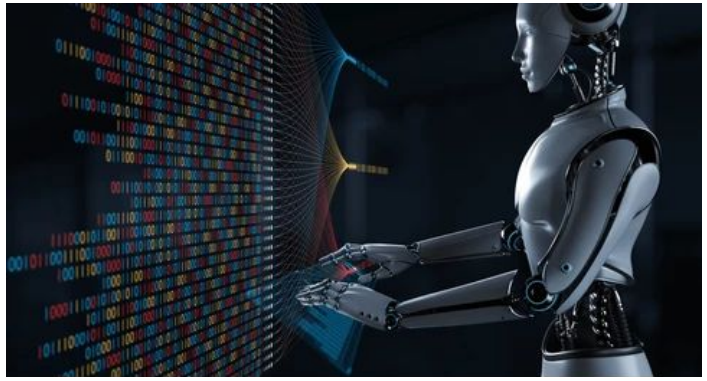
UNIVERSITY OF NORTH CAROLINA CHARLOTTE

AutoPwn: Automatic Code-Reuse Exploit Generation Framework with Agentic AI

Kaleb Bacztub[1,3], Dylan Christensen[1], Dr. Arun Ravindran[2], Dr. Meera Sridhar[1],

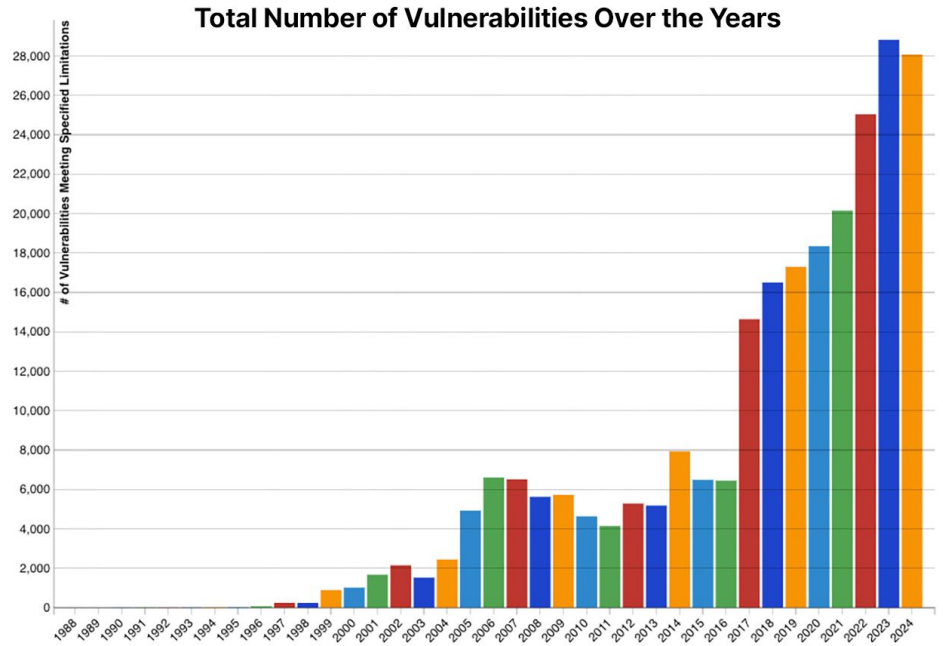
[1]Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, NC, USA
[2]Department of Electrical and Computer Engineering, University of North Carolina at Charlotte, Charlotte, NC, USA
[3]Current affiliation: Purdue Indianapolis - Cybersecurity BS/MS (work conducted as REU student at UNC Charlotte)
*Work supported in part by NSF-CNS #2244424, UNC Charlotte and UNC System



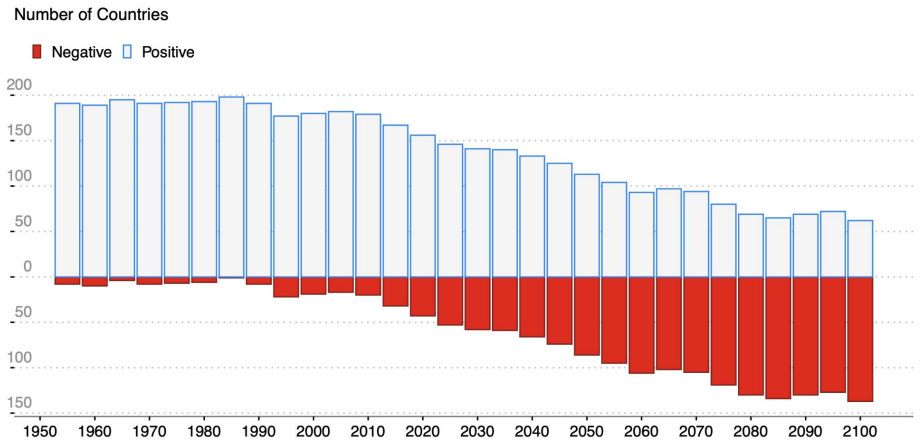
Section 1: Introduction & Motivation

The Challenge: IoT & Exploit Generation

- Context: IoT/CPS devices are ubiquitous but often lack modern defenses.
- The Problem: Vulnerability assessment is manual, tedious, and requires high expertise.
- Existing Automation: AEG provides some automation, but its capabilities are limited.
- The Gap: Traditional AEG struggles with logic and context.



