


PA193 - Secure coding principles and practices



LAB: Static analysis of source code

Łukasz Chmielewski  chmiel@fi.muni.cz (email me with your questions/feedback)
Centre for Research on Cryptography and Security, Masaryk University

CRCS
Centre for Research on
Cryptography and Security

Overview - Lab

- Goal: Learn how to use basic tools
- Discuss false positives / false negatives
- Part I – Github Actions
- Part II - Tools
 - Check C/C++ code with with CppCheck, compiler warnings, VS PReFast
 - Check Java code with FindBugs

Disclaimer (Part I)

- The slides for this seminar (and part of the lecture) are based on the lecture for PV080.
 - there will be some pv080 on screenshots
 - we have more work than in PV080
- If you have already been absolved this course, try to enjoy it again 😊 and try to do extra tasks

Idea of the seminar

- Prepare repo with vulnerable code
 - IS->buggycode.zip – Part I and II
 - IS->crypto-java.zip – Part II, but you can try also with GitHub
- Part I: enable automatic static analysis via GitHub Actions
 - Several providers of analysis environment (custom or standard tools)
 - Trigger by commit, investigate warnings/errors found
 - Warning: in Code Scanning Actions there may be glitches, UI bugs and tool failures
- Part 2: Standalone Tools
- Fix it, review again

Basic analysis of C/C++ source code with various tools

CODE SCANNING WITH GITHUB + ACTIONS + CODACY

Steps

1. Create repo on GitHub
2. Enable code analysis
3. Clone repo locally
4. Insert code with vulnerability, commit and push
5. Investigate results of analysis
6. Fix selected issue, rerun analysis
7. Repeat from step 5.

Create repo on GitHub

- Online at github.com
- Make repo public
 - GitHub Actions are free only for public ones
- Add readme, .gitignore, license
 - Generally good practice
- For simplicity, don't mix languages
 - Put code of single lang in repo (e.g, c++)
 - Makes automatic analysis more difficult

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * /

Great repository name. Try to use lowercase letters without spaces. How about [curly-carnival?](#)

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

Enable code scanning actions

- Online at github.com
- *Github* → *Repo* → *Security* → Set up code scanning
- Select Codacy Security Scan (scroll down in offered scans)
 - ‘Set up this workflow’ button

The screenshot shows the GitHub interface for a repository named 'petrs / pv080_test_cpp'. The 'Security' tab is selected and highlighted with a red box. In the 'Security overview' section, the 'Code scanning alerts' option is highlighted with a red box, and the 'Set up code scanning' button is also highlighted with a red box. In the 'Security analysis from the Marketplace' section, the 'Codacy Security Scan' action is highlighted with a red box, and its 'Set up this workflow' button is also highlighted with a red box.

Get started with code scanning

Automatically detect common vulnerabilities and coding errors

CodeQL Analysis
by GitHub

Security analysis from GitHub for C, C++, C#, Java, JavaScript, TypeScript, Python, and Go developers.

Set up this workflow

Security analysis from the Marketplace

42Crunch API Security Audit
by 42crunch

Use the 42Crunch API Security Audit REST API to perform static application security testing (SAST) on OpenAPI/Swagger files.

Codacy Security Scan
by Codacy

Free, out-of-the-box, security analysis provided by multiple open source static analysis tools.

Unwatch 1 Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki **Security** Insights Settings

Overview
Security policy
Security advisories 0
Dependabot alerts
Code scanning alerts

Security overview

- **Security policy**
Define how users should report security vulnerabilities for this repository
Set up a security policy
- **Security advisories**
View or disclose security advisories for this repository
View security advisories
- **Dependabot alerts**
Get notified when one of your dependencies has a vulnerability
Enable Dependabot alerts
- **Code scanning alerts**
Automatically detect common vulnerability and coding errors
Set up code scanning

Commit configuration file for Codacy scan

- No changes required to `codacy_analysis.yml`
 - Start commit → Commit new file
 - Can be found at `/.github/workflows/ codacy_analysis.yml` for later edits

