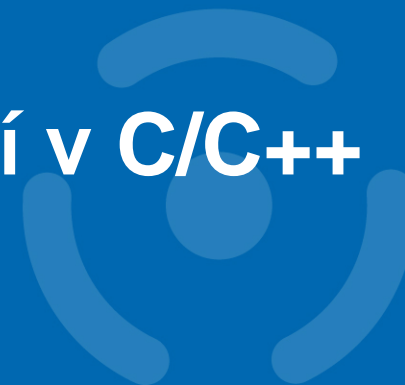


# PB173 - Tématický vývoj aplikací v C/C++ (podzim 2013)



Skupina: [Aplikovaná kryptografie a bezpečné programování](#)

<https://is.muni.cz/auth/el/1433/podzim2013/PB173/index.qwarp?fakulta=1433;obdobi=5983;predmet=734514;prejit=2957738;>

Petr Švenda [svenda@fi.muni.cz](mailto:svenda@fi.muni.cz)

Konzultace: G.201, Úterý 13-13:50



Centre for Research on  
Cryptography and Security

# Security code review

- Architecture overview
  - Design choices and possible design flaws
- Code review
  - How well is architecture actually implemented
- Whitebox, greybox & blackbox testing
  - different level of access to code and documentation
- Available tools
  - mainly for code review

## Security code review (2)

- You will always have a limited time
  - try to rapidly build overall picture
  - use tools to find low hanging fruit
- Focus on most sensitive and problematic areas
  - use tools to focus your analysis scope
- More eyes can spot more problems
  - experts on different areas

# Architecture overview

## Architecture overview

- Get all information you can quickly
- Assets
  - What has the value in the system?
  - What damage is caused when successfully attacked?
  - What mechanisms are used to protect assets?
- Roles
  - Who has access to what?
  - What credentials needs to be presented?
- Thread model
  - What is expected to do harm?
  - What are you defending against?

## Architecture overview (2)

- Usage of well established techniques and standards
- Comparison with existing schemes
  - What is the advantage of new scheme?
  - Why changes were made?
- Security tradeoffs documented
  - Possible threat, but unmitigated?
  - Is documented or overlooked?

## Sensitive data flow mapping

- Identify sensitive data
  - password, key, protected data...
- Find all processing functions
  - and focus on them
- Create data flow between functions
  - e.g. Doxygen call graph
- Inspect when functions can be called
  - Is key schedule validity checked?
  - Can be function called without previous function calls?
- Where are sensitive data stored between calls?

## Protocol design (and implementation)

- Packet confidentiality, integrity and authenticity
- Packet removal/insertion detection
- Replay attack
- Reflection attack
- Man in the middle



# Code overview

# Cryptography usage

- CIA (Confidentiality, Integrity, Availability)
  - Plaintext data over insecure channel? Encrypted only?
  - Can be packet send twice (replay)?
  - What is the application response on data modification?
- What algorithms are used
  - Broken/insecure algorithms? MD5? simple DES?
- What key lengths are used?
  - < 90 bits symmetric crypto?
  - < 1024 bits asymmetric crypto?
- Random number generation
  - Where the key comes from?
  - Is source entropic enough?
  - srand() & rand()?

## Cryptography usage (2)

- Key creation
  - Where the keys originate? Enough entropy?
  - Who has access?
- Key storage
  - Hard-coded keys
  - Keys in files in plaintext
  - Keys over insecure channels
  - Keys protected by less secure keys
- Key destruction
  - How are keys erased from memory?
  - Can exception prevent key erase?

# Cryptography implementation

- Implementation from well known libraries?
- Own algorithms?
  - security by obscurity?
  - usually not secure enough
- Own modifications?
  - Why?
  - sometimes used to prevent compatible programs
  - decreased number of rounds?
  - Performance optimization with security impact?

## Code inspection

- Overall code logic
- Memory management - allocation, input validation
- String operations – copy, concatenate, string termination
- Data flow – conditional jumps, test of return values
- Race conditions (TOCTOU)

## Input validation

- Hard (and expensive) to do right
- Always use white-listing (what is allowed), not black listing (what is banned)
- Check for buffer overruns
  - functions called with attacker's input
  - dangerous functions (strcpy...)
  - arrays with fixed lengths
- Large inputs in general
  - try to insert 1KB of text instead of user name
- Fuzzing
  - large amount of automated inputs with different length

## Recommended reading

- Process of security code review
  - <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01668009>
- Why cryptosystems fail, R. Anderson
  - <http://www.cl.cam.ac.uk/~rja14/Papers/wcf.pdf>
- Software Security Code Review
  - <http://www.softwaremag.com/l.cfm?doc=2005-07/2005-07code>
- Static code analysis tools
  - [http://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)
- Security in web applications (OWASP)
  - [http://www.owasp.org/index.php/Code\\_Review\\_Introduction](http://www.owasp.org/index.php/Code_Review_Introduction)

