MATLAB EXPO 2019

Accelerating embedded software verification with Polyspace static code analysis

Stefan David
Agenda

1. Making Software Safe and Secure
2. Polyspace Static Analysis
3. Team Collaboration with Polyspace
1. Making Software Safe and Secure
Security is on consumers’ minds

…of customers would never buy from an OEM if they had been hacked

82%

…of automakers admit their organization had been hacked in the past 2 years

85%

according to 2016 KPMG Consumer Loss Barometer study
In the News.... Embedded Software Security - New Challenge

Hackers remotely kill a Jeep on the highway—with me in it

Source: https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/
In the News.... Embedded Software Security - New Challenge

Source: https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/
Safety & Security Goals

Safety

System

issue

Environment

Security

Environment

attack

System

Note: Security issues may cause safety issues
2016 = 6447
2017 = 14714
2018 = 16555

source: https://www.cvedetails.com (CVE ... Common Vulnerabilities and Exposures)
When Software Safety and Security Matter

- Industries where safety and security matter
  - Automotive, Aerospace, Medical Device, Industrial Machinery

- Governed by functional safety and other standards
  - ISO 26262, DO-178, IEC 62304, IEC 61508
  - ISO/SAE 21434, RTCA DO-326
  - MISRA, CERT, AUTOSAR

- Static analysis provides certification credits
  - For standards such as ISO 26262 and DO-178
Top 10 Secure Coding Practices

Validate inputs

Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].

Heed compiler warnings and use static and dynamic analysis tools

Avoided for your compiler and eliminate warnings by modifying the code [C MSC00-A, C++ MSC00-A]. Use static and dynamic analysis tools to detect and eliminate additional security flaws.

Architect/Design Software for security policies

Software architecture and design your software to implement and enforce security policies. For
“Program testing can be used to show the presence of bugs, but never to show their absence”

Edsger Dijkstra, Computer Science Pioneer

“Given that we cannot really show there are no more errors in the program, when do we stop testing?”

Brent Hailpern, Head of Computer Science, IBM

Dijkstra, “Notes on Structured Programming” (1972)
2. Polyspace Static Analysis

*For software written in C, C++, and Ada*
Proving Absence of Critical Run-Time Errors

- How many run-time errors are possible?
  1. Divide by zero
  2. Overflow
  3. Uninitialized variables

- How to test all floating point variable combinations?

- How do you prove that this code will not fail?

```c
float x, y;
...
x = x / (x - y);
```
Proving Absence of Critical Run-Time Errors

```c
float where_are_errors(float input)
{
    float x, y, k, l, limit = 1000.0f;
    if (input <= -limit || input >= limit) return (-9999.0f);
    k = input / 100.0f;
    x = 2.0f;
    y = k + 5.0f;
    while (x <= 10.0f)
    {
        x++;
        y = y + 3.141592f;
    } 
    if (3.0f*k + 100.0f >= 71.0f)
    {
        y++;
        x = x / (x - y);
    }
    return x;
}
```

Proven mathematically by Polyspace that run-time error will *not* occur
Experiences from the field...

Using Polyspace code verifiers...

- Identified and fixed potential run-time errors and unsafe code
- Reliably analyzed C codebase early, without test cases and compilation!

“Independent, systematic code reviews, compliance to MISRA-C”

“Bug Finder and Code Prover provided 1-2 Man-Year savings and automated capability in parallel to development which were not available otherwise”

(Source: Ralph Paul, Head of Flight Test & Dynamics, Solar Impulse)
Proving Absence of Critical Defects & Vulnerabilities (33)

Defect & Vulnerability Checkers (251)

Coding Standards, Cybersecurity Guidelines

Proves code Safe and Secure

33 most critical run-time checks

Supports DO-178 and ISO 26262

Bug Finder

- Produce code metrics
- Check coding standards
- Find defects and vulnerabilities

Code Prover

- Proves code Safe and Secure
- 33 most critical run-time checks
- Supports DO-178 and ISO 26262

Polyspace Tools

MATLAB EXPO 2019
Example: Optimize design and architecture

Non Robust Module

Potential Runtime Error inside!!!
Example: Optimize design and architecture

Non Robust Module

Legacy code

Additional Range-Limiting Code

Free from Runtime Errors
Using Static Analysis to Make Software Safe and Secure

- Find bugs without code execution
  - Code analyzed without running tests
  - Identify bugs and coding rule violations for MISRA, AUTOSAR, CERT

- Prove absence of critical run-time errors
  - Identify code that will never experience errors regardless of run-time conditions

- Complements dynamic testing
  - Used together, you can find more bugs for higher quality code
Polyspace Customer References

KOSTAL Asia R&D Center Receives ISO 26262 ASIL D Certification for Automotive Software

Alenia Aermacchi Develops Autopilot Software for DO-178B Level A Certification

Miracor Eliminates Run-Time Errors and Reduces Testing Time for Class III Medical Device Software
3. Team Collaboration with Polyspace
Proving Absence of Critical Run-Time Errors with Polyspace
Workflow with New Polyspace Products in R2019a