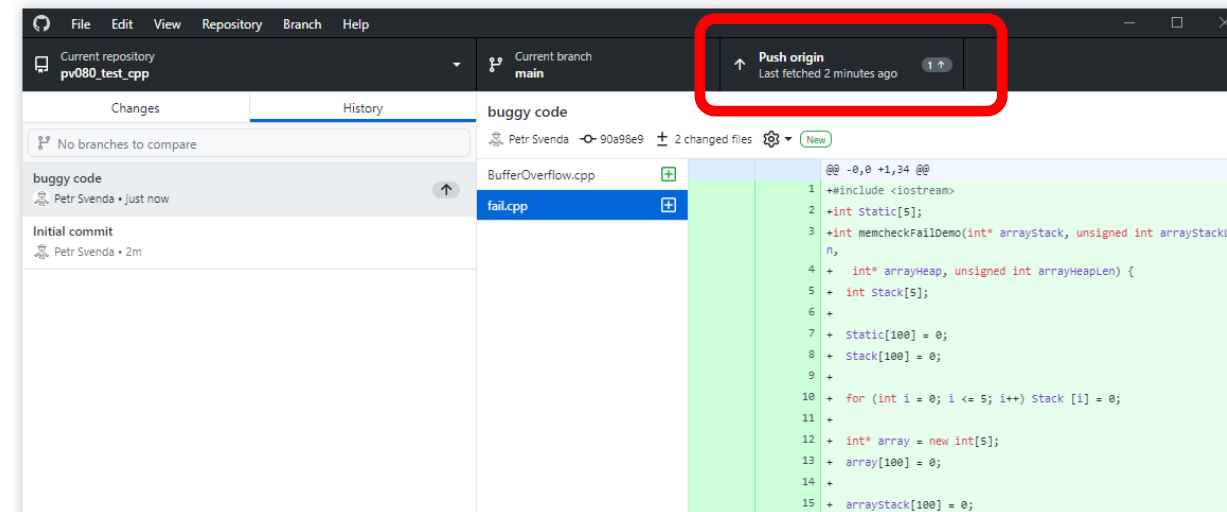
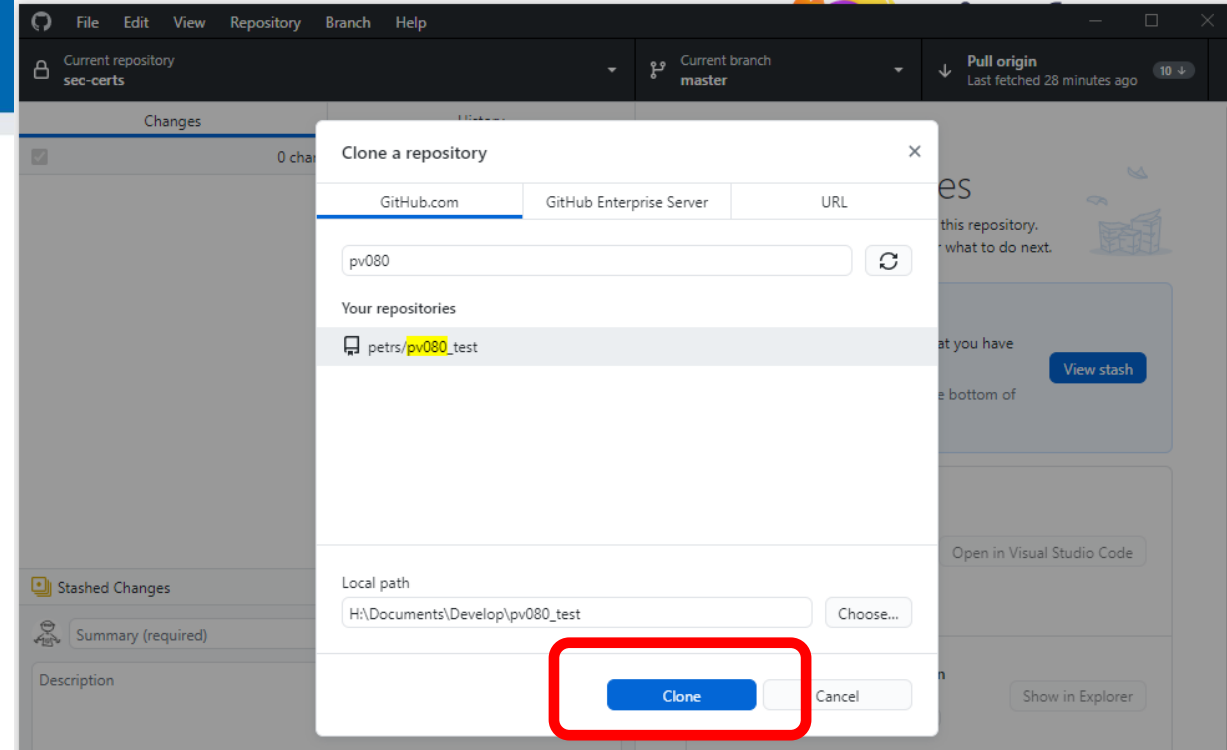
The screenshot shows the GitHub repository interface for `petrs / pv080_test_cpp`. The file `codacy-analysis.yml` is being edited in the `/.github/workflows/` directory. The file content is as follows:

```
1 # This workflow checks out code, performs a Codacy security scan
2 # and integrates the results with the
3 # GitHub Advanced Security code scanning feature. For more information on
4 # the Codacy security scan action usage and parameters, see
5 # https://github.com/codacy/codacy-analysis-cli-action.
6 # For more information on Codacy Analysis CLI in general, see
7 # https://github.com/codacy/codacy-analysis-cli.
8
9 name: Codacy Security Scan
10
11 on:
12   push:
13     branches: [ main ]
14   pull_request:
15     branches: [ main ]
16
17 jobs:
18   codacy-security-scan:
19     name: Codacy Security Scan
20     runs-on: ubuntu-latest
21     steps:
22       # Checkout the repository to the GitHub Actions runner
23       - name: Checkout code
24         uses: actions/checkout@v2
25
26       # Execute Codacy Analysis CLI and generate a SARIF output with the security issues identified during the analysis
27       - name: Run Codacy Analysis CLI
```

The interface highlights the `Start commit` button and the `Commit new file` button. The `Commit new file` dialog is open, showing the file name `Create codacy-analysis.yml` and the user `petr@svenda.com`. The `Commit directly to the main branch` option is selected.

Prepare repo content

- Locally on your PC
- Clone repository on your PC
 - GitHub Desktop File → Clone
 - git checkout your_repository.git
- Copy example buggy code into your repo and commit
 - IS → Study materials, buggycode.zip
 - Commit new files, push to repo (Push origin)



Analyze results I.

