PA193 - Secure coding principles and practices

LABS: Language level vulnerabilities:
Buffer overflow, type overflow, strings

Petr Švenda svenda@fi.muni.cz

CROCS
Centre for Research on Cryptography and Security

```
// 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 demoBufferOverflowData() {
	int		unused_variable = 30;
#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("\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);

	// 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);
	}
	if (userRights == ADMIN_USER) {
		printf("\nWelcome, all mighty admin user '%s'!\n", userName);
		fflush(stdout);
	}

	// How to FIX:
	//memset(userName, 0, USER_INPUT_MAX_LENGTH);
	//fgets(userName, USER_INPUT_MAX_LENGTH - 1, stdin);
	//memset(passwd, 0, USER_INPUT_MAX_LENGTH);
	//fgets(passwd, USER_INPUT_MAX_LENGTH - 1, stdin);
}
```

Setup

- Create new Visual Studio 2013 Project
  - File->New->Project->VisualC++->Win32 Console app
  - Turn off ‘Precompiled header’ and ‘SDL checks’
- Copy content of BufferOverflow.cpp from IS instead of generated main file
- Try to compile (disable warning on gets() function)
  - #define _CRT_SECURE_NO_WARNINGS
- Insert breakpoint (begin of demoBufferOverflowData()) – F9
- Run program in debug mode – F5
- Execute next step of program – F10
- Display memory (must be in debugging session), Debug → Windows → Memory

```
void demoBufferOverflowData() {
      int           unused_variable = 30;
#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("\n");

      // Get user name
      printf("login as: ");
      gets(userName);

      // Get password
      printf("%s@vulnerable.machine.com: ", userName);
      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);
      }
      if (userRights == ADMIN_USER) {
            printf("\nWelcome, all mighty admin user '%s'!\n", userName);
      }
}
```

Variable containing current access rights

Array with fixed length (will be overwritten)

Help output of address of local variables stored on the stack

Reading username and password (no length checking)

Print information about current user rights

Data in memory

```
void demoBufferOverflowData() {
    int     unused_variable = 30;
#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", &unu
    printf("%-20s: %p\n", "userRights", &userRigh
    printf("\n");

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

100 %

Memory 1

Address: 0x0024FB1B

Columns: Auto

```
0x0024FB1B  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FB2C  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FB3D  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FB4E  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FB5F  cc 02 02 02 02 02 02 02 02 cc cc cc cc cc cc cc
0x0024FB70  01 01 01 01 01 01 01 01 cc cc cc cc cc cc cc 6e
0x0024FB81  00 00 00 cc cc cc cc cc cc cc cc 1e 00 00 00 cc
0x0024FB92  cc cc 85 20 45 b8 6c fc 24 00 9a 20 dd 00 00 00
0x0024FBA3  00 00 00 00 00 00 e0 fd 7f cc cc cc cc cc cc cc
0x0024FBB4  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FBC5  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FBD6  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FBE7  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FBF8  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0024FC09  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