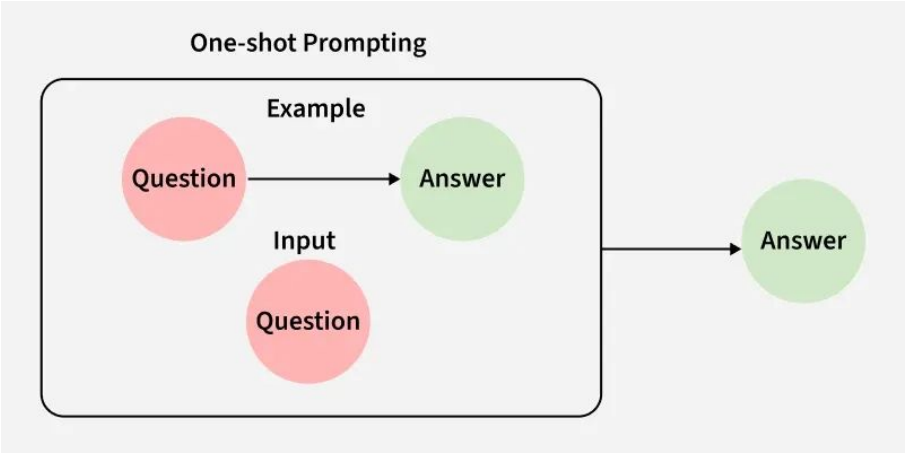
Growth of Vulnerabilities over time

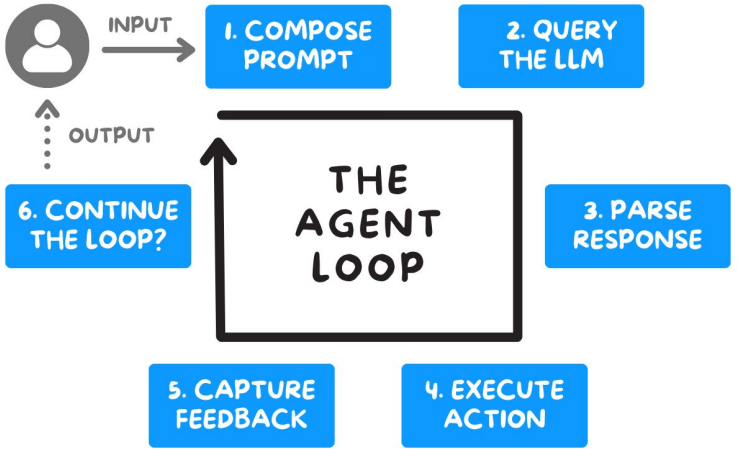


Decline of Workforce Participation

Enter Agentic AI

- Agentic AI = LLMs that reason, plan, use tools, and iterate.
- Moves beyond one-shot prompting → continuous feedback loops.
- AutoPwn: an orchestrated, multi-phase exploitation pipeline powered by an LLM agent.
- Enables autonomous strategy selection & adaptation.



