Risk-based Approach to Cyber Vulnerability Assessment using Static Analysis

Sponsor: OUSD(R&E) | CCDC AC  [ART-006]

By
Barry Horowitz, Tim Sherburne
Peter Beling, Cody Fleming, Stephen Adams, Davis Loose

11th Annual SERC Sponsor Research Review
November 19, 2019
FHI 360 CONFERENCE CENTER
1825 Connecticut Avenue NW, 8th Floor
Washington, DC 20009

www.sercuarc.org
U.S. Provisional Patent Application Serial No. 62/935,835
Filed on November 15, 2019
Title: Method and System for Risk-based Approach to Cyber Vulnerability Assessment Using Static Analysis
UVA LVG Reference: HOROWITZ-RISK (02620-01)
Static Analysis for Detecting Cyber Attack Vulnerabilities

• Static Analysis (SA) is a method of computer program debugging that is done by examining the code without executing the program.

• The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards.

• Automated tools can assist programmers and developers in carrying out static analysis.

As defined by Margaret Rouse of Whatis.com
Static Analysis Tools Applied to Discovering Cyber Security Vulnerabilities

- A significant number of static analysis tools exist for discovering cyber security vulnerabilities supporting a variety of:
  - Operating Systems, including Linux and Windows
  - Programming and scripting languages, including C/C++, Java, Perl and Python

- Tools discover SW structures that can support the objectives of cyber attackers, and provide input to programmers and designers for consideration regarding modification of the identified vulnerable SW

- Tool outputs include identifying specific lines of code that are problematic and should be candidates for modification
DoD User Experience and the Research Problem

- DoD user experience has identified significant productivity issues with using Static Analysis tools for cyber security purposes:
  - Time and analyst skill level required to assess identified vulnerabilities as worthy of correction
  - The large number of identified vulnerabilities, most of which are not considered as worthy of change (false positives)

- UVA is involved with the Armament Division of the Army in exploring a system level risk based methodology for using automation for prioritizing detected vulnerabilities derived from static analysis tools

- Methodology includes a system user developed prioritized list of system failure (hard and soft) consequences to be avoided
System Diagram (pre-resilience) for a hypothetical Landmine-based Weapon System

- **C2 Center (Doctrinal and Mission Planning Information)**
- **UAV**: Command, Static Images, Metadata & Flight Data
- **Operator Station**: Text Messages, Encrypted Communication Channel, Wireless Encrypted Network
- **Radio Relay**: Obstacle 1, Obstacle 2, Obstacle n
- **Operator**: 2 IR Cameras
- **Emulator**
## Example Table of Prioritized System Failure Consequences to be Avoided

<table>
<thead>
<tr>
<th>Attack Outcome</th>
<th>Related Attack Target(s)</th>
<th>User Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inappropriate firings via manipulating operator commands</td>
<td>Operator control display, radio comm links</td>
<td>1</td>
</tr>
<tr>
<td>Delays in fire time (sufficient delay to cross field)</td>
<td>Obstacles, control station, radio comm links</td>
<td>1</td>
</tr>
<tr>
<td>Delays in deployment</td>
<td>Obstacles, deployment support equipment</td>
<td>1</td>
</tr>
<tr>
<td>Deactivation of a set of obstacles</td>
<td>Obstacles</td>
<td>1</td>
</tr>
<tr>
<td>Delays in situational awareness</td>
<td>Operator display, sensors</td>
<td>2</td>
</tr>
<tr>
<td>Prevent or corrupt transmission of situational awareness data</td>
<td>Radio comm links, operator display, sensors</td>
<td>2</td>
</tr>
<tr>
<td>Gain information to help adversary navigate through field</td>
<td>Obstacle, operator control station</td>
<td>2</td>
</tr>
<tr>
<td>Reduced operational lifespan</td>
<td>Obstacle</td>
<td>3</td>
</tr>
<tr>
<td>Prevent transmission/execution of non-firing commands</td>
<td>Operator display, obstacles</td>
<td>3</td>
</tr>
<tr>
<td>Delays in sending/receiving C2 information</td>
<td>Operator display, radio comm links</td>
<td>4</td>
</tr>
<tr>
<td>Delays in un-deployment</td>
<td>Obstacles</td>
<td>4</td>
</tr>
</tbody>
</table>
**Problem:** Which of the consequences to be avoided are dependent on which lines of code identified via static analysis
• Dynamic system testing currently includes tests that relate to other than cyber attack stimulants for system fault tolerance and resilience features:
  — Safety
  — Operator errors
  — Out of Spec situations (e.g., overloads, potential anomalous circumstances)
  — System countermeasures (electronic warfare, tampering)
  — Technology component failures (hard and soft failures)
  — Etc.