Autos | Locals | Memory 1 | Threads | Modules | Watch 1

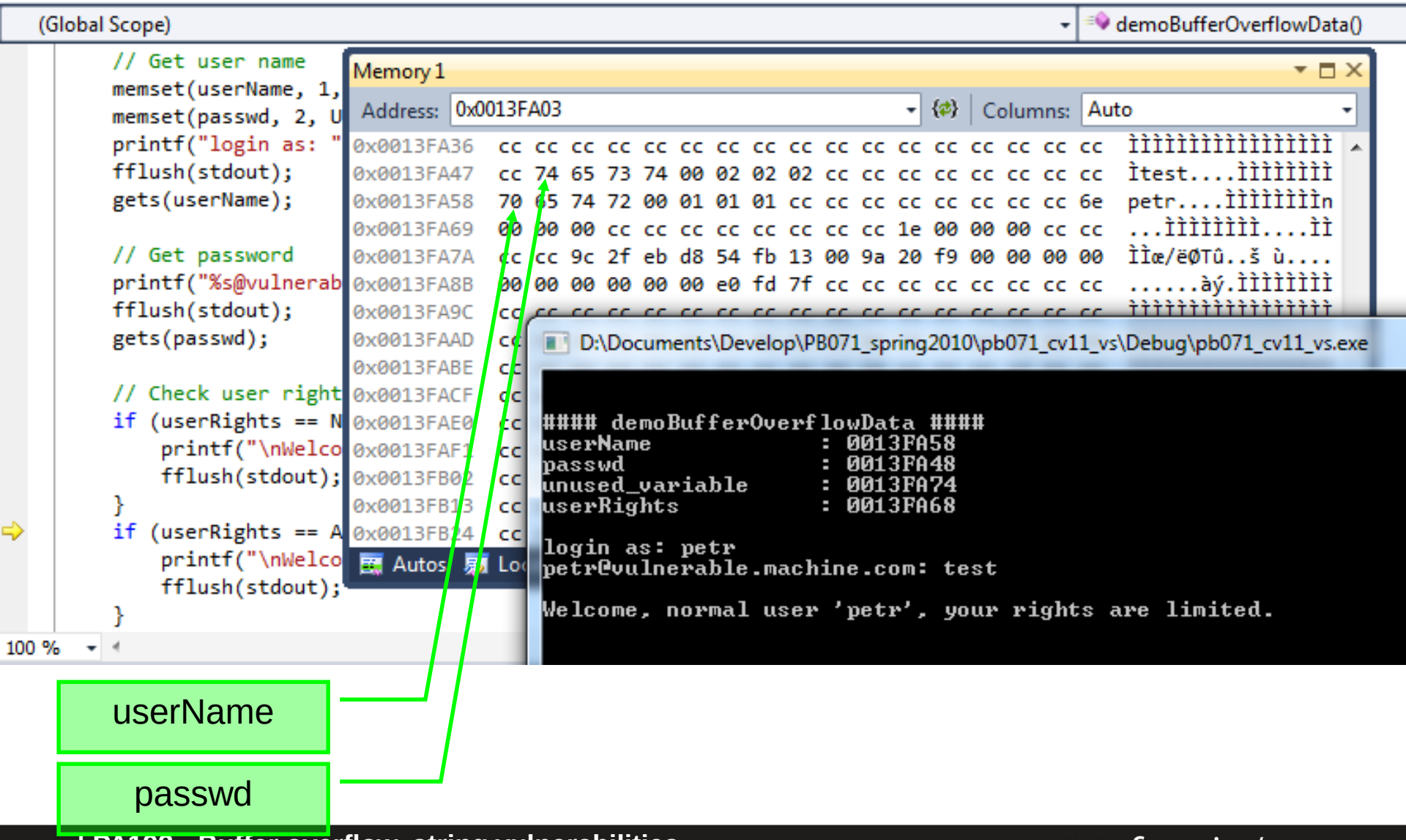
userName

passwd

userRights

unused_variable

Running without malicious input



Running with malicious input – userName

insert ‘evil’ into userName

```
// 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: ", userNam
fflush(stdout);
gets(passwd);

// Check user rights (set to NORMAL_USER and
if (userRights == NORMAL_USER) {
    printf("\nWelcome, normal user '%s', your
    fflush(stdout);
}
if (userRights == ADMIN_USER) {
    printf("\nWelcome, all mighty admin user
```

100 %

Memory 1

Address: 0x0024FB1B Columns: Auto

```
0x0024FB1B  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB2C  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB3D  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB4E  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB5F  cc 02 02 02 02 02 02 02 cc cc cc cc cc cc cc cc  Ì.......ÌÌÌÌÌÌÌÌ
0x0024FB70  65 76 69 6c 00 01 01 01 cc cc cc cc cc cc cc 6e  evil....ÌÌÌÌÌÌÌn
0x0024FB81  00 00 00 cc cc cc cc cc cc cc cc 1e 00 00 00 cc  ...ÌÌÌÌÌÌÌ....ÌÌ
0x0024FB92  cc cc 85 20 45 b8 6c fc 24 00 9a 20 dd 00 00 00  ÌÌ. E¸lü$.š Ý...
0x0024FBA3  00 00 00 00 00 00 e0 fd 7f cc cc cc cc cc cc cc  ......àý.ÌÌÌÌÌÌÌ
0x0024FBB4  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBC5  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBD6  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBE7  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBF8  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FC09  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
```

Autos | Locals | Memory 1 | Threads | Modules | Watch 1

Running with malicious input - passwd

Insert
‘1234567812345678Devil I am. Ha Ha’
into passwd

```
printf("login as: ");
fflush(stdout);
gets(userName);

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

// Check user rights (set to NORMAL_U
if (userRights == NORMAL_USER) {
    printf("\nWelcome, normal user '%
    fflush(stdout);
}
if (userRights == ADMIN_USER) {
    printf("\nWelcome, all mighty adm
    fflush(stdout);
}

// How to FIX:
```

