PA193 Secure coding principles and practices

Overview of the subject

Zdeněk Říha & Petr Švenda

CROCS
Centre for Research on
Cryptography and Security

PA193 Secure coding principles and proposal

- Relatively new subject
  - Introduced in September 2013
- Secure coding
  - How to write code in a secure way
  - So that the program cannot be attacked/exploited
  - ≠ Programming of security applications
- 2/2/2
  - Lecture: 2 hours weekly
  - Seminar: 2 hours weekly (2 seminar groups)
  - Homework: about 3-6 hours weekly

Aims of the subject

- To learn how to program in a way that the resulting application is more secure
  - Free from security related bugs
  - Cannot be attacked/exploited
- To understand security consequences of decisions made by programmer
- Many issues are independent on programming language
- Most examples are based on C/C++ and Java

Position of PA193 among other subjects

- PV079 – Applied cryptography
  - Practical aspects of cryptography
- PV181 – Laboratory on Security and Applied Cryptography
  - Using common crypto libraries and smart cards
- PA018 Advanced Topics in Information Technology Security
  - Practical project
- PA168 Postgraduate seminar on IT security and cryptography
  - Discussions on current issues of IT Security
- PB173 Domain specific development in C/C++
  - Group focused on implementation security and applied crypto

Requirements

- Basic knowledge of (applied) cryptography and IT security
  - symmetric vs. asymmetric cryptography, PKI
  - block vs. stream ciphers and usage modes
  - hash functions
  - random vs. pseudorandom numbers
  - basic cryptographic algorithms (AES, DES, RSA, EC, DH)
  - risk analysis
- Practical experience in programming with C/C++ language
- Basic knowledge in formal languages and compilers
- User-level experience with Windows and Linux OS

Organization

- Lectures + seminars + homeworks + project + exam
- Homeworks
  - assigned every second week/seminar
  - individual work of each student
  - expected workload: 3-6 hours
- Project
  - groups of 2-3 students
  - divided into three parts with 2 different deadlines
  - topic assigned in first half of semester
  - project defense in mid-term and last seminar of the term
  - expected workload: 20 hours/project/participant

Grading

- Points
  - Homework (30)
  - Project (30)
  - Written exam (90)
- Grading
  - A ≥ 90% of maximum number of points
  - B ≥ 80% of maximum number of points
  - C ≥ 70% of maximum number of points
  - D ≥ 60% of maximum number of points
  - E ≥ 50% of maximum number of points
  - F < 50% of maximum number of points

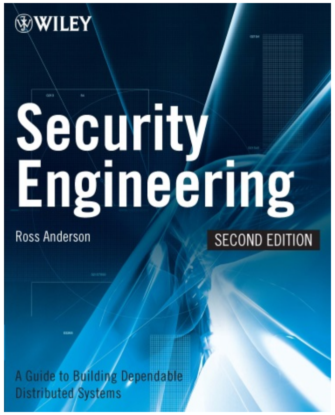
Attendance

- Lectures
  - Attendance not obligatory, but highly recommended
  - Not recorded
- Seminars
  - Attendance obligatory
  - Absences must be excused at the department of study affairs
  - 2 absences are ok
- Homeworks and projects
  - Done during students free time (e.g. at the dormitory)
  - Access to our lab is possible

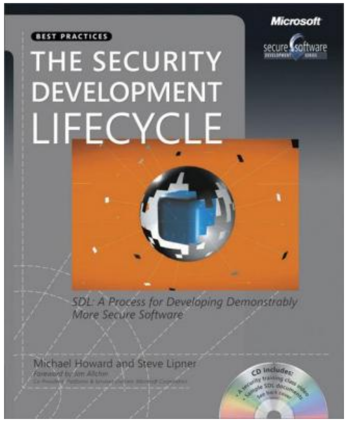
Course resources

- Lectures (PDF) available in IS
  - IS = Information System of the Masaryk University
- Homeworks/assignments available in IS
  - Submissions also done via IS
- Additional tutorials/papers/materials from time to time will also be provided in IS
  - To better understand the issues discussed
- Recommended literatures
  - To learn more …