- Observe scheduled, running and finished actions
- Online at github.com
- *Github* → *Repo* → *Actions*
- Re-run jobs if desired
 - Done on same commit!
 - Useful if Action failed due to external service

The screenshot shows the GitHub interface for the repository 'petrs / pv080_test_cpp'. The 'Actions' tab is selected and highlighted with a red box. Below the repository name, there is a 'Merge branch 'main' of https://github.com/petrs/pv08...' commit. A 'Re-run jobs' button is also highlighted with a red box. The 'Codacy Security Scan' job is shown as successful, with a blue bar and a red box around the job name. The job details are expanded, showing the following steps:

- Set up job (10s)
- Checkout code (1s)
- Run Codacy Analysis CLI (45s)

The logs for the 'Run Codacy Analysis CLI' step are visible, showing the command and output:

```

55 ▶ Run codacy/codacy-analysis-cli-action@1.1.0
56 --2020-12-02 14:06:44-- https://raw.githubusercontent.com/codacy/codacy-analysis-cli/4.0.0/bin/codacy-analysis-cli.sh
57 Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.248.133
58 Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.248.133|:443... connected.
59 HTTP request sent, awaiting response... 200 OK
60 Length: 3107 (3.0K) [text/plain]
61 saving to: 'STDOUT'
62
63      OK ...
64
65      100% 48.7M=0s
66
67 2020-12-02 14:06:45 (48.7 MB/s) - written to stdout [3107/3107]
68
69

```

Analyze results II.

- Online at github.com
- *Github* → *Repo* → **Security**
 - When actions are finished
- **Code scanning alerts**
 - Sorted by tool (e.g., Cppcheck)
- **Shown similarly to Issues**
 - Open, Closed
 - Can be filtered (severity...)
 - But visible only to repo developers

The screenshot shows the GitHub interface for the repository 'petrs / pv080_test_cpp'. The 'Security' tab is selected and highlighted with a red box, showing 60 alerts. The 'Code scanning' section is also highlighted with a red box, displaying a summary of 54 Open and 0 Closed alerts. A list of specific alerts is shown below, with one alert highlighted in a red box: 'Array 'Stack[5]' accessed at index 100, which is out of bounds. file:///codacy/fail.cpp#L8 • Detected 5 minutes ago'. Other alerts include 'Array index out of bounds; 'arrayStack' buffer size is 20 and it is accessed at offset 400.', 'Array 'Static[5]' accessed at index 100, which is out of bounds.', 'Array 'Stack[5]' accessed at index 5, which is out of bounds.', 'Memory leak: data_copy', and several MISRA rules.

Notes

- Standard Issues are used to report bugs or ask for / plan enhancements and new features (usually opened manually)
- Code scanning alerts are similarly treated, but opened automatically, visible only to developers
- Results from tool(s) are transformed to standardized 'OASIS Static Analysis Results Interchange Format (SARIF) TC', which GitHub can process, and display issues based on it

Analyze results III.

- Bug triage
 - atm, bug properties cannot be changed
 - (expect UI change in future)
- Can be dismissed (=> will not be fixed)
 - E.g., if False positive, not relevant...
 - Severity is set by original tools
 - Expect unification in future
 - Dismiss only bugs you are sure about!

Array index out of bounds; 'arrayStack' buffer size is 20 and it is accessed at offset 400.

Open ⚠ Warning

branch: main ▾

file:///codacy/fail.cpp 📄

Preview unavailable
Sorry, we couldn't find this file in the repository.

Array index out of bounds; 'arrayStack' buffer size is 20 and it is accessed at offset 400.
Cplusplus (reported by Codacy)

| Tool | Rule ID |
|--------------------------------|-------------------------|
| Cplusplus (reported by Codacy) | cplusplus_ctuArrayIndex |

No rule help available for this alert.

🕒 First appeared in commit 0d0ffeb 13 minutes ago

🔗 Merge branch 'main' of https://github.com/petrs/pv080_test_cpp into main ✔️ 0d0ffeb

file:///codacy/fail.cpp#L15 on branch main

Dismiss ▾

Dismiss reason ✕

False positive
This alert is not valid

Used in tests
This alert is not in production code

Won't fix
This alert is not relevant

Fix bug(s)

- Locate reported bug in source code
 - (Note: for the moment, bug preview at Github is not working)
 - Use file and line number to locate (e.g., fail.cpp#L7 => line 7 in fail.cpp)
- Fix bug
 - E.g., Static[5]; → Static[101];
 - (Note: not proper fix, check length instead)
- Commit, Push
 - Will trigger analysis again
- Fixed issues are now in ‘Closed’ category
 - Introducing and fixing commit is visible in history

Array 'Static[5]' accessed at index 100, which is out of bounds.

Open Warning

file:///codacy/fail.cpp

Preview unavailable
Sorry, we couldn't find this file in the repository.

Array 'Static[5]' accessed at index 100, which is out of bounds.
Cppcheck (reported by Codacy)

Branch: main

Dismiss

file:///codacy/fail.cpp

Preview unavailable
Sorry, we couldn't find this file in the repository.

Array 'Static[5]' accessed at index 100, which is out of bounds.
Cppcheck (reported by Codacy)

| Tool | Rule ID |
|-------------------------------|--------------------------------|
| Cppcheck (reported by Codacy) | cppcheck_arrayIndexOutOfBounds |

No rule help available for this alert.