## Static analysis tools

- List of static checkers
  - <http://spinroot.com/static/>
  - [http://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)
  - [https://security.web.cern.ch/security/recommendations/en/code\\_tools.shtml](https://security.web.cern.ch/security/recommendations/en/code_tools.shtml)
- We will be interested in C/C++ checkers
  - but tools exists for almost any language



## Both free and commercial tools

- Commercial tools
  - PC-Lint (Gimpel Software)
  - Klocwork Insight (Klocwork)
  - Coverity Prevent (now under HP)
  - Microsoft PRefast (included in Visual Studio)
- Free tools
  - Rough Auditing Tool for Security (RATS) <http://code.google.com/p/rough-auditing-tool-for-security/>
  - CppCheck <http://cppcheck.sourceforge.net/>
  - Flawfinder <http://www.dwheeler.com/flawfinder/>
  - Splint <http://www.splint.org/>
  - FindBugs <http://findbugs.sourceforge.net> (for Java programs)
  - Doxygen's call graphs from source <http://www.stack.nl/~dimitri/doxygen/>
  - ...

# Cppcheck



- A tool for static C/C++ code analysis
  - Open-source freeware, <http://cppcheck.sourceforge.net/>
- Last version 1.61 (2013-08-03)
- Used to find bugs in open-source projects (Linux kernel... )
- Command line & GUI version
- Standalone version, plugin into IDEs, version control...
  - Code::Blocks, Codelite, Eclipse, Jenkins...
  - Tortoise SVN
  - not Visual Studio ☹
- Cross platform (Windows, Linux)
  - `sudo apt-get install cppcheck`

## Cppcheck – what is checked?

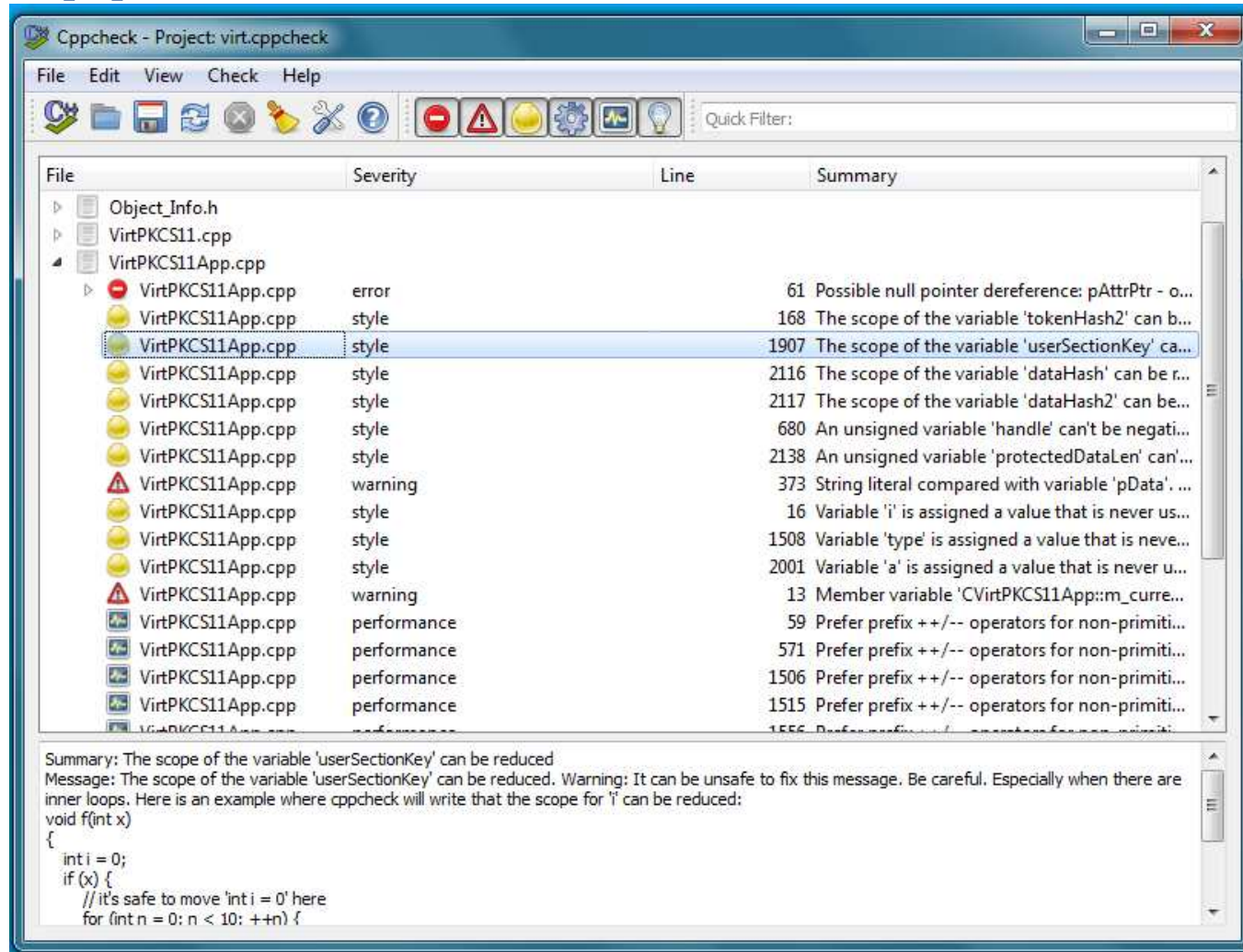
- Bound checking for array overruns
- Suspicious patterns for class
- Exceptions safety
- Memory leaks
- Obsolete functions
- sizeof() related problems
- String format problems...
- See full list