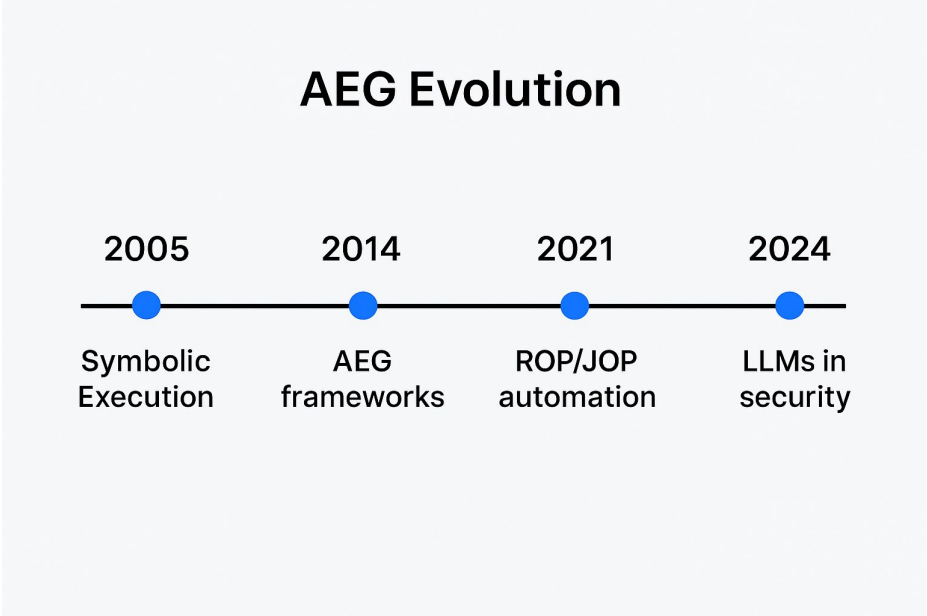


Contributions

- AutoPwn: a closed-loop agentic framework for code-reuse exploit generation.
- Orchestrates recon → crash → offset → strategy → payload → execution.
- Defense-aware pivots across No Protections → NX → ASLR.
- Key Result: Demonstration that agentic AI can autonomously perform AEG, including a full 2-stage ASLR bypass (ret2plt → dynamic ret2libc).

Section 2: Background & Related Work: Automated Exploit Generation (AEG) Landscape

- Traditional AEG: symbolic execution, fuzzing, constraint solving.
- Good at finding crashes → weak at building full ROP chains.
- Modern work adds ROP chain assembly, summaries, heuristics, genetic search.
- Still brittle, lacks global reasoning & adaptation.

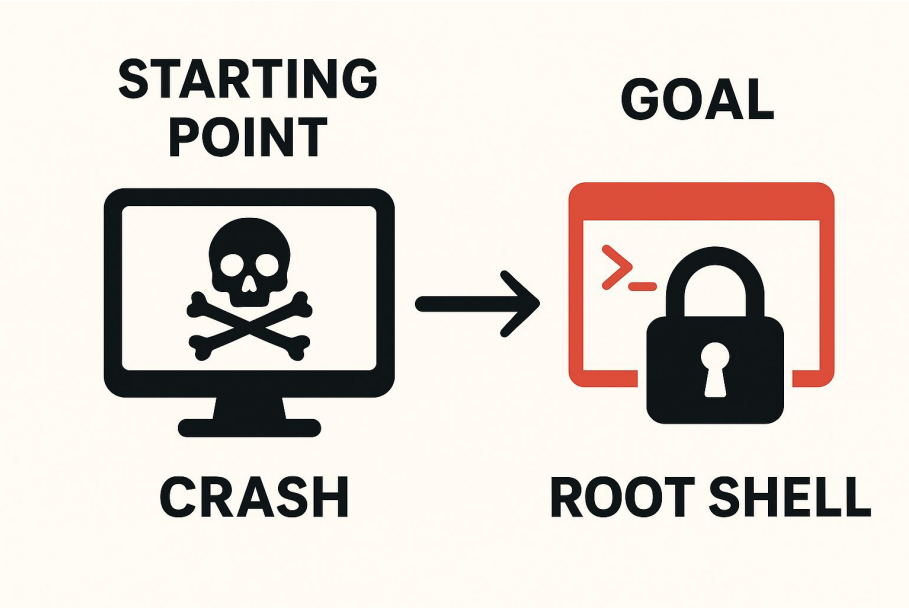


Addressing the "Cyber Grand Challenge"

- DARPA CGC focused on defense: autopatching, machine-speed reasoning.
- AutoPwn focuses on offense: generating realistic exploits.
- Complementary: AI attacker used to strengthen AI defenders.
- Builds a foundation toward automated attacker–defender ecosystems.