First appeared in commit 0d0ffeb 34 minutes ago

Merge branch 'main' of https://github.com/petrs/pv080_test_cpp into main
file:///codacy/fail.cpp#L7 on branch main

Fixed in branch main with commit 68e89f2 1 minute ago

fix

Scanning of python source code with

SCANNING OF PYTHON CODE

Setup Python actions on repo

- Find an action that will find some security issues in the uploaded code.
- Which tool have you used?
- Which issues have you found?
- If a tool does not work then use another one!

Notes

- X scan requires no special configuration (same as Codacy)
- Provides a good explanation of a bug

Bit more advanced setup, CodeQL code analysis, configurable build steps

CODE SCANNING WITH GITHUB + ACTIONS + CODEQL

CodeQL basics

- Your source code → CodeQL code → rules executed on that canonical code
 - Adding support for new language (e.g., Go) => just convert Go source code to CodeQL canonical form and then use all already existing rules
- CodeQL uses own language to write analysis rules
 - Many existing security rules are already written, you don't need to learn this language or write own rules to use it
- CodeQL is integrated in GitHub Actions or can be run for external CI
 - We will use integrated option
 - <https://docs.github.com/en/free-pro-team@latest/github/finding-security-vulnerabilities-and-errors-in-your-code/enabling-code-scanning-for-a-repository>
- Note: difference between dedicated tool (e.g., cppcheck) and CodeQL
 - Single tool for single language – detection rules must be written again for new lang
 - CodeQL – detection rules are written for canonical code, new lang requires only to write conversion between lang code and canonical code

Setup CodeQL actions on repository

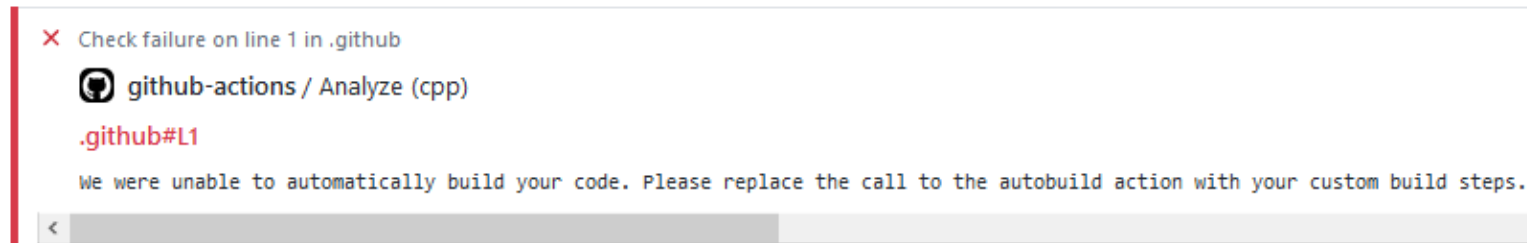
- Create new repository (e.g., pv080_test_python), clone locally
- Enable code scanning actions
 - Pick CodeQL (instead of Codacy)
- Check codeql-analysis.yml before commit
 - Modify set of target languages
 - language: ['cpp']
- Copy buggy code to repo, push

```
27 strategy:
28   fail-fast: false
29
30   matrix:
31     language: [ 'cpp', 'java', 'python' ]
32     # CodeQL supports [ 'cpp', 'csharp', 'go', 'java', 'javascript', 'py
33     # Learn more:
34     # https://docs.github.com/en/free-pro-team@latest/github/finding-seci
35
36 steps:
37   - name: checkout repository
```

The screenshot shows the GitHub interface for a repository named 'petrs/pv080_test'. The 'Security' tab is active, displaying a sidebar with navigation options: Overview, Security policy, Security advisories (0), Dependabot alerts, and Code scanning alerts. The main content area features a 'Get started with code scanning' section with the text 'Automatically detect common vulnerabilities and coding errors'. A prominent 'CodeQL Analysis' card by GitHub is highlighted with a red border, containing a description of the tool and a 'Set up this workflow' button. Below this, the 'Security analysis from the Marketplace' section lists other tools: '42Crunch API Security Audit' and 'Codacy Security Scan'.

Fixing build for CodeQL

- CodeQL Action may fail with:



- Reason
 - Analysis for some languages works on the compiled code/bytecode (e.g., Java)
 - Static analysis generally runs on unfinished code, but not always
 - One shall not commit broken code to repo anyway
- Fix: tell CodeQL how to build

Fixing build for CodeQL I.

- GitHub CodeQL tries to compile your code
 - But how it knows how to compile your project?
- Autobuild feature is only heuristic (=> can be wrong, can fail)
 - <https://docs.github.com/en/free-pro-team@latest/github/finding-security-vulnerabilities-and-errors-in-your-code/configuring-the-codeql-workflow-for-compiled-languages>
 - Depends on CI operating system
 - Search for .sln or .vcxproj (MS Visual Studio), then call MSBuild.exe
 - Search for build.bat, build.cmd, and build.exe, then run it
 - Search for Makefile, then call make
 - Starts in repo root, then try in subdirectories...
- Tip: Start with simplest example, make it work, then make more complicated

Fixing build for CodeQL II.