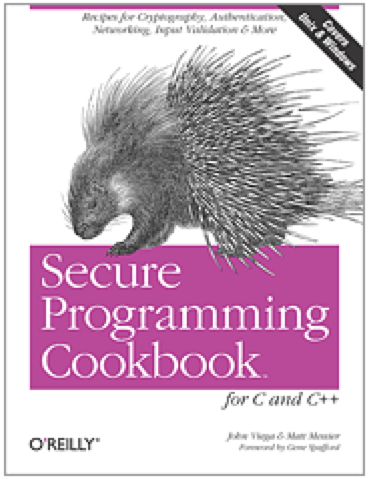
Recommended literature



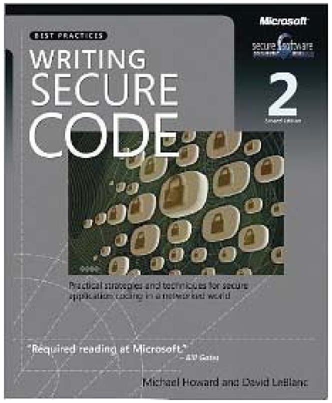
- Ross Anderson - Security engineering, Wiley


BEST PRACTICES
Microsoft
secure software
THE SECURITY DEVELOPMENT LIFECYCLE
SDL: A Process for Developing Demonstrably More Secure Software
Michael Howard and Steve Lipner

- Michael Howard, Steve Lipner - Secure Development Lifecycle, MS Press
- John Viega, Matt Messier - Secure programming cookbook, O'Reilly


Recipes for Cryptography, Authentication, Networking, Input Validation & More
Secure Programming Cookbook
for C and C++
O'REILLY
John Viega & Matt Messier

- Michael Howard - Writing secure code, MS Press


BEST PRACTICES
Microsoft
secure software
WRITING SECURE CODE
2
Practical strategies and techniques for secure application coding in a networked world
"Required reading at Microsoft."
Michael Howard and David LeBlanc

Plagiarism

- Homeworks
  - Must be worked out independently by each student
- Projects
  - Must be worked out by a team of 3 students
  - Every team member must show his/her contribution
- Plagiarism, cut&paste, etc. is not tolerated
  - Plagiarism is use of somebody else words/programs or ideas without proper citation
  - IS helps to recognize plagiarism
  - If plagiarism is detected student is assigned -5 points
  - In more serious cases the Disciplinary committee of the faculty will decide

Topics covered (order is not fixed)

1. Language level vulnerabilities: Buffer overflow, type overflow, …
2. Defence in depth, …
3. Input processing (all input is evil …)
4. (Automatic) Code checking
5. Security testing: blackbox vs. whitebox testing, fuzzing, …
6. Access control, privilege separation, …
7. Automata based programming, securing API, …