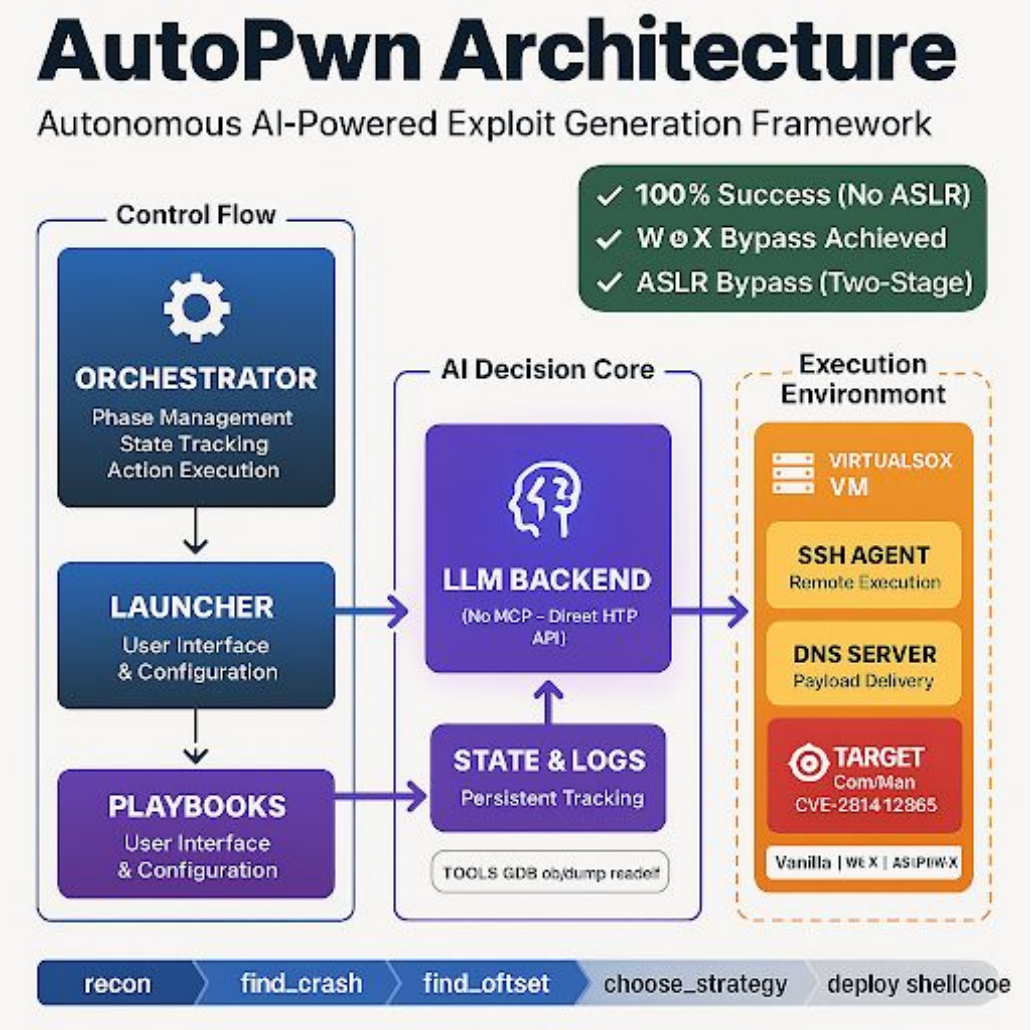
Scope & Threat Model

- Input: a known crash or PoC, not the CVE number.
- Attacker = remote, non-root access.
- Agent cannot rely on local debugging on the target.
- Goal: move from Crash → Root Shell autonomously.



Section 3: System Design
AutoPwn Architecture

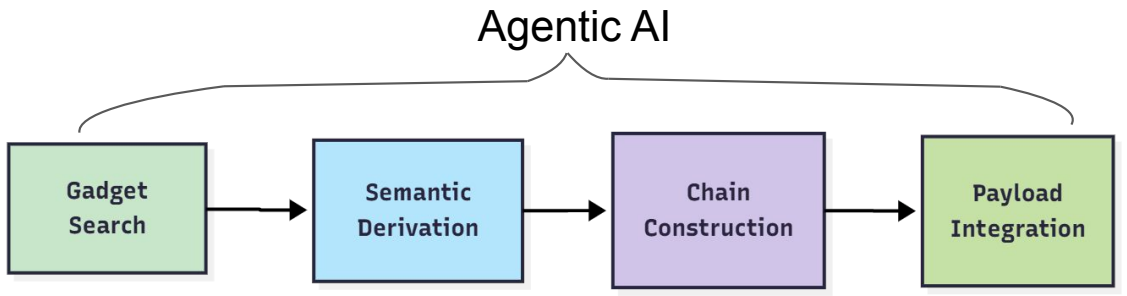
Components:
1. Orchestrator: Python-based controller.
2. LLM Agent: The "Brain" (Reasoning).
3. Tool Layer (MCP): The "Hands" (Execution).
4. Digital Twin: Virtualized target environment.



The Agentic Workflow

Phase-Based Execution:

1. Reconnaissance.
2. Strategy Selection.
3. Gadget Discovery.
4. Payload Synthesis.



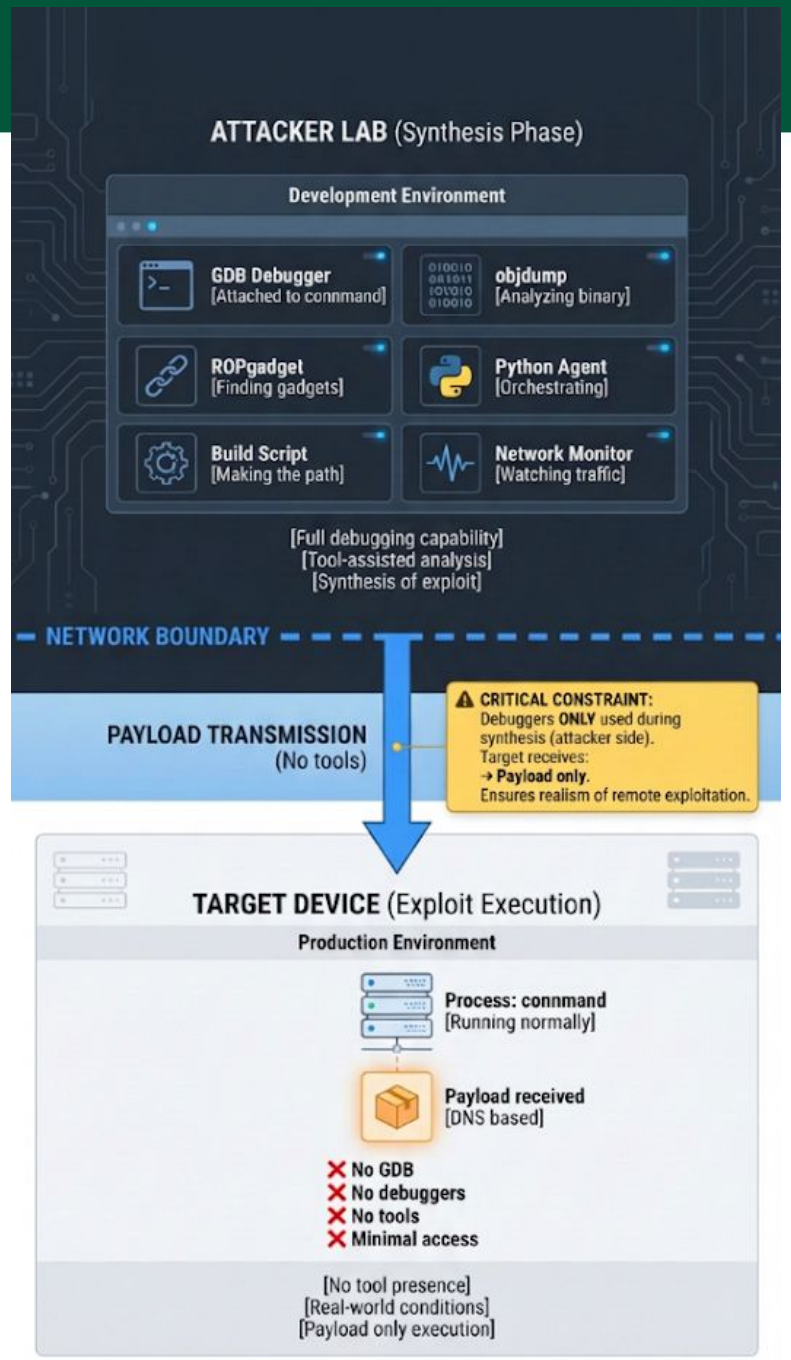
Tool Integration

- Tools: GDB, checksec, ROPgadget, Python utilities.
- ROPgadget parsed directly by LLM (no angrop).
- Agent uses raw text output → semantic reasoning over gadgets.
- Full automation through MCP adapters.

```
140  ---
141  [AI PARSER] Extracted result: {'address': '0x08053940'}
142  [AGENT] Running ROPgadget to find ROP gadgets for analysis...
143  [AGENT] Executing: ROPgadget --binary /usr/sbin/connmand 2>/dev/null | head -100 || echo ''
144  [AGENT] ROPgadget output received (6729 characters)
145  [AGENT] Falling back to AI parser for ROP gadget extraction...
146  [AGENT] ROPgadget output:
147  Gadgets information
148  ============================================================
149  0x0810ac49 : aaa ; add al, ch ; ret
150  0x0804ae69 : aaa ; add byte ptr [eax], al ; xchg eax, ecx ; ret
151  0x0810f1b5 : aaa ; add dword ptr [eax], eax ; ret 0xfbd1
152  0x08110ced : aaa ; cld ; jmp esi
153  0x080701b2 : aaa ; jmp 0x80701cf
154  0x08083655 : aaa ; jmp 0x8083655
155  0x08068255 : aaa ; lcall 0x9010:0xc483fffe ; leave ; ret
156  0x0808a6e0 : aaa ; mov eax, 0x80e8052 ; jmp 0x808a71e
157  0x0808a9c1 : aaa ; mov eax, 0x80e815d ; jmp 0x808a9ff
158  0x080c04dd : aaa ; mov eax, 0xffffff8d ; jmp 0x80c051b
159  0x08072145 : aaa ; or esi, edi ; inc dword ptr [ebx + 0x30a310c4] ; ret
160  0x08111a88 : aaa ; outsb dx, byte ptr [esi] ; cld ; jmp dword ptr [edx]
161  0x0807a370 : aaa ; push cs ; or byte ptr [eax + 0x59], ch ; cmp al, 0xe ; or al, ch ; retf
162  0x080771...
163  [AI PARSER] Task: Analyze the ROPgadget output and find useful ROP gadgets. Look for gadgets that can be used in exploit chains, such as 'pop reg; ret', 'mov [reg], reg; ret', or
164  [AI PARSER] Input text length: 6729 characters
165  [AI PARSER] Input text (first 500 chars):
166  ---
```

Realism vs. Simulation

- Debuggers only used in the attacker VM, never on the target.
- Simulates a real-world remote exploitation scenario.
- Payload is the only artifact delivered to the target.
- Ensures threat-model correctness.