- The solution depends on build system for your project
 - Make, gradle, ant, maven...
 - We will only discuss simple direct build with g++ and makefile
- Option 1: Makefile into repo root (g++ fail.cpp)
 - Feel free to use improved makefile scripts
 - Generally better solution than option 2
- Option 2: Direct specification in codeql-analysis.yml
 - Disable autobuild by commenting it out with #
 - Insert conditional statement based on language
 - Example here for cpp and java
 - Python is left with autobuild
 - More flexibility in configuration, more changes to scripts

```
main:
  g++ ./fail.cpp
```

```
50 # Autobuild attempts to build any compiled languages (C/C++, C#, or Java).
51 # If this step fails, then you should remove it and run the build manually (see below)
52 #- name: Autobuild
53 # uses: github/codeql-action/autobuild@v1
54
55 - if: matrix.language == 'cpp'
56   name: Build cpp
57   run: |
58     g++ ./fail.cpp
59
60 - if: matrix.language == 'java'
61   name: Build Java
62   run: |
63     ant -f ./build.xml compile
64
65 - if: matrix.language == 'python'
66   name: Build Python
67   uses: github/codeql-action/autobuild@v1
```


Notes

- The goal of this exercise is to show that the configuration can be hard.
- In this seminar finding the issues is not important with CodeQL.
- More at code review seminar.

Setup Action to observe new vulnerabilities in your dependencies, notify you and even propose automatic patch

CHECKING SECURITY OF DEPENDENCIES GITHUB + DEPENDABOT

Enable dependabot

- Enable Dependabot alerts
 - You will receive notification about vulnerable dependency
- Enable Dependabot security updates
 - You will receive automatic pull requests fixing vulnerable dependency
 - Always analyze automatic pull requests for correctness

The screenshot shows the 'Configure security and analysis features' page in a GitHub repository. On the left is a sidebar menu with options: Options, Manage access, Security & analysis (highlighted), Branches, Webhooks, Notifications, Integrations, and Deploy keys. The main content area has a title 'Configure security and analysis features' and a subtitle 'Security and analysis features help keep your repository secure and updated. By enabling these features, you're granting us permission to perform read-only analysis on your repository.' Below this are three sections: 'Dependency graph' (with a 'Disable' button), 'Dependabot alerts' (with an 'Enable' button, highlighted by a red box), and 'Dependabot security updates' (with an 'Enable' button, also highlighted by a red box).

The screenshot shows a GitHub pull request titled 'Bump junit from 4.12 to 4.13.1 in /CryptoOperationsExtractor #1'. It is marked as 'Merged' and shows the commit history. A yellow banner indicates 'This automated pull request fixes a security vulnerability'. Below this, there are statistics for conversation, commits, checks, and files changed. The pull request description includes the commit message 'Bump junit from 4.12 to 4.13.1 in /CryptoOperationsExtractor' and lists the changes: bumping junit from 4.12 to 4.13.1, updating release notes, and updating the changelog. The pull request is signed-off by 'dependabot[bot]' and is verified. The diff view shows the change in the 'pom.xml' file, where the version of junit is updated from 4.12 to 4.13.1.

Notes

- Dependabot is well established feature of GitHub
- GitHub checks for vulnerabilities in major libraries (dependencies) and notify you if your repo use it
- If you enable it for a project without dependencies then not much will happen.
- You can try to create a repository with a pom file with a vulnerable version of the library, but that is extra task.

Run tools (e.g., cppcheck) locally without Github Actions. Suitable for projects with proprietary code, troubleshooting, execution with non-standard parameters etc.

RUNNING TOOL(S) LOCALLY

Cppcheck for C++ files

- For small files, you may try cppcheck online
 - <https://cppcheck.sourceforge.net/demo/>
 - Paste fail.cpp into browser and Check
 - Compare with errors as reported by Codacy
- Run cppcheck from command line
 - Get latest release
 - <https://github.com/danmar/cppcheck/releases>
 - Run `cppcheck --enable=all fail.cpp`
- Run cppcheck via GUI
 - Allows for analysis of folders, sorting by severity...

