Software Fuzzing Workflow & Tool for Slowing Adversarial Attack Tools

Adam Brannon, Noah Sickels, William Lochte

MENTOR: Dr. David Coe

CPE495 COMPUTER ENGINEERING DESIGN I
CPE 488: CYBERSECURITY ENGINEERING CAPSTONE I

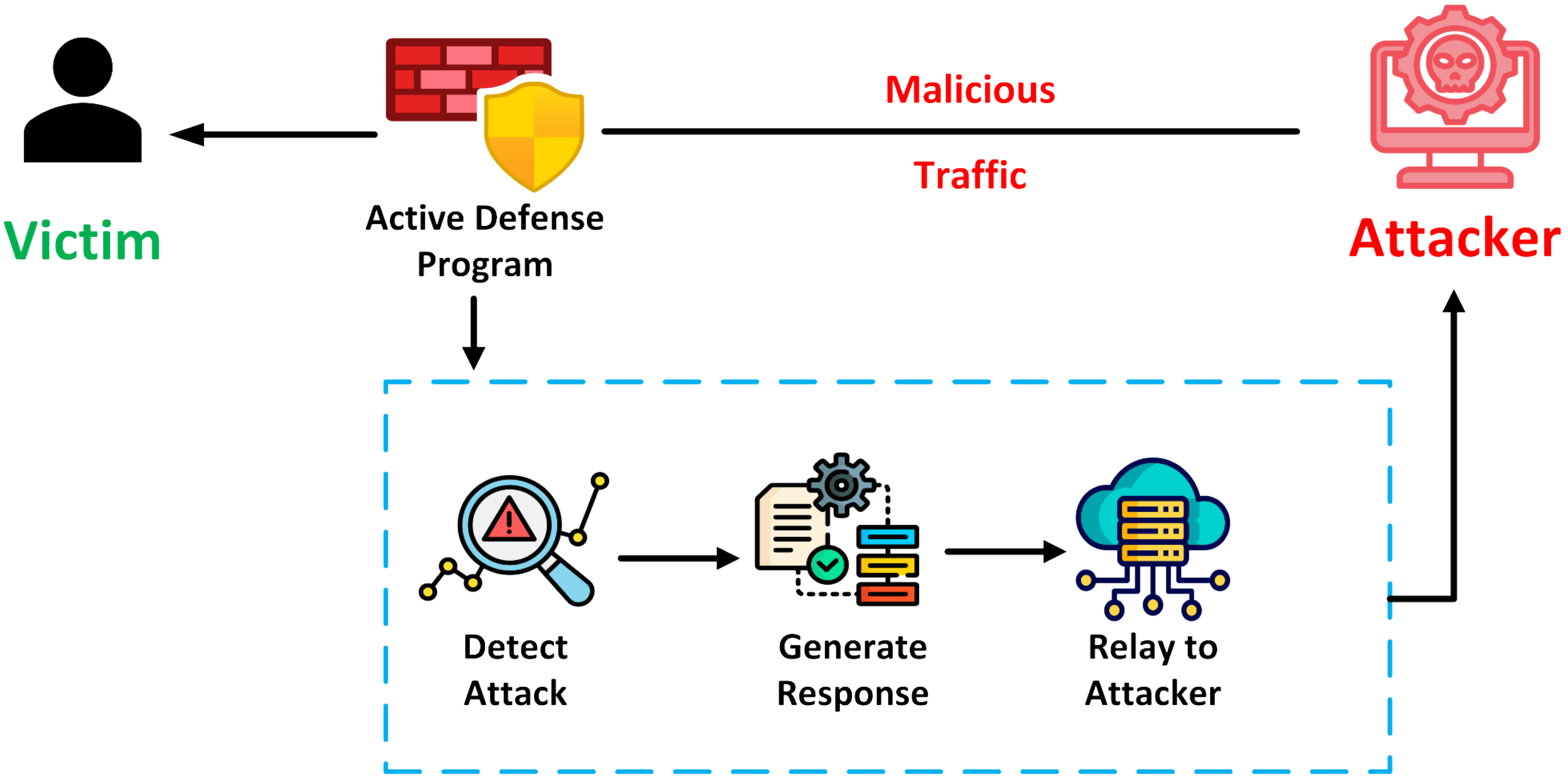
ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF ALABAMA IN HUNTSVILLE