1. Developers check-in code into repository, Build Engineer has configured Jenkins to run Polyspace analysis
2. Jenkins initiates Polyspace analysis run on the server (periodically or at program milestones)
3. Once Polyspace analysis run concludes, results are uploaded to Polyspace Access
4. Team Lead/Manager, QA, Developers use web browser to review results, open Jira defects, monitor quality metrics
Bob is the Build Engineer
He has configured Polyspace in a Jenkins CI workflow
Quinn is a Quality Engineer
She is responsible for triaging software defects

- She received an email notification from last night’s Jenkins initiated Polyspace analysis
- The email indicates several findings were found in her project
- She click on the link in the email to view the findings in Polyspace Access

Polyspace Code Verification: 114 new findings for project Zen

Sun 3/17/2019 6:02 PM
Bob Builder
To: Quin Quality

Polyspace found 114 new findings when analyzing 'xent':
- To view details, check attached file and follow urls.
- To go to directly to project, follow: https://polyspace-access:9443/metrics/index.html?a=review&p=81&r=1996.

You can see the Jenkins log file here: http://jenkins-polyimage:8080/job/polyspace-modules/38/console.

Bob Builder
Build Engineer, Tools Group
(508) 647-3027 Builder@mathworks.com
Quinn is a Quality Engineer
She is responsible for triaging software defects
Dara is a software developer
She is responsible for writing code and fixing defects

- Dara has been assigned 2 defect tickets in Jira
- She opens the first JIRA ticket and clicks the Polyspace Access link
Dara is a software developer
She is responsible for writing code and fixing defects
Martin is a project manager.
Summary

- Use Polyspace to achieve high quality software with reduced testing effort
  - Prove that your code will not cause safety hazards or security issues

- Polyspace fits software development workflows
  - Jenkins for build automation and Jira for bug tracking

- Supports team based collaboration
  - Results published for web-browser based review by developers and quality engineers
  - Dashboards to show quality metrics for project and safety managers.
Finally.... Jeep Hack: Deterministic Random Number Generator

```
int c_max = 12;
int c_min = 8;
unsigned int t = time((void *)0);
unsigned int len = (rand() % (c_max - c_min + 1)) + c_min;
unsigned int v10 = rand();
int v11 = convert_byte_to_ascii_letter(v10 % 62);
password[v9] = v11;
v9++;
```

default: ID 2: ‘rand’ is a cryptographically weak PRNG.
To make your program more secure, use ‘CryptGenRandom’ (Windows) or ‘RAND_bytes’ (OpenSSL) instead.

time() = integer 2,147,483,647 possibilities (2^{32}-1)

impossible

01-01-1970
impossible

01-19-2038

today
End
Backup
New Polyspace Products in R2019a

1. Products for web browser results access
   - *Polyspace Bug Finder Access* and *Polyspace Code Prover Access*
   - Web-browser based review of static code analysis results
   - Integration with Jira

2. Products for servers
   - *Polyspace Bug Finder Server* and *Polyspace Code Prover Server*
   - Support for Continuous Integration systems such as Jenkins

3. Products for desktop use
   - *Polyspace Bug Finder* and *Polyspace Code Prover*
   - Find bugs and run time errors before submitting code to repository
## Polyspace Helps Makes C, C++, and Ada Safe and Secure

<table>
<thead>
<tr>
<th>Safety Standards:</th>
<th>Security Standards:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• DO-178 (aero)</td>
<td>• MISRA</td>
</tr>
<tr>
<td>• ISO 26262 (auto)</td>
<td>• AUTOSAR</td>
</tr>
<tr>
<td>• IEC 61508 (industrial)</td>
<td>• CERT-C</td>
</tr>
<tr>
<td>• IEC 62304 (medical)</td>
<td>• CWE</td>
</tr>
<tr>
<td>• EN 50128 (rail)</td>
<td>• ISO 17961</td>
</tr>
<tr>
<td></td>
<td>• MISRA-C:2012 Appendix 1</td>
</tr>
<tr>
<td></td>
<td>• Tainted data tracking</td>
</tr>
</tbody>
</table>

### Reliability and Robustness

**Code Proving**
- Prove absence of critical runtime errors (or find even the slightest vulnerability)
- Exhaustive: all possible inputs, control flows, data flows (no instrumentation, execution, test cases)
- Sound: no false negatives

### Quality

- **Coding Standards**
- **Find Probable Bugs, Defects**
- **Code Metrics**
- **Formal Method: Runtime Behavior, Debugger-like view**
- **Review Scopes / Software Quality Objectives**
- **Simulink Integration: trace issues in generated code back to model**
Optional Workflow: Analyze and Verify Code Prior to Check-In

• Run Polyspace Bug Finder and Polyspace Code Prover interactively

• Analyze code before it gets checked into the source code repository
When To Use Polyspace

- Checking generated code
  - Integrated code may consist of handwritten code + generated code
  - For certification, check coding standards (MISRA, CERT)
  - For AUTOSAR, prove interface requirements are met