Online Demo

Enter code: (max 1024 characters)

```
#include <jstream>
int Static[5];
int memcheckFailDemo(int* arrayStack, unsigned int arrayStackLen,
int* arrayHeap, unsigned int arrayHeapLen) {
int Stack[5];

Static[100] = 0;
Stack[100] = 0;

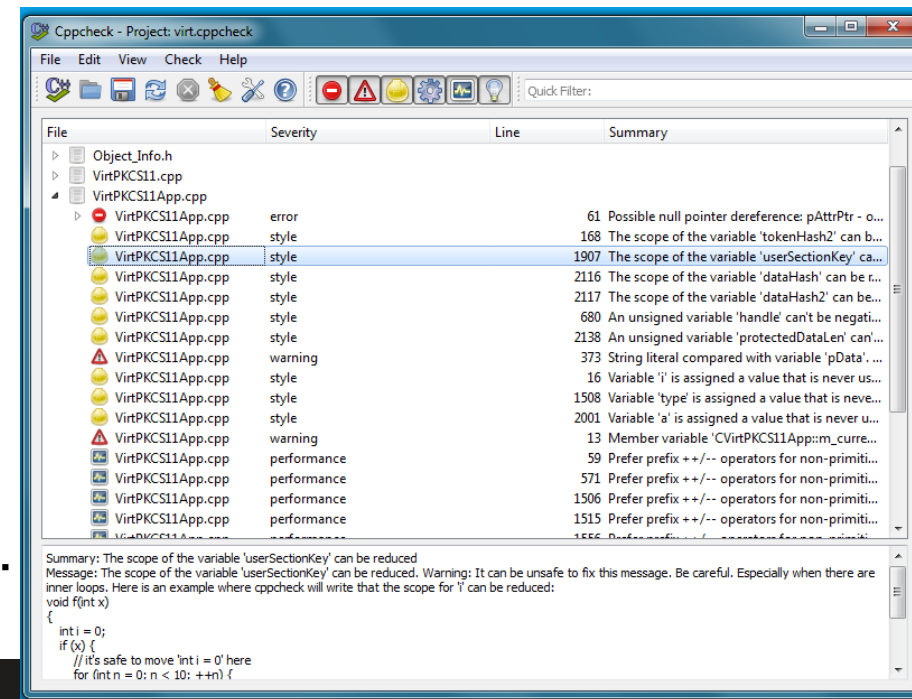
for (int i = 0; i <= 5; i++) Stack [i] = 0;

int* array = new int[5];
array[100] = 0;

arrayStack[100] = 0;
arrayHeap[100] = 0;

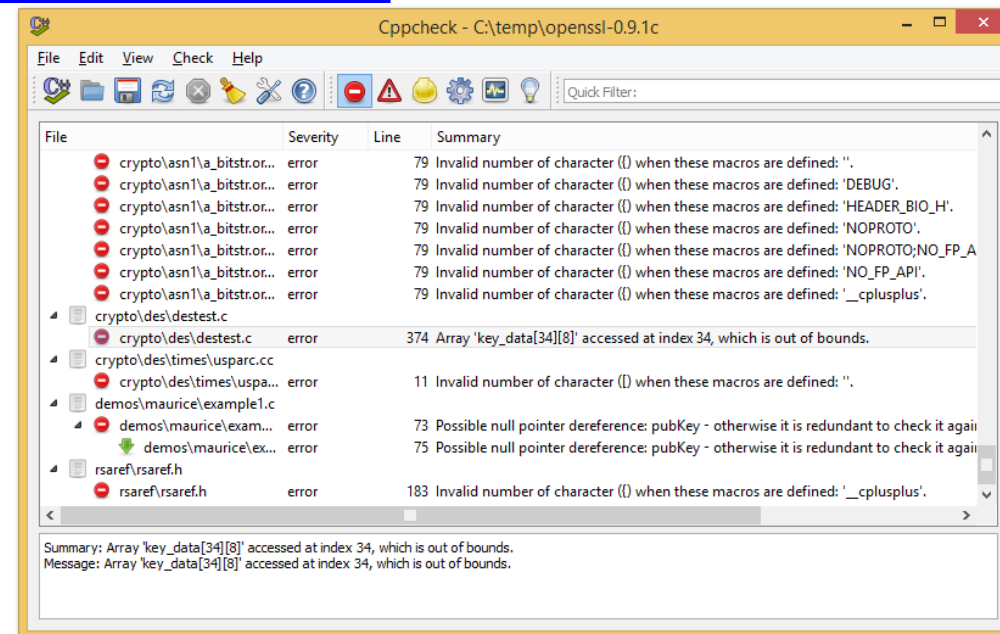
for (unsigned int i = 0; i <= arrayStackLen; i++) {
arrayStack[i] = 0;
}
for (unsigned int i = 0; i <= arrayHeapLen; i++) {
```

Check



CPPCheck + OpenSSL

- First run it against buggy code
- Second run against some old OpenSSL0.9.xx (around 1998)
 - <https://packetstormsecurity.com/crypt/SSL/openssl/page5/>
 - Or a bit newer: <https://www.openssl.org/source/old/0.9.x/>
 - It might take much time.
 - What are the bugs?
- Run against newest OpenSSL
 - <ftp://ftp.openssl.org/source/>
 - Why not completely clean yet?



Hearthbleed bug



- OpenSSL 1.0.1 through 1.0.1f
- Download <https://www.openssl.org/source/openssl-1.0.1e.tar.gz>
- Locate function `dtls1_process_heartbeat(SSL *s)`
 - `SsLt1_lib.c`
- Will your static analyzers find anything?
 - Don't be sad, even Coverity didn't before the bug was exposed
 - <http://security.coverity.com/blog/2014/Apr/on-detecting-heartbleed-with-static-analysis.html>

PREfast - Microsoft static analysis tool

BufferOverflow - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Local Windows Debugger - Auto

Code Analysis (Global Scope)

Server Explorer Toolbox

All Projects (3) All Results (3)

C6386 Write overrun
Buffer overrun while writing to 'userName': the writable size is '8' bytes, but '4294967295' bytes might be written.