email: ajb0093@uah.edu, ncs0016@uah.edu, wpl0001@uah.edu

Stalling Attacker’s Tools

- Attackers today use tools that are extremely efficient, often capable of sending hundreds if not thousands of requests per minute or send packets infrequently over multiple days to avoid detection.
- Therefore, it is important to slow down or stop attackers. An active defense mechanism can do this by neutralizing an attacking application, wasting an attacker’s time and resources.
  - Useful for all businesses with information to protect.
- It will also be helpful to perform a counterattack using binary exploitation to install malware on the attacker’s computer in cases where it is legal to do so.
  - Could be useful in a nation-state scenario.

Charger Active Defense Overview



Victim

Active Defense Program

Malicious Traffic

Attacker

Detect Attack

Generate Response

Relay to Attacker

Innovation

- Push the industry forward by finding a new exploitable vulnerability for a cyber attack tool.
- Fuzzing is a well documented technique for finding vulnerabilities in software but it is not usually placed on the attack tools themselves to find exploitable vulnerabilities as an active defence mechanism.
- Implementation of an adaptive real-time defense strategy, focused on proactive defense instead of reactive.

Marketing Requirements

1. Workflow should prioritize attack tools that are more likely to have vulnerabilities
2. Workflow should fuzz test attack tools for vulnerabilities
3. Proof of concept program should mitigate or slow an attacking tool
4. Proof of concept program should provide an active defense response
5. Proof of concept should show that the workflow functions
6. Proof of concept should use AI/LLM to generate responses capable of neutralizing or slowing the attacking application
7. Findings should be placed into a IEEE/ACM style paper

Engineering Requirements

Marketing Requirements	**Engineering Requirements**	**Justification**
1	Workflow should prioritize tools that are at least 25% C/C++ or have memory errors during static analysis	C/C++ are memory unsafe languages, which mean they are more likely to have memory vulnerabilities. We allow static analysis as an alternative to support other languages
1,2	Workflow should choose a static analysis tool	Static analysis will help with prioritizing attack tools
2,4,5	Workflow should find at least 1 exploitable vulnerability within attacking application	To be able to stop or slow the attacking application
3,4,6	Proof of concept should be able to crash or hang the attacking application for at least 1 second	A 1 second hang will greatly reduce the efficiency of the attacking application

See Appendix A for full list

Market & Competition

Market and competition primarily consists of two main categories:

- **IDS/IPS Solutions**
  - Snort3, *Cisco*
    - Efficient, Robust, Free/Open-Source
  - Suricata, *OISF*
    - Easy to use, Customizable, Direct rule feeds, Partially Free/Open-Source

- **EDR Solutions**
  - CrowdStrike Falcon, *CrowdStrike*
    - Broad OS support, 24/7 Support Service, $99-$185/Device per month
  - SentinelOne Singularity Complete, *SentinelOne*
    - Real-time monitoring, reactive threat isolation, $36/VM per month

Comparative Advantages over Market Competitors

- **Snort3**, *Cisco*
  - Can be easily misconfigured.
  - Generates more false positives and false negatives.
  - Designed to prevent traffic on specific ports; does not affect adversary’s attack tool.
- **Suricata**, *OISF*
  - Rule feeds for specific attacks and attack tools are paywalled.
  - Does not affect adversary’s attack tool.
- **CrowdStrike Falcon**, *CrowdStrike*
  - Can be expensive; lose all access when license expires.
  - Generates high false negatives.
  - Cloud dependency.
  - Complex integration process.
- **SentinelOne Singularity Complete**, *SentinelOne*
  - Can be expensive; lose all access when license expires.
  - Generates high false negatives.
  - Vendor lock-in.

Existing Projects

- **EkHunter**:
  - Automatically detects presence of web-based exploit vulnerabilities, and derives test cases to compromise integrity of exploit kit and even kit operator.
- **AutoFuzz**:
  - Framework for automatically testing and fuzzing network protocols for implementation flaws.
- **LSSM**:
  - AI/ML to actively resist deductive attacks from network traffic analysis.

Eshete, Birhanu, et al. "EKHunter: A Counter-Offensive Toolkit for Exploit Kit Infiltration." *NDSS*. 2015. http://dx.doi.org/10.14722/ndss.2015.23237.