Example Trace: Agent Reasoning

- Orchestrator logs show JSON-only tool calls.
- Example: Agent computes offset from crash at 0x41414141.
- Fully interpretable reasoning loop → transparent debugging.
- Demonstrates how agent switches tools based on context.

```
---
[AI PARSER] Extracted result: {'gadgets': [{'address': '0x0805ef6c', 'instruction': 'aad 0xc9 ; ret', 'useful': True}, {'address': '0x080b49d4', 'instruction': 'aam 0xc9 ;
ret', 'useful': True}, {'address': '0x08104f90', 'instruction': 'aam 0x95 ; add byte ptr [eax], al ; ret', 'useful': True}, {'address': '0x080bec3b', 'instruction': 'aad 0x90
; leave ; ret', 'useful': True}, {'address': '0x080b2a08', 'instruction': 'aam 0x90 ; leave ; ret', 'useful': True}, {'address': '0x0809acd7', 'instruction': 'aad 0x11 ; or
byte ptr [eax], bh ; leave ; ret', 'useful': True}, {'address': '0x0809334f', 'instruction': 'aam 0x11 ; or byte ptr [ecx], al ; nop ; pop ebp ; ret', 'useful': True},
{'address': '0x080f7d59', 'instruction': 'aad 0 ; add byte ptr [esi + esi*4 - 8], dh ; jmp esp', 'useful': True}, {'address': '0x080f7d19', 'instruction': 'aam 0 ; add byte
ptr [ebp + esi*4 - 8], dh ; jmp esp', 'useful': True}, {'address': '0x080f7f1c', 'instruction': 'aad 0xd4 ; clc ; jmp esp', 'useful': True}, {'address': '0x08109778',
'instruction': 'aam 0xa2 ; stc ; jmp esp', 'useful': True}]}
[AGENT] AI parser successfully extracted 11 ROP gadgets
[AGENT]   Gadget 1: 0x0805ef6c - aad 0xc9 ; ret
[AGENT]   Gadget 2: 0x080b49d4 - aam 0xc9 ; ret
[AGENT]   Gadget 3: 0x08104f90 - aam 0x95 ; add byte ptr [eax], al ; ret
[AGENT]   Gadget 4: 0x080bec3b - aad 0x90 ; leave ; ret
[AGENT]   Gadget 5: 0x080b2a08 - aam 0x90 ; leave ; ret
[AGENT] Executing: readelf -l /usr/sbin/connmand | grep -A 1 '\.rodata' || echo ''
[AGENT] Executing: strings -t x /usr/sbin/connmand | grep '/bin/sh' | head -1
[AGENT] Executing: objdump -s -j .rodata /usr/sbin/connmand | grep -A 5 '/bin/sh' || echo ''
[AI PARSER] Task: find the hexadecimal address of the '/bin/sh' string in the binary. For non-PIE binaries, base is typically 0x400000. Respond with JSON like {'address':
'0x...'}
[AI PARSER] Input text length: 0 characters
[AI PARSER] Input text:
```

Section 4: Evaluation & Results

Experimental Setup

- Target: ConnMan (IoT Connectivity Daemon).
- Vulnerability: CVE-2017-12865 (Stack Buffer Overflow
- The model was not give the CVE Number

- Configurations:
  1. No Protection.
  2. W⊕X (NX).
  3. ASLR + W⊕X (NOPIE).

Results Overview

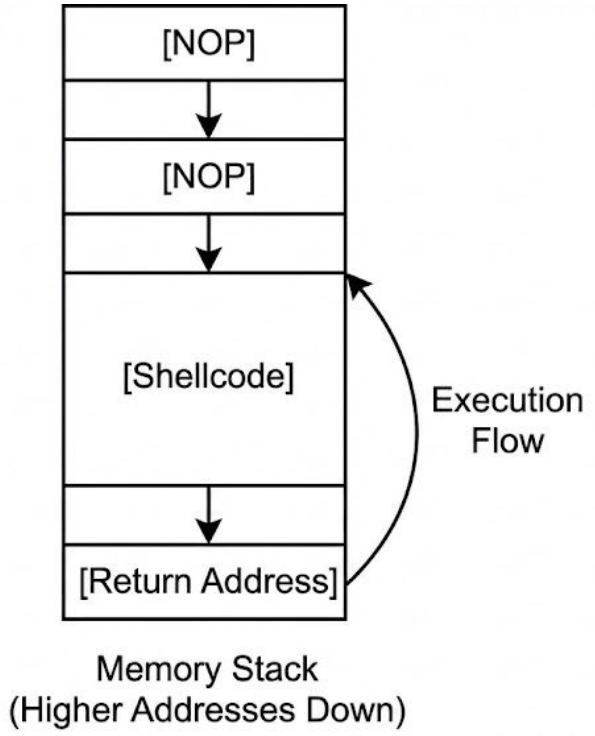
Success/Exploitation across all three configurations.