Line Explanation
16 'userName' is an array of 8 elements
32 Invalid write to 'userName[4294967295]

More information
bufferoverflow.cpp (Line 32)
Warning Actions

C6386 Write overrun
bufferoverflow.cpp (Line 37)

C6011 Dereferencing null pointer
bufferoverflow.cpp (Line 106)

```
void demoBufferOverflowData() {
    int unused_variable = 3;
    #define NORMAL_USER 'n'
    #define ADMIN_USER 'a'
    int userRights = NORMAL_USER;
    #define USER_INPUT_MAX_LENGTH 8
    char userName[USER_INPUT_MAX_LENGTH];
    char passwd[USER_INPUT_MAX_LENGTH];

    // print some info about variables
    printf("%-20s: %p\n", "userName", userName);
    printf("%-20s: %p\n", "passwd", passwd);
    printf("%-20s: %p\n", "unused_variable", &unused_variable);
    printf("%-20s: %p\n", "userRights", &userRights);
    printf("%-20s: %p\n", "demoBufferOverflowData", demoBufferOverflowData);
    printf("\n");

    // Get user name
    memset(userName, 1, USER_INPUT_MAX_LENGTH);
    memset(passwd, 2, USER_INPUT_MAX_LENGTH);
    printf("login as: ");
    fflush(stdout);
    gets(userName);
}
```

ANALYZE WINDOW HELP

- Start Performance Analysis Alt+F2
- Start Performance Analysis Paused Ctrl+Alt+F2
- Launch Performance Wizard...
- Compare Performance Reports...
- Profiler
- Concurrency Visualizer
- JavaScript Analysis
- Run Code Analysis on Solution Alt+F11
- Configure Code Analysis for Solution**
- Run Code Analysis on Only BufferOverflow
- Configure Code Analysis for BufferOverflow
- Calculate Code Metrics for Selected Project(s)
- Calculate Code Metrics for Solution
- Windows

PREfast – example buggycode

The image shows a screenshot of Visual Studio Code with a code analysis window on the left and a C++ source file on the right.

Code Analysis Window (Left):

- Code Analysis: Analyze Search
- All Projects (3) All Results (3)
- C6386 Write overrun**
Buffer overrun while writing to 'userName': the writable size is '8' bytes, but '4294967295' bytes might be written.
Line Explanation
16 'userName' is an array of 8 elements (8 bytes)
32 Invalid write to 'userName[4294967294]', (writable range is 0 to 7)
[More information](#)
bufferoverflow.cpp (Line 32)
Warning Actions
- C6386 Write overrun**
bufferoverflow.cpp (Line 37)
- C6011 Dereferencing null pointer**
bufferoverflow.cpp (Line 106)

Source File: BufferOverflow.cpp (Global Scope)

```
#define ADMIN_USER 'a'
int userRights = NORMAL_USER;
#define USER_INPUT_MAX_LENGTH 8
char userName[USER_INPUT_MAX_LENGTH];
char passwd[USER_INPUT_MAX_LENGTH];

// print some info about variables
printf("%-20s: %p\n", "userName", userName);
printf("%-20s: %p\n", "passwd", passwd);
printf("%-20s: %p\n", "unused_variable", &unused_variable);
printf("%-20s: %p\n", "userRights", &userRights);
printf("%-20s: %p\n", "demoBufferOverflowData", demoBufferO
printf("\n");

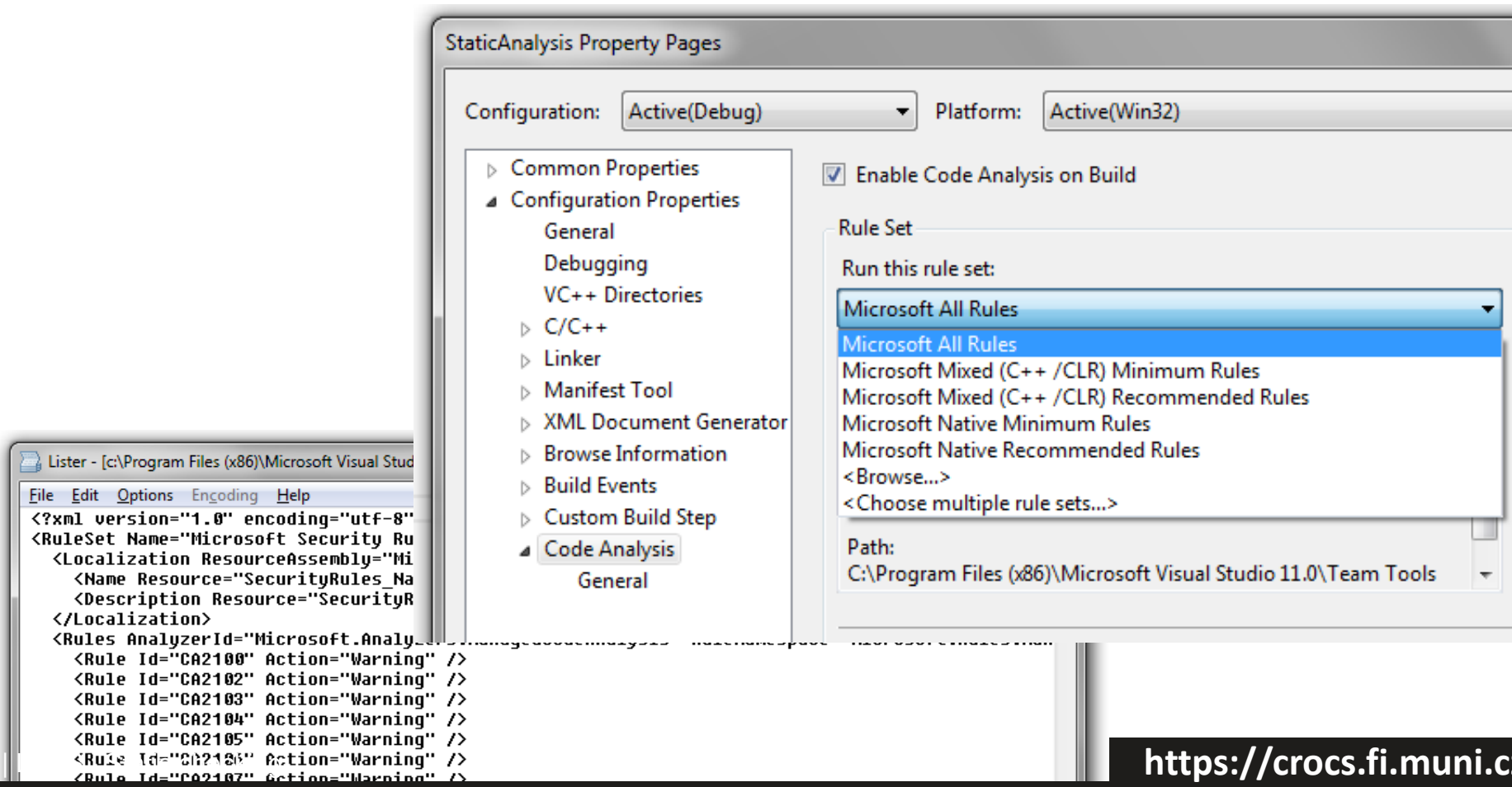
// Get user name
memset(userName, 1, USER_INPUT_MAX_LENGTH);
memset(passwd, 2, USER_INPUT_MAX_LENGTH);
printf("login as: ");
fflush(stdout);
gets(userName);
```