Gorbunov, Serge, and Arnold Rosenbloom. "AutoFuzz: Automated Network Protocol Fuzzing Framework." *IJCSNS* 10.8 (2010): 239.

Z. Zhou, X. Kuang, L. Sun, L. Zhong and C. Xu, "Endogenous Security Defense against Deductive Attack: When Artificial Intelligence Meets Active Defense for Online Service," in IEEE Communications Magazine, vol. 58, no. 6, pp. 58-64, June 2020, doi: 10.1109/MCOM.001.1900367.

Existing Patents

- **IMMIX-Intrusion Detection & Prevention Systems**, *Cylance Inc.*
  - Endpoint detection and response utilizing machine learning.

- **“Detection of Anomalies, Threat Indicators, & Threats to Network Security”**, *Splunk*
  - Methodology for detecting anomalies in activity on a computer network by processing, using a plurality of machine-learning anomaly models to identify, generate, and process anomaly data.

- **Automated Analysis using Sandbox & Machine Learning Classification** *Veracode Inc.*
  - Methodology for automated behavioral and static analysis using a sandboxed environment and machine learning detection.

IMMIX, Rahul C. Kashyap, Vadmin D. Kotov, Samuel J. Oswald, 2018, ID: EP3568792B1

Splunk Security Platform, Sudhakar Muddu, Christos Tryfonas, 2019, ID: US20190342311A1

Veracode Automated Analysis Methodology, Theodora H. Titonis, Nelson R. Manohar-Alers, Christopher J. Wysopal, 2017, ID: EP2610776B1

Possible Approaches

- **Smart Fuzzing**
  - Uses built-in intelligence
  - Achieves higher code coverage
  - More efficient in terms of computation and time

- **Dumb Fuzzing**
  - Generates completely random input
  - May uncover bugs more readily
  - Time and computationally intensive

- **Comparison**
  - Smart fuzzers focus on specific areas of code
  - Smart fuzzers are generally more favorable due to efficiency
  - Dumb fuzzers explore broader possibilities

Possible Fuzzing Targets

- **Masscan**, *asynchronous port scanner* - 99.8% C
- **Aircrack-ng**, *WiFi security auditing suite* - 71.9% C
- **Medusa**, *login brute-forcer* - 69.8% C
- **Reaver**, *WPS brute-forcer* - 99.4% C
- **Yersinia**, *framework for layer-2 attacks* - 95.9% C

Tool	Est. Importance	Est. Likelihood of Vulnerability	Priority
Aircrack-ng	5	3	15
Reaver	3	4	12
Yersinia	1	4	4
Masscan	3	3	9
Medusa	2	5	10