- Configurations:
  1. No Protection.
  2. W⊕X (NX).
  3. ASLR + W⊕X (NOPIE).

No-Security	W^X Enabled	ASLR + W^X Enabled
✓	✓	✓

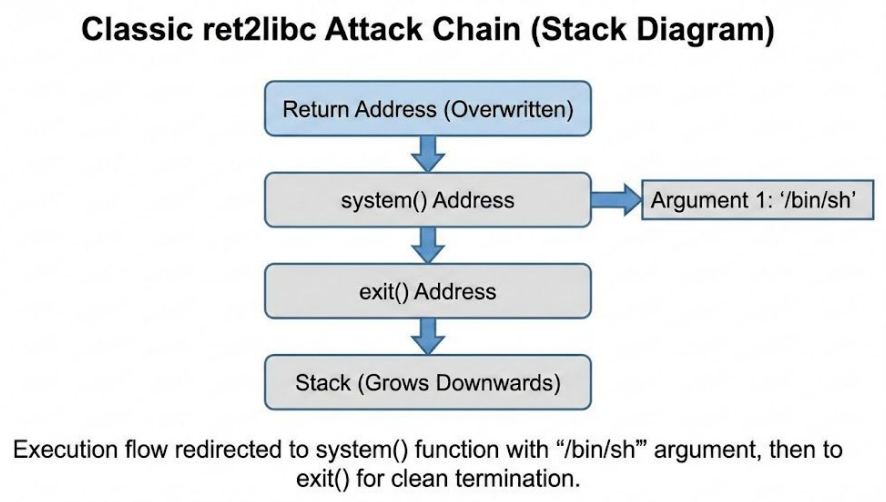
Case 1: No Protections

- Stack executable.
- Agent crafts shellcode + NOP sled.
- Computes offset → injects shellcode reliably.
- Achieves root shell.



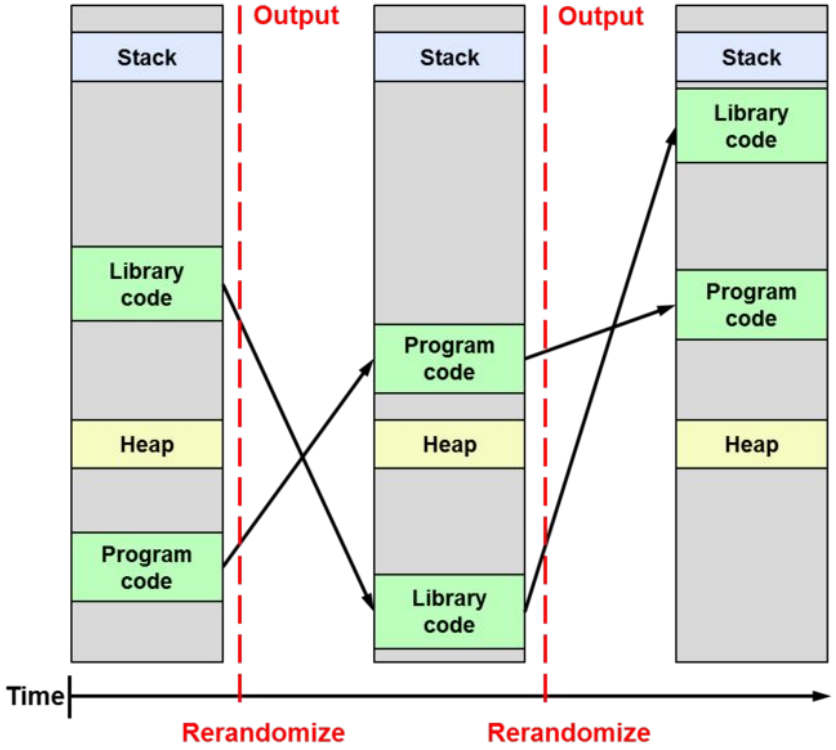
Case 2: W⊕X (NX Enabled)

- Agent detects non-executable stack via recon.
- Pivots to ret2libc automatically.
- Chains: system("/bin/sh") → exit().
- Fully autonomous payload generation.



Case 3: The Challenge of ASLR

- Address randomization breaks hardcoded ROP.
- Previous models and setups failed here.
- AutoPwn adopts a 2-stage leak-and-exploit strategy.
- Uses Global Offset Table (GOT) → leak libc → compute base.
- This target uses ASLR (NOPIE): libc is randomized, but the binary base is fixed.