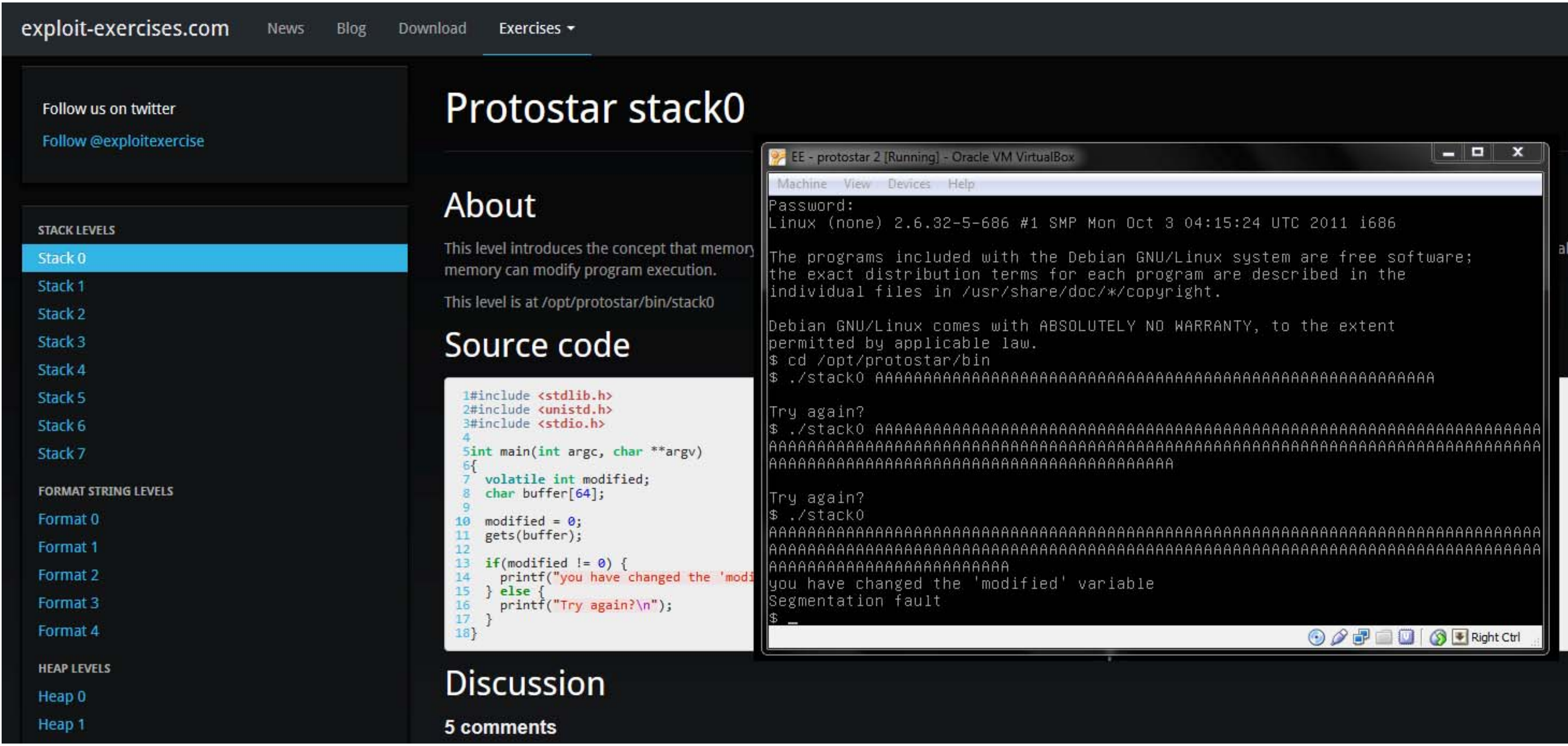
Topics covered

8. Integrity of modules, parameters, temp files, …
9. Concurrent issues: IPC, race conditions, Valgrind, …
10. (Pseudo)random numbers, their generation and usage, …
11. Security primitives: secure channel, secure storage, key management, …
12. Security code review

Labs - organization

- Dedicated teaching room in the security laboratory (A403)
- Pre-prepared environments (Windows, Linux)
  - – compilers, analyzers...
- Virtual images for selected exercises
  - – can be used also outside laboratory
- Necessary software available for students
  - – freeware tools preferred for easy home-use

Protostar virtual image with exercises



Follow us on twitter

Follow @exploitexercise

STACK LEVELS

- Stack 0
- Stack 1
- Stack 2
- Stack 3
- Stack 4
- Stack 5
- Stack 6
- Stack 7

FORMAT STRING LEVELS

- Format 0
- Format 1
- Format 2
- Format 3
- Format 4

HEAP LEVELS

- Heap 0
- Heap 1

Protostar stack0

About

This level introduces the concept that memory
memory can modify program execution.

This level is at /opt/protostar/bin/stack0

Source code

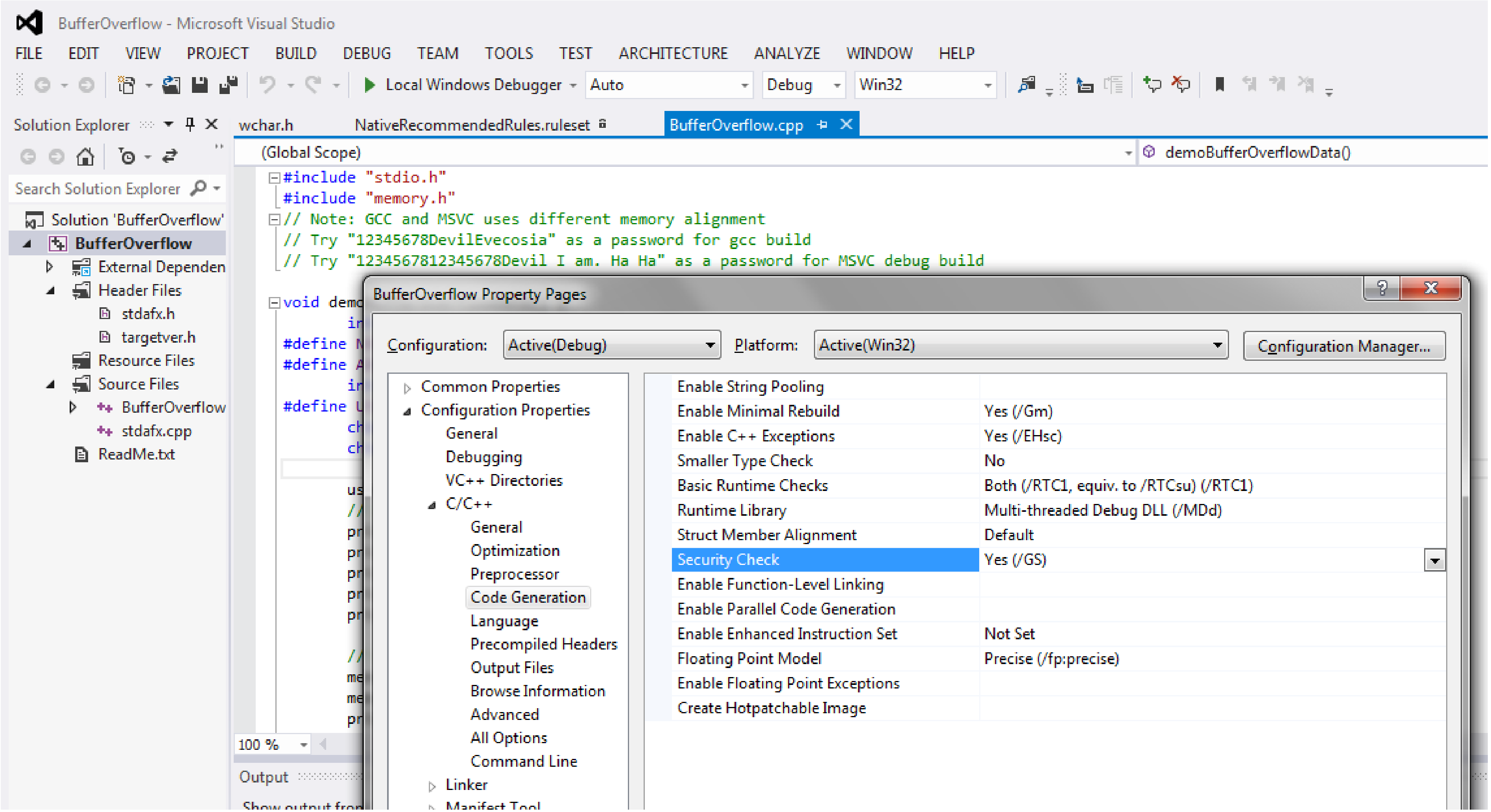
```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
  volatile int modified;
  char buffer[64];

  modified = 0;
  gets(buffer);

  if(modified != 0) {
    printf("you have changed the 'modi
  } else {
    printf("Try again?\n");
  }
}
```