PREfast – what can be detected

- Potential buffer overflows
- Memory leaks, uninitialized variables
- Excessive stack usage
- Resources – release of locks...
- Incorrect usage of selected functions
- List of all code analysis warnings <http://msdn.microsoft.com/en-us/library/a5b9aa09.aspx>

PREfast settings

- <http://msdn.microsoft.com/en-us/library/ms182025.aspx>



FindBugs/FindSecurityBugs - Java

- Download Eclipse
- Download FindBugs <http://findbugs.sourceforge.net/> as a plugin to eclipse
- Download FindSecurityBugs (plugin to FindBugs plugin)
 - <https://find-sec-bugs.github.io/>
- Run FindBugs in Eclipse
 - Ask me for help

FindBugs/FindSecurityBugs - Java

- Note: you need compiled *.jar for analysis
 - And source code for quick display of problems 😊
 - `import com.google.common.io.BaseEncoding;`
 - `import org.slf4j.Logger;`
 - `import org.slf4j.LoggerFactory;`
- Extract content of IS → [crypto-java.zip](#)
- Run FindBugs
- Ask me in case of issues 😊

Discussion

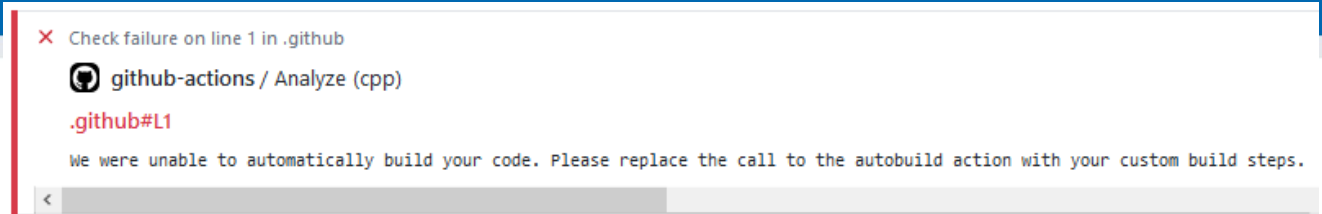
- Can you find false positive?
- Every student: name and describe the most severe bug you found

Final Questions

- What was the most severe issue that you found?
- What tool was the best for you?
- What are differences between the tools that you used?
- What more would you like from your static tools?

Some hints on common issues

TROUBLESHOOTING



Troubleshooting

- Analysis is not finished yet
 - Wait an hour, try to make another bogus commit (update file)
- Start from small working examples, then extend to larger project
 - E.g., simple main.java, only later large java project via ant
- Analyze failed to start for specific language
 - GitHub Actions usually requires code to be compilable
 - Analysis for some languages works on the compiled code/bytecode (e.g., Java)
 - (static analysis runs on unfinished code, but one shall not commit broken code to repo)
 - Github will invoke autobuild feature
 - Tries to build various languages as defined here
 - <https://docs.github.com/en/free-pro-team@latest/github/finding-security-vulnerabilities-and-errors-in-your-code/configuring-the-codeql-workflow-for-compiled-languages>
- Paths case sensitivity
 - Linux is case-sensitive for path names while Windows isn't
 - /java/ and /Java/ are the same on Windows, but not on Linux
- Clicking on log of 'Perform Code QL Analysis' shows nothing
 - Likely GitHub bug, click left on the Analyze (language), then again on 'Perform Code QL Analysis'
- Makefile requires tabs, not spaces

Some tips

- Setup scanning tools at the beginning of new project
 - And make sure all bugs are always fixed (similar to “compile cleanly” mantra)
- Look at the text logs produced by actions (click on named Action)
 - What tool was executed, what configuration...

**NO HOMEWORK ASSIGNMENT THIS
WEEK 😊**

CHECK-OUT



symmetry.physio

Checkout

- Which of the seminar parts you enjoyed most?
- Write three items you liked (ideally inserted as single word each)
- Write to sli.do when displayed

**THANK YOU FOR COMING, SEE YOU
NEXT WEEK**