Scale of 0-5 where 0 is least important/likely and 5 is most important/likely

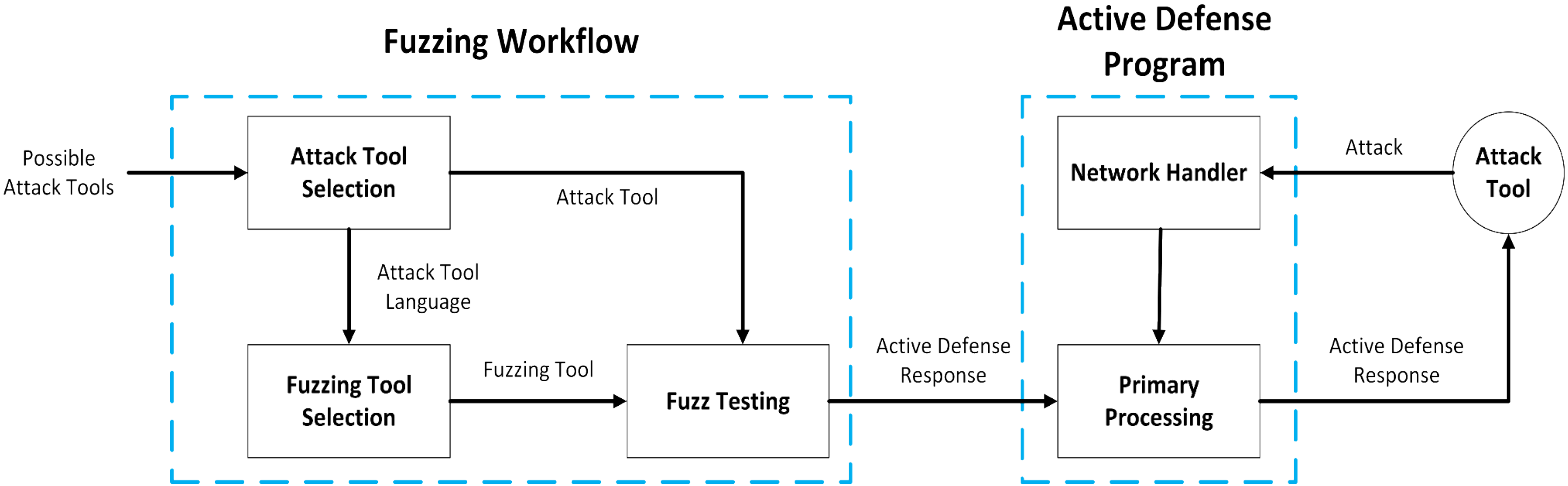
Risks for the Fuzzing Workflow

- **Attack Tool Identification**
  - Likelihood: Medium-High
  - Impact: High
  - Mitigation: Extensively test each of the various attack tool candidates with several analysis tools to ensure attack tool is sufficient
- **Vulnerability Discovery Capability**
  - Likelihood: High
  - Impact: Critical
  - Mitigation: Employ multiple fuzzing techniques, integrate with static analysis tools.
- **Fuzz Testing Workflow**
  - Likelihood: Medium
  - Impact: Moderate
- **Active Defense Response Generation**
  - Likelihood: High
  - Impact: Critical
  - Mitigation: Develop multiple response types, implement thorough testing procedures
- **Scalable Framework Implementation**
  - Likelihood: Medium
  - Impact: Moderate
- **Overall Risk Exposure:** High-Moderate

Summary

- Modern attack tools are quick and efficient. To give a security team time to respond, it is important have a way to slow or stop them.
- We plan to accomplish this by providing network responses that cause an attacking tool to crash or hang
- This should slow down the attack or prevent the attacker from using that specific tool
- We plan on developing a workflow that streamlines this process to make it easier for others to accomplish the same thing