- Check new code as soon as it written
  - Find issues early, when it is easier and cheaper to fix

- For heritage or legacy code
  - Fix issues when modifications to code are made
  - Create a baseline, only review new findings
  - Justify findings you don’t wish to fix or review again
Abstract

Do you need evidence that your code will not cause safety hazards or security issues? Polyspace products allow you to achieve the highest levels of software quality with reduced testing effort. Using formal methods based static code analysis, it can prove that your code is free from certain critical run-time errors. The analysis can be done interactively by software developers during code development to quickly find coding defects and violations of safety and security standards like MISRA, CERT-C/C++. When used with Continuous Integration tools such as Jenkins, Polyspace helps improve software quality, safety, and security across your projects. Results are published for web-browser based code review with tracing information to identify the root cause of defects. Polyspace supports modern team collaboration dashboards to show quality metrics for project and safety managers. Integration with defect tracking tools such as Jira help manage issues across your development enterprise.
Outline

- Static Analysis Concepts
  - Why is it important, what is it
  - Relevance to Auto, Aero, Med, IAM industries

- Polyspace Static Analysis
  - Proving absence of run-time errors
  - Polyspace products
  - Customer references (values and benefits)

- Team Collaboration with Polyspace
  - Workflow overview with new products
  - Build automation – runs Polyspace on server, sends email notifications
  - Quality Engineer, Team Lead – reviews results, triages and assigns defects
  - Developer – uses PS Access to debug defects, fixes code, does pre-submit checks
  - Project, Quality Manager – monitors trends
  - Pre-submit workflow

- Summary
Workflow for Quality Engineers

- Quin is a Quality Engineer

- She has received an email notification indicating XX new defects have been found in various projects that were analyzed last night

- She clicks on the links in the email to view results of the analysis
- She looks at the Project Overview Dashboard to identify projects and issues to focus on
- She can triage issues and opens Jira tickets from the PS Access web-browser

- She notices that code belonging to Dara the developer has dead code in a case statement
- She opens a Jira ticket from within Polyspace Access and assigns defect in Jira to Dara

- Show video of these tasks in Polyspace Access
Software Developer Responding to Issues

- Dara looks at defects assigned to her in Jira
- She clicks on the link the Jira ticket to debug issue via web-browser with Polyspace Access
- She notices that priorities can be set, annotations can be provided to report on status, all from within the web-browser interface of Polyspace Access
- She uses the information provided by the tool (result details and contextual help) to formulate a fix for the defect
- Dara fixes the code to address the unreachable case statement and checks it in
- Show video of developer performing these tasks with Polyspace Access and in the code editor to fix the defect
Workflow for Project Manager

- Doug is a project manager with responsibility for software quality

- He monitors overall project status via web-browser dashboard
- He checks SQO levels and compliance to standards (MISRA, CERT)
- He also can see that the defect that Dara fixed has been confirmed to be fixed in the last analysis run that was initiated by Jenkins

- Show screenshots or short video of these tasks
Workflow for Developers

- Dara is a software developer

- She is tasked with adding a new feature which requires changing the behavior of a function that has a case statement

- She makes the code change, then runs her unit tests, which all pass, then checks the code into the source code repository

- Show short video of code edits and command line execution of unit tests
Bob Builder

Polyspace Code Verification: 114 new findings for project Zen

To: Quin Quality

Polyspace found 114 new findings when analyzing 'xent':
- To view details, check attached file and follow urls.
- To go to directly to project, follow: https://polyspace-access:9443/metrics/index.html?a=review

You can see the Jenkins log file here: http://jenkins-polyspace:8080/job/polyspace_modules/1888/

Bob Builder
Build Engineer, Tools Group
(508) 647-3027 bbuilder@mathworks.com

Polyspace Code Verification: 2 new findings for project Zen

To: Quin Quality

Polyspace found 2 new findings when analyzing 'Pointer_Arithmetic':
- To view details, check attached file and follow urls.
- To go to directly to project, follow: https://polyspace-access:9443/metrics/index.html?a=review&p=81&r=1898.

You can see the Jenkins log file here: http://jenkins-polyspace:8080/job/polyspace_modules/38/console.
Quinn is a Quality Engineer
She is responsible for triaging software defects

- She has received an email notification from indicating 2 new findings were found in her project
- She click on the link in the email to view new findings in Polyspace Web UI
- The results list shows 2 findings that are in Dara’s code
- She opens two Jira tickets and assigns them to Dara
Dara is a software developer
She is responsible for writing code and fixing defects

- She opens the first JIRA ticket and clicks the Polyspace Access link
- She uses the information provided by the tool (result details and contextual help) to formulate a fix for the defect
- She fixes the defect in her IDE and check-in the changes
▪ She opens the second JIRA ticket and clicks the Polyspace Access link
▪ She determines that no code changes are required
▪ She changes the status to justified
▪ She writes a comment to explain her reasoning
Proving Absence of Critical Run-Time Errors with Polyspace
Proving Absence of Critical Run-Time Errors with Polyspace