[http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main\\_Page#Checks](http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main_Page#Checks)

## Cppcheck – categories of problems

- **error** – when bugs are found
- **warning** - suggestions about defensive programming to prevent bugs
- **style** - stylistic issues related to code cleanup (unused functions, redundant code, constness...)
- **performance** - suggestions for making the code faster.
- **portability** - portability warnings. 64-bit portability. code might work different on different compilers. etc.
- **information** - Informational messages about checking problems

# Cppcheck



# 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 BufferOverflow.cpp (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);
}
```

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 - Microsoft static analysis tool

- Visual Studio Ultimate and Premium Editions
- Documentation for PREfast
  - <http://msdn.microsoft.com/en-us/library/windows/hardware/gg487351.aspx>
- PREfast tutorial
  - <http://www.codeproject.com/Articles/167588/Using-PREfast-for-Static-Code-Analysis>
  - <http://www.cs.auckland.ac.nz/~pgut001/pubs/sal.html>
- Can be enabled on every build
  - not enabled by default, time consuming
- Can be extended by source code annotation (SAL)

# PREfast – example bufferOverflow

The screenshot displays the Visual Studio Code interface. On the left, the Code Analysis window shows a warning for a buffer overflow. The warning text is: "Buffer overrun while writing to 'userName': the writable size is '8' bytes, but '4294967295' bytes might be written." Below this, a "Line Explanation" section shows: "16 'userName' is an array of 8 elements (8 bytes)" and "32 Invalid write to 'userName[4294967294]', (writable range is 0 to 7)".

On the right, the BufferOverflow.cpp file is open, showing the following code:

```
#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);
```

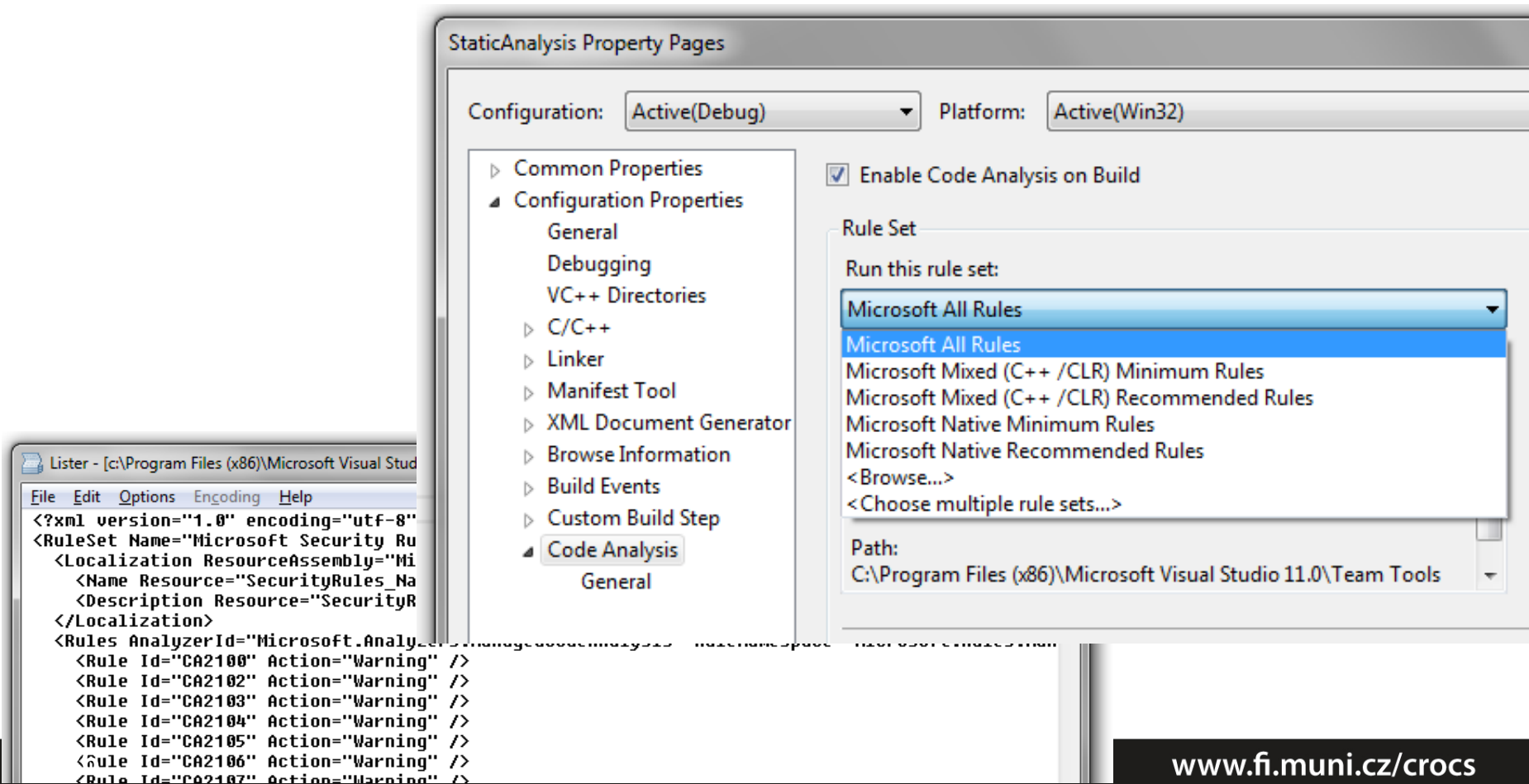


## PREfast – what can be detected

- 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 (VS 2012)

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



The screenshot displays the 'StaticAnalysis Property Pages' dialog box in Visual Studio 2012. The 'Configuration' is set to 'Active(Debug)' and the 'Platform' is 'Active(Win32)'. The 'Code Analysis' section is expanded, and the 'Rule Set' dropdown is open, showing 'Microsoft All Rules' selected. The 'Path' is set to 'C:\Program Files (x86)\Microsoft Visual Studio 11.0\Team Tools'. A small window in the background shows XML code for the rule set configuration.

```
<?xml version="1.0" encoding="utf-8"
<RuleSet Name="Microsoft Security Ru
<Localization ResourceAssembly="Mi
  <Name Resource="SecurityRules_Na
  <Description Resource="SecurityR
</Localization>
<Rules AnalyzerId="Microsoft.Analyz
  <Rule Id="CA2100" Action="Warning" />
  <Rule Id="CA2102" Action="Warning" />
  <Rule Id="CA2103" Action="Warning" />
  <Rule Id="CA2104" Action="Warning" />
  <Rule Id="CA2105" Action="Warning" />
  <Rule Id="CA2106" Action="Warning" />
  <Rule Id="CA2107" Action="Warning" />
```



## Practical assignment

- Every team will make its own documentation & code available online
  - upload to IS repository (available to others)
  - deadline 13.11. 12:00
- Other teams will make security analysis of the architecture and code (2 projects)
  - after 13.11. 12:00
- Points will be awarded according to:
  - number&severity of problems found in reviewed projects
  - quality of own architecture and code

# Practical assignment

- Some tips what to analyze:
  - which functions are manipulating with sensitive information
  - where is random numbers coming from
  - code bugs?
- Use some analysis tools
  - gcc -Wall -Wextra
  - MSVS:Project→C/C++ →General →Warning level (/W4 /Wall)
  - call graphs (e.g., Doxygen, <http://cecko.eu/public/doxygen>)
  - Cppcheck (C/C++, Windows) <http://cppcheck.sourceforge.net/>
  - ...

## Practical assignment (2)

- Summarize your findings
  - problem identification + severity + applicability + short description
  - 2 pages enough (per project)

**Identifikace problému:** A\_x (celková bezpečnostní architektura) / C\_x (kód implementace)

**Závažnost:** nízká / střední / vysoká / není možné rozhodnout

**Proveditelnost útoku:** snadná (lze přímo externím útočníkem) / v závislosti na dalších součástech systému / není možné rozhodnout (obvykle značí potenciální zranitelnost, kde ale detailní postup pro možné zneužití přímo neznáme)

**Popis problému:** místo výskytu v kódu ve tvary soubor.c:číslo\_řádku:funkce – popis

**Navrhované řešení:** jednoduchý popis (v případě, že jsme návrh schopni poskytnout)