• These focused dynamic system tests provide a basis to use already available compiler-based SW tracing results as a means for identifying the specific SW modules and files used in the process of evaluating system design related to specific faults.
• Hypothesis regarding application of static analysis results to prioritization of cyber attack risks:
  — System risks which are currently tested for will include significant consequence overlaps with those derived from cyber attacks, thereby providing a basis for using SW tracing based upon already existing system tests as a mechanism for identifying which of the static analysis results relate to which of the identified cyber attack consequences
  — As cyber attack resilience emerges as an additional area of system design, dynamic system test results for cyber resilience can be utilized to enable more effective use of new static analysis results that emerge over the life cycle due to:
    o system design changes,
    o changing cyber attack techniques,
    o new findings that result from modifications in static analysis tool designs
Initial Research Findings
Initial Research Context

Static Analysis

Automated Code Trace

Mission Aware Prioritization

static analysis alerts

code & testcase

code & testcase

code

GCC

Code Coverage per Test Case

Static Analysis

ARDUPILOT

SWAMP

university virginia engineering research center
ArduPilot is an open source autopilot system supporting multiple autonomous vehicle types. SITL (software in the loop) simulator allows ArduPilot execution without vehicle hardware. ArduPilot’s Auto Test suite allows for the creation of repeatable tests of autopilot behavior based on SITL simulator.

Component Name | KLOC (C/C++)
--- | ---
Helicopter | 15
Fixed Wing Plane | 13
Land Rover | 6
Submarine | 6
Shared Libraries | 177
**Total** | **217**

https://github.com/ArduPilot/ardupilot

SSRR 2019

November 19, 2019
Initial Research Context

- **Static Analysis**
  - **SWAMP**
  - **ARDUPilot**
- **Automated Code Trace**
  - GCC
- **Code Coverage per Test Case**
  - TXT
- **Mission Aware Prioritization**
- **Static Analysis Alerts**
  - TXT
Led by the Morgridge Institute for Research in Madison WI, the Software Assurance Marketplace (SWAMP) is a no-cost, cloud service that provides Static Code Analysis to developers and researchers.

Available SA Tools (C/C++)  |  Type  
---|---
Clang  |  Open Source  
Cppcheck  |  Open Source  
CodeSonar  |  Commercial  
Coverity  |  Commercial  
Code DX (Consolidated Results Viewer)  |  Commercial  

- Variation in SA tool results are similar to findings of NIST Static Analysis Tool Exposition (SATE).
- Alert Density of 2.0 equates to 2,000 alerts for a 1 million LOC project.

---

<table>
<thead>
<tr>
<th>Code DX Results Summary</th>
<th>Alert Count</th>
<th>Alert Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool A</td>
<td>439</td>
<td>2.02</td>
</tr>
<tr>
<td>Tool B</td>
<td>778</td>
<td>3.59</td>
</tr>
<tr>
<td>Tool C</td>
<td>66</td>
<td>.30</td>
</tr>
<tr>
<td>Tool D</td>
<td>43</td>
<td>.20</td>
</tr>
</tbody>
</table>

(Ardupilot) Total Alerts | 1,326 |

1. Licensing terms prevent publication of tool specific results (list is reordered)
2. Little to no overlap between tools
3. Alerts / 1,000 Lines of Code (LOC)
4. Over full code base (all vehicle modes, test drivers, link protocol libs, etc.)
Initial Research Context

- **Automated Code Trace**
  - Input: code & testcase
  - Output: Code Coverage per Test Case

- **Static Analysis**
  - Input: code
  - Output: Static Analysis Alerts

- **Mission Aware Prioritization**
  - Input: Code Coverage per Test Case

**Tools and Technologies**
- **ARDUPIL**
- **SWAMP**
- **GCC**
Code Trace Overview

Code is instrumented at *Compile-time* to output function / line execution counts per code file at *Run-time*.