Discussion

5 comments

```
Password:
Linux (none) 2.6.32-5-686 #1 SMP Mon Oct 3 04:15:24 UTC 2011 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
$ cd /opt/protostar/bin
$ ./stack0 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Try again?
$ ./stack0 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Try again?
$ ./stack0
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
you have changed the 'modified' variable
Segmentation fault
$ _
```

Compiler settings for /DEP and /ASLR

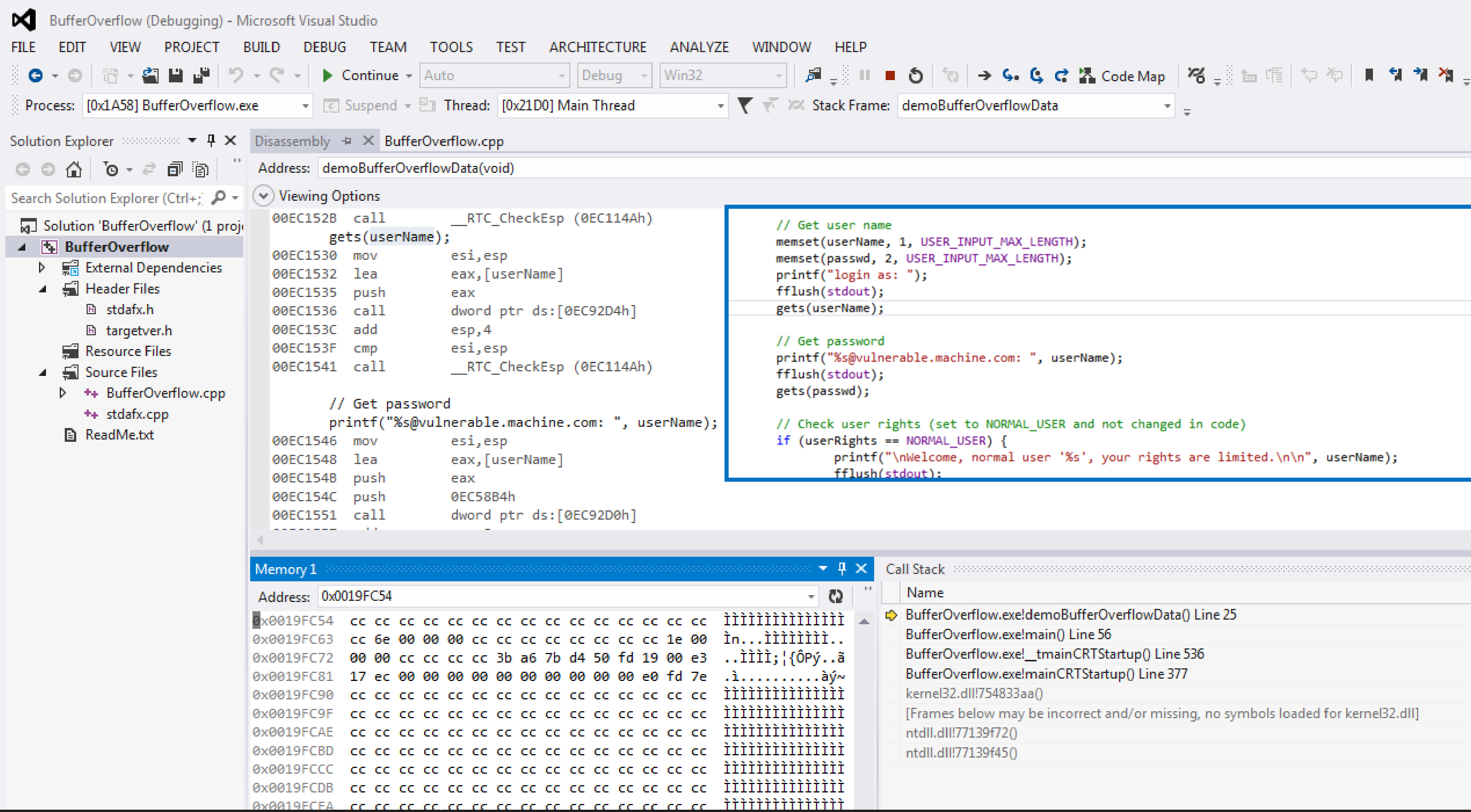


```
#include "stdio.h"
#include "memory.h"
// Note: GCC and MSVC uses different memory alignment
// Try "12345678DevilEvecosia" as a password for gcc build
// Try "1234567812345678Devil I am. Ha Ha" as a password for MSVC debug build

void demo
#define
#define
#define
```