Functional Decomposition



Fuzzing Workflow

Active Defense Program

Possible Attack Tools

Attack Tool Selection

Attack Tool

Attack Tool Language

Fuzzing Tool Selection

Fuzzing Tool

Fuzz Testing

Active Defense Response

Network Handler

Primary Processing

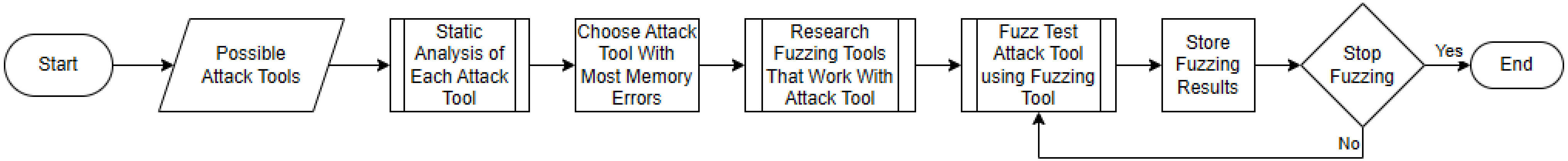
Attack

Attack Tool

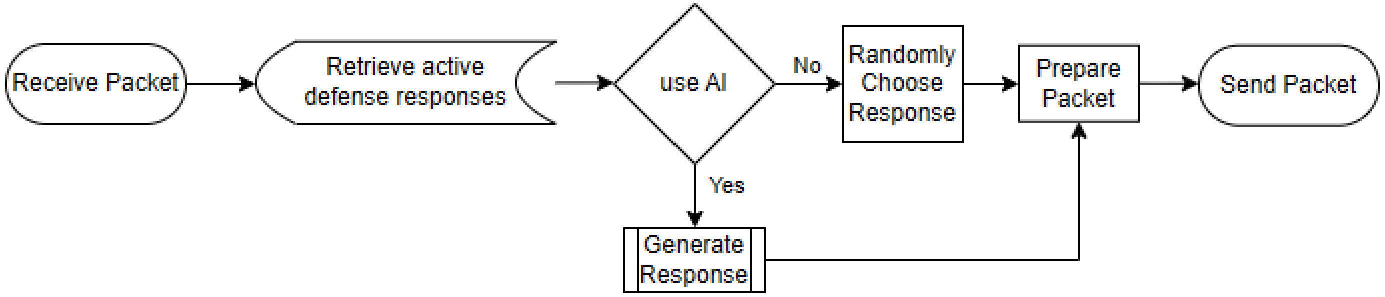
Active Defense Response

Behavioral Decomposition

Fuzzing Workflow


Start
Possible Attack Tools
Static Analysis of Each Attack Tool
Choose Attack Tool With Most Memory Errors
Research Fuzzing Tools That Work With Attack Tool
Fuzz Test Attack Tool using Fuzzing Tool
Store Fuzzing Results
Stop Fuzzing
Yes
End
No

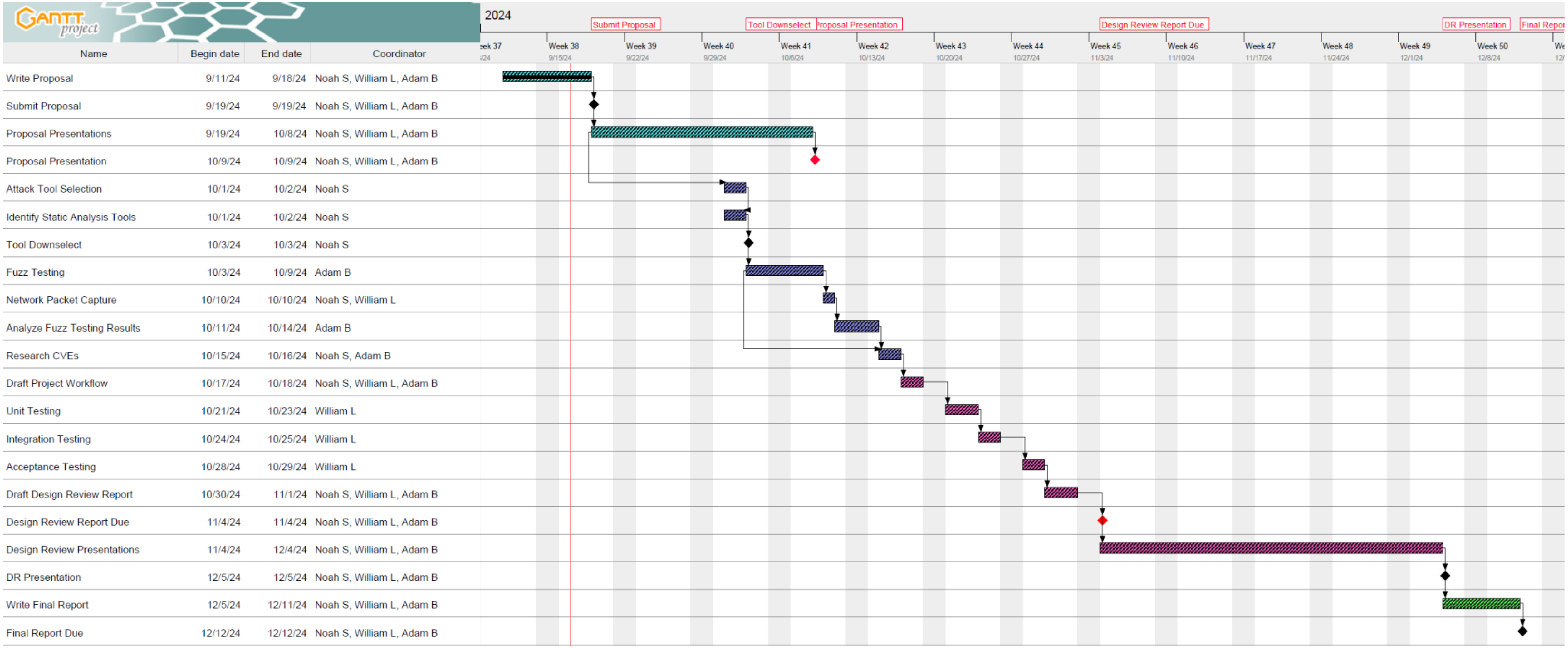
Proof of Concept



Testing Plan

- Unit Tests
    - Valid fuzzing test response(s) acquisition
    - Network traffic acquisition
    - Attack tool network traffic identification
- Integration Tests
    - Analyzing captured network traffic
    - Sending generated response back to adversary
    - Script sends proper response based on identified traffic
- Acceptance Tests
    - Workflow adequately finds and generates counter-attack responses to adversarial applications
    - Appropriate response generated
    - Attack tool network traffic correctly identified