Each Test Case captures a Coverage Report

![coverage report](https://gcc.gnu.org/onlinedocs/gcc/Gcov.html)

**Example Code Coverage Tools**

- GNU C Compiler (GCC) – GCOV
- LLVM – X-ray
- Greenhills – Time Machine : Coverage Trace

**Example Greenhills Users**

- Jet Propulsion Lab
- Northrop Grumman
- Lockheed Martin

https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
https://llvm.org/docs/XRay.html
https://www.ghs.com/products/timemachine.html
ArduPilot Code Trace Example

ArduPilot Helicopter SITL build includes:
- **7,546 Functions**
- **69,743 LOC\(^1\)**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Test Type</th>
<th>Functions Executed</th>
<th>LOC Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loiter to Altitude</td>
<td>Safety</td>
<td>533</td>
<td>4291</td>
</tr>
<tr>
<td>Battery Failsafe</td>
<td>Component Failure</td>
<td>329</td>
<td>2927</td>
</tr>
<tr>
<td>Camera Control</td>
<td>Component Failure</td>
<td>391</td>
<td>3377</td>
</tr>
<tr>
<td>GPS Glitch</td>
<td>Out of Spec</td>
<td>373</td>
<td>3287</td>
</tr>
</tbody>
</table>

Merged Coverage Results for “Common” Code Executed Across Test Cases.

<table>
<thead>
<tr>
<th>Common Across:</th>
<th>Functions Executed</th>
<th>LOC Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (or more) Test Cases</td>
<td>376</td>
<td>3319</td>
</tr>
<tr>
<td>3 (or more) Test Cases</td>
<td>343</td>
<td>2914</td>
</tr>
<tr>
<td>4 Test Cases</td>
<td>245</td>
<td>2260</td>
</tr>
</tbody>
</table>

\(^1\)Does not include #define, #if <def> compiler directive, test code
GCC includes tooling for an HTML view - to assist visualization of results.

“Common” (across 4 test cases) libraries include:
- Attitude Control
- Battery Monitor
- Etc.

### Directory

<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Lines</td>
<td>Executed / Total</td>
<td>% Functions</td>
</tr>
<tr>
<td>AP_Motors</td>
<td>7.9 %</td>
<td>108 / 1374</td>
</tr>
<tr>
<td>AP_Arming</td>
<td>10.1 %</td>
<td>40 / 397</td>
</tr>
<tr>
<td>AC_PID</td>
<td>12.1 %</td>
<td>44 / 365</td>
</tr>
<tr>
<td>AP_NavEKF2</td>
<td>12.9 %</td>
<td>617 / 4778</td>
</tr>
<tr>
<td>AP_SmartRTL</td>
<td>17.7 %</td>
<td>66 / 372</td>
</tr>
<tr>
<td>AP_InertialNav</td>
<td>20.7 %</td>
<td>6 / 29</td>
</tr>
<tr>
<td>AP_BattMonitor</td>
<td>22.6 %</td>
<td>140 / 619</td>
</tr>
<tr>
<td>AC_AttitudeControl</td>
<td>25.3 %</td>
<td>287 / 1136</td>
</tr>
<tr>
<td>AP_Stats</td>
<td>31.2 %</td>
<td>20 / 64</td>
</tr>
</tbody>
</table>
Initial Research Context

**Automated Code Trace**

- Code & testcase
- GCC

**Static Analysis**

- Code
- SWAMP

**Mission Aware Prioritization**

- Code Coverage per Test Case
- Static Analysis Alerts

**Alerts**

- Code Coverage per Test Case
- Static Analysis Alerts
ArduPilot Test Case Correlation

Static Analysis Alerts are Attenuated by Correlation to High Priority Mission Test Case Code Files (4 Static Analysis Tools with 4 Test Cases)

<table>
<thead>
<tr>
<th>SA Tool</th>
<th>Total Alerts</th>
<th>ArduPilot SA Alerts Correlated to Code Coverage Test Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Alerts found in Files Executed in at least 1 Test Case</td>
</tr>
<tr>
<td>Tool A</td>
<td>439</td>
<td>91</td>
</tr>
<tr>
<td>Tool B</td>
<td>778</td>
<td>204</td>
</tr>
<tr>
<td>Tool C</td>
<td>66</td>
<td>3</td>
</tr>
<tr>
<td>Tool D</td>
<td>43</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>1,326</td>
<td>310</td>
</tr>
</tbody>
</table>

Initial Findings:

- For ArduPilot, filtering alerts based on Mission Priority test cases, provides attenuation of ~ 75%
- A small number of test cases can provide reasonable alert filtering results
Additional Research Efforts

• Conduct experiments with DoD application SW, engaging static analysis personnel to determine productivity advances

• Engage with DT and OT testing organizations to determine how one might address this opportunity into test programs

• Use of Natural Language AI technology to prioritize the cyber risk-related test cases through integration of documentation derived from Static Analysis, Hazard Analysis and prior test cases results

• Consider application to non-weapon related physical systems and IT systems

• Army interest in possible integration with SEI SCALe Tool