Configuration: Active(Debug)   Platform: Active(Win32)

Enable String Pooling	
Enable Minimal Rebuild	Yes (/Gm)
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Runtime Library	Multi-threaded Debug DLL (/MDd)
Struct Member Alignment	Default
Security Check	Yes (/GS)
Enable Function-Level Linking	
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	

Deeper look into disassembly



Process: [0x1A58] BufferOverflow.exe

Thread: [0x21D0] Main Thread

Stack Frame: demoBufferOverflowData

Address: demoBufferOverflowData(void)

```
00EC152B  call        __RTC_CheckEsp (0EC114Ah)
       gets(userName);
00EC1530  mov         esi,esp
00EC1532  lea         eax,[userName]
00EC1535  push        eax
00EC1536  call        dword ptr ds:[0EC92D4h]
00EC153C  add         esp,4
00EC153F  cmp         esi,esp
00EC1541  call        __RTC_CheckEsp (0EC114Ah)

       // Get password
       printf("%s@vulnerable.machine.com: ", userName);
00EC1546  mov         esi,esp
00EC1548  lea         eax,[userName]
00EC154B  push        eax
00EC154C  push        0EC58B4h
00EC1551  call        dword ptr ds:[0EC92D0h]
```

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

        // Get password
        printf("%s@vulnerable.machine.com: ", userName);
        fflush(stdout);
        gets(passwd);

        // Check user rights (set to NORMAL_USER and not changed in code)
        if (userRights == NORMAL_USER) {
                printf("\nWelcome, normal user '%s', your rights are limited.\n\n", userName);
                fflush(stdout);
```

Address: 0x0019FC54

```
0x0019FC54  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FC63  cc 6e 00 00 00 cc cc cc cc cc cc cc cc 1e 00  Ìn...ÌÌÌÌÌÌÌÌ..
0x0019FC72  00 00 cc cc cc cc 3b a6 7b d4 50 fd 19 00 e3  ..ÌÌÌÌ;¦{ÔPý..ã
0x0019FC81  17 ec 00 00 00 00 00 00 00 00 00 00 e0 fd 7e  .ì..........àý~
0x0019FC90  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FC9F  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FCAE  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FCBD  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FCCC  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FCDB  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0019FCEA  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
```

```
BufferOverflow.exe!demoBufferOverflowData() Line 25
BufferOverflow.exe!main() Line 56
BufferOverflow.exe!__tmainCRTStartup() Line 536
BufferOverflow.exe!mainCRTStartup() Line 377
kernel32.dll!754833aa()
[Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]
ntdll.dll!77139f72()
ntdll.dll!77139f45()
```