Project Timeline



Name	Begin date	End date	Coordinator
Write Proposal	9/11/24	9/18/24	Noah S, William L, Adam B
Submit Proposal	9/19/24	9/19/24	Noah S, William L, Adam B
Proposal Presentations	9/19/24	10/8/24	Noah S, William L, Adam B
Proposal Presentation	10/9/24	10/9/24	Noah S, William L, Adam B
Attack Tool Selection	10/1/24	10/2/24	Noah S
Identify Static Analysis Tools	10/1/24	10/2/24	Noah S
Tool Downselect	10/3/24	10/3/24	Noah S
Fuzz Testing	10/3/24	10/9/24	Adam B
Network Packet Capture	10/10/24	10/10/24	Noah S, William L
Analyze Fuzz Testing Results	10/11/24	10/14/24	Adam B
Research CVEs	10/15/24	10/16/24	Noah S, Adam B
Draft Project Workflow	10/17/24	10/18/24	Noah S, William L, Adam B
Unit Testing	10/21/24	10/23/24	William L
Integration Testing	10/24/24	10/25/24	William L
Acceptance Testing	10/28/24	10/29/24	William L
Draft Design Review Report	10/30/24	11/1/24	Noah S, William L, Adam B
Design Review Report Due	11/4/24	11/4/24	Noah S, William L, Adam B
Design Review Presentations	11/4/24	12/4/24	Noah S, William L, Adam B
DR Presentation	12/5/24	12/5/24	Noah S, William L, Adam B
Write Final Report	12/5/24	12/11/24	Noah S, William L, Adam B
Final Report Due	12/12/24	12/12/24	Noah S, William L, Adam B

- **Critical path duration:** ~85 days

Individual Responsibility

Adam Brannon:

- Primary: Fuzzer
- Secondary: Primary processing

Noah Sickels:

- Primary: Attack tool selection
- Secondary: Fuzz tool selection

William Lochte:

- Primary: Packet Capture
- Secondary: Testing (Unit, Integration, Acceptance)

Cost Estimation

Estimated Cost: $300.00

Expected Software:

- AFL (including forks)
  - Cost: $0.00
- LDRA
  - Cost: $0.00
  - Licensed to UAH
- Premium Chat GPT
  - Cost: $20.00/person per month

Labor Cost: 150 hours/person, 450 hours total

CPE 495/496 Expected Deliverables

ID	Activity	Description	Deliverables / Checkpoints	Duration (Days)	People	Resources	Prede-cessors
1	**Tool(s) Selection**						
1.1	Attack Tools Selection	Choose attack tools to analyze	● Identify two potential tools for testing	1	Noah		
1.2	Fuzzing Tool(s) Selection	Choose tools to analyze with	● Identify all fuzzing tools for workflow	1	Noah		
1.3	Pre-screening Tools	(opt) Prescreen attack tools for candidate	● Identify two potential tools for testing	1	Adam		
2	**Tool Analysis**						
2.1	AFL/AFL++ Fuzzing	Perform testing on attack tool with AFL/AFL++	● Confirmed crashes/hangs	3	Adam	● AFL tools ● Attack tools ● Test bench	1.1, 1.2
2.2	LDRA Static Analysis	Perform testing on attack tool with LDRA	● Confirmed memory leaks	3	Will (1) Adam (2)	● LDRA TestBed ● LDRA TBRun ● Test bench	1.1, 1.2

CPE 495/496 Expected Deliverables

ID	Activity	Description	Deliverables / Checkpoints	Duration (Days)	People	Resources	Prede-cessors
3	**Networking Traffic**						
3.1	Attack tool network traffic	Capture traffic using Wireshark while running attack tool	● Captured network traffic .PCAP file	1	Noah	● Attack tool ● Wireshark ● Metasploitable VM ● Test bench	1.1
4	**Test Data Analysis**						
4.1	Review attack tool network traffic	Review captured network traffic from attack tool.	● Identify attack tool traffic behavior and interaction	1	Adam	● Captured network traffic .PCAP file ● Wireshark ● Test bench	1.1, 3.1
4.2	Review Found Memory Leaks	Review discovered memory leaks from fuzzing & analysis test	● Identify actionable memory leaks	1-2	Will (1) Adam (2)	● AFL/AFL++ test results ● LDRA test results	1.1, 1.2, 2.2
4.3	**Research Attack Tool**						
4.3.1	Find relevant CVEs	Search for CVEs relevant to attack tool	● Relevant CVE ID(s)	1-2	Adam (1) Noah (2)	● CVE database ● Exploit-DB	1.1, 2.1, 2.2, 3.1

