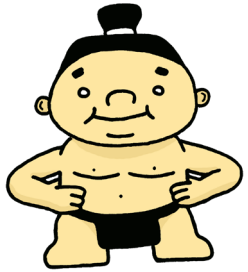
Scenario: attacker selects ξ = [0, 0, 1, 1] (exploit vulnerabilities 3 and 4)

Vampire conjecture (simplified for presentation):

Ask: (violatedKnapsack DefenderStrategy1)
Vampire: proof found. Patch cost 120 exceeds W^d = 100.
Strategy REJECTED. Proof trace identifies the violated budget axiom.

Thank You

Questions and Discussion



Jonathan Goohs Jr. • jonathan.goohs@nps.edu

Adam Pease • adam.pease@nps.edu

SUMO: https://www.ontologyportal.org

github.com/ontologyportal/sumo

SigmaKEE (SUMO development environment): github.com/ontologyportal/sigmakee