ASLR Bypass - Stage 1 (Leakage)

- ret2plt: call write@plt to leak exit@got.
- Leak captured from system logs (journalctl).
- Works under remote-attacker constraints.
- Produces runtime libc pointer.
- NOPIE simplifies the attack: only the libc base must be leaked.

ASLR Bypass - Stage 2 (Calculation & Pwn)

- Compute libc base from leaked address.
- Derive system(), exit(), “/bin/sh” at runtime.
- Assemble ret2libc dynamically.
- Final payload delivers root shell.

Verifying Success

- Shell indicators: uid=0, #, or interactive shell tokens.
- Exit code inspection confirms control flow.
- Pipeline re-tries if indicators missing.
- Clear evidence of root execution.

```
process 12437 is executing new program: /bin
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Addressing "Generalizability”

- Only one target?
- AutoPwn is a framework, not a single exploit.
- Logic is general:
  Recon → Leak → Calc → ROP.

Section 5: Discussion & Conclusion

Limitations

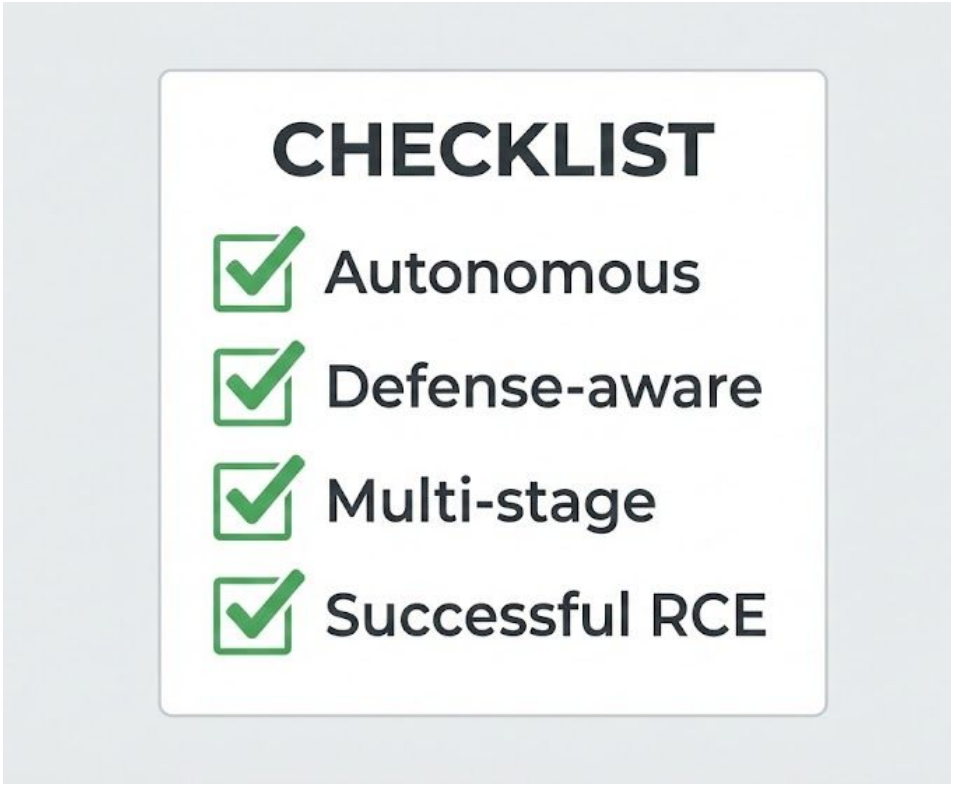
- Does not handle strict bad-character constraints.
- Current design focuses on stack bugs (not heap).
- ROPgadget output parsing is noisy for LLMs.
- Scaling requires semantic gadget indexing.
- Full PIE-enabled binaries remain unsupported; PIE randomizes the binary base and requires an additional leak.

Future Work

- Vector-DB gadget indexing for semantic search.
- Expanding target suite (multiple IoT binaries).
  - As well as expanding to add more realistic protections
- Automated attacker–defender loop experiments.
- Integrating multiple cooperating LLM agents.

Conclusion

- AutoPwn demonstrates autonomous exploit generation.
- Effective across No Protections → NX → ASLR.
- First AI-driven 2-stage ASLR bypass in this setting.
- Positions AI as a tool for defensive stress-testing.



Thank you

Any Questions?

Paper

Kaleb Bacztub; kbacztu@purdue.edu

Dylan Christensen; rchris25@charlotte.edu

Dr. Arun Ravindran; aravindr@charlotte.edu

Dr. Meera Sridhar; msridhar@charlotte.edu

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

- Prepared to answer questions specifically about the prompt engineering and JSON