CPE 495/496 Expected Deliverables

ID	Activity	Description	Deliverables / Checkpoints	Duration (Days)	People	Resources	Prede-cessors
5	**Project Workflow**						
5.1	Draft fuzzing workflow	Begin fuzzing workflow design and report	● Complete fuzzing workflow	3	All	● Fuzz test results ● LDRA testing results ● CVE entries	1.2, 4.1
6	**Testing Plan**						
6.1	Unit Tests	Unit testing of individual components	● Workflow components verification	1-2	Will (1) Noah (2)	● Verified fuzz testing workflow ● Test bench	5.1
6.2	Integration Tests	Integration tests of components together		1-2	Will (1) Noah (2)	● Verified components from prior tests ● Test bench	5.1, 6.1
6.3	Acceptance Tests	Customer acceptance of workflow	● Meet with customer ● Customer acceptance of fuzzing workflow	1-2	Will (1) Adam (2)	● Successful unit/integration tests	5.1, 6.1, 6.2
6.4	Revise Test Plans	Revise testing plans as needed based on feedback	● Finalized, verified testing workflow	1-2	Adam (1) Will (2)		6.1, 6.2, 6.3

CPE 495/496 Expected Deliverables

ID	Activity	Description	Deliverables / Checkpoints	Duration (Days)	People	Resources	Prede-cessors
7	**Design Review**						
7.1	Draft Design Review report	Begin writing Design Review report	● Design Review report	3	All	● Completed fuzzing workflow	5.1, 6.1, 6.2, 6.3
7.2	Design Review due	Design Review submission	● Design Review report	1	All	● Completed DR report	7.1
7.3	Design Review Presentation	Class group presentation for Design Review	● Design Review report	1	All	● Completed DR report	7.1, 7.2
8	**Final Report**						
8.1	Draft Final Report	Begin writing final project report	● Final project report	5	All	● Comments from Design Review	7.3
8.2	Final Report due	Final Report submission	● Final project report	1	All	● Completed Final project report	7.3, 8.1

Questions

Appendix A:

Full Engineering Requirements

Appendix A: Engineering Requirements

Marketing Requirements	Engineering Requirements	Justification
1	Workflow should prioritize tools that are at least 25% C/C++ or have memory errors during static analysis	C/C++ are memory unsafe languages, which mean they are more likely to have memory vulnerabilities. We allow static analysis as an alternative to support other languages
1,2	Workflow should choose a static analysis tool	Static analysis will help with prioritizing attack tools
1,2	Workflow should choose a fuzzing tool	Choice of fuzzing tool will depend on scenario
2,4,5	Workflow should find at least 1 exploitable vulnerability within attacking application	To be able to stop or slow the attacking application
3,4,6	Proof of concept should be able to crash or hang the attacking application for at least 1 second	A 1 second hang will greatly reduce the efficiency of the attacking application

Appendix A: Engineering Requirements

Marketing Requirements	Engineering Requirements	Justification
7	Report should include keywords at the start	IEEE report standard
7	Report should include abstract at the start	IEEE report standard
7	Report should include byline at start	IEEE report standard
7	Figures, tables and equations should be at the top or bottom of the page not the middle	IEEE report standard
7	Report body should use 10 pt Times New Roman font	IEEE report standard
7	Report body should be formatted as two columns per page	IEEE report standard
7	Report should use IEEE style headers	IEEE report standard

Abstract

Modern attack tools are efficient, allowing them to attack quickly. To mitigate this risk, it is important to have a way to slow or stop them. Our solution plans to accomplish this by finding network responses that cause attack tools to crash or hang, and then providing those network responses whenever the attack tool makes a request. We expect that our project will lead to a workflow for countering attack tools using fuzz testing.