Memory 1

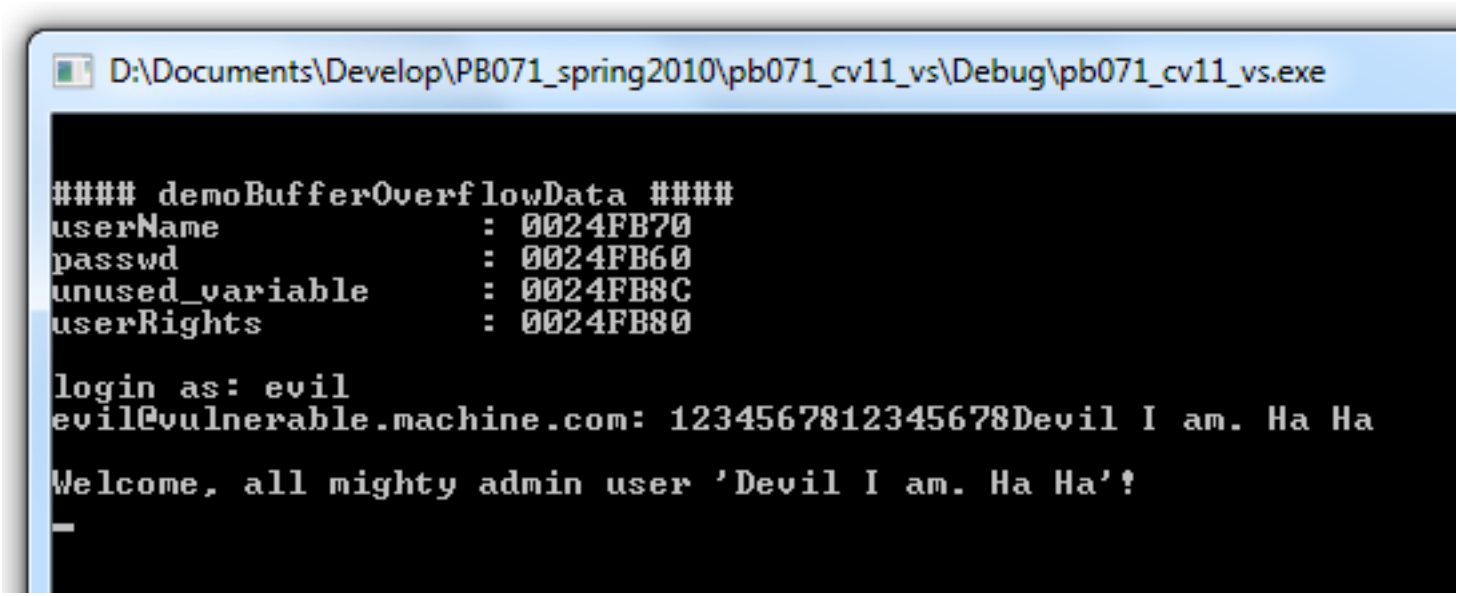
Address: 0x0024FB1B Columns: Auto

0x0024FB1B	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB2C	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB3D	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB4E	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FB5F	cc	31	32	33	34	35	36	37	38	31	32	33	34	35	36	37	Ì1234567812345678
0x0024FB70	44	65	76	69	6c	20	49	20	61	6d	2e	20	48	61	20	48	Devil I am. Ha Ha
0x0024FB81	00	00	00	cc	cc	cc	cc	cc	cc	cc	cc	1e	00	00	00	cc	...ÌÌÌÌÌÌÌÌ....ÌÌ
0x0024FB92	cc	cc	85	20	45	b8	6c	fc	24	00	9a	20	dd	00	00	00	ÌÌ. E¸lü$.š Ý....
0x0024FBA3	00	00	00	00	00	00	e0	fd	7f	cc	cc	cc	cc	cc	cc	cc	......àý.ÌÌÌÌÌÌÌÌ
0x0024FBB4	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBC5	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBD6	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBE7	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FBF8	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
0x0024FC09	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ

Autos | Locals | Memory 1 | Threads | Modules | Watch 1

- Too long password overflow **`userName`** and **`userRights`**

Running with attacker input - result



Questions (debug mode)

- How are userName, password and userRights positioned in memory?
- How you will find memory location (address) of *userRights* variable?
- How many bytes you need to write into *userName* variable to change *userRights* ?
- Can you get admin rights by changing userName only?

Questions (debug mode)

- Why is program throwing debugger exception when finishing function demoBufferOverflowData()?
- How program was able to detect memory corruption?
- Why 0xcc bytes are here? How you can type 0xcc into terminal?
- Can you get admin rights without raising runtime exception (*memory around userName variable corrupted*) when leaving demoBufferOverflowData()?
- Where you can find return address?
- What should be the return address value?
  - Try R-Click -> Go to Disassembly

Questions (release mode)

- Release mode, /GS on
  - What is memory layout with respect to debug mode?
  - Can you still execute buffer overflow and change userRights?
  - What is the value of canary word?
- Release mode, /GS off
  - What is the influence of /GS disabled?
  - What is the impact on addresses of variables?
  - Can you be admin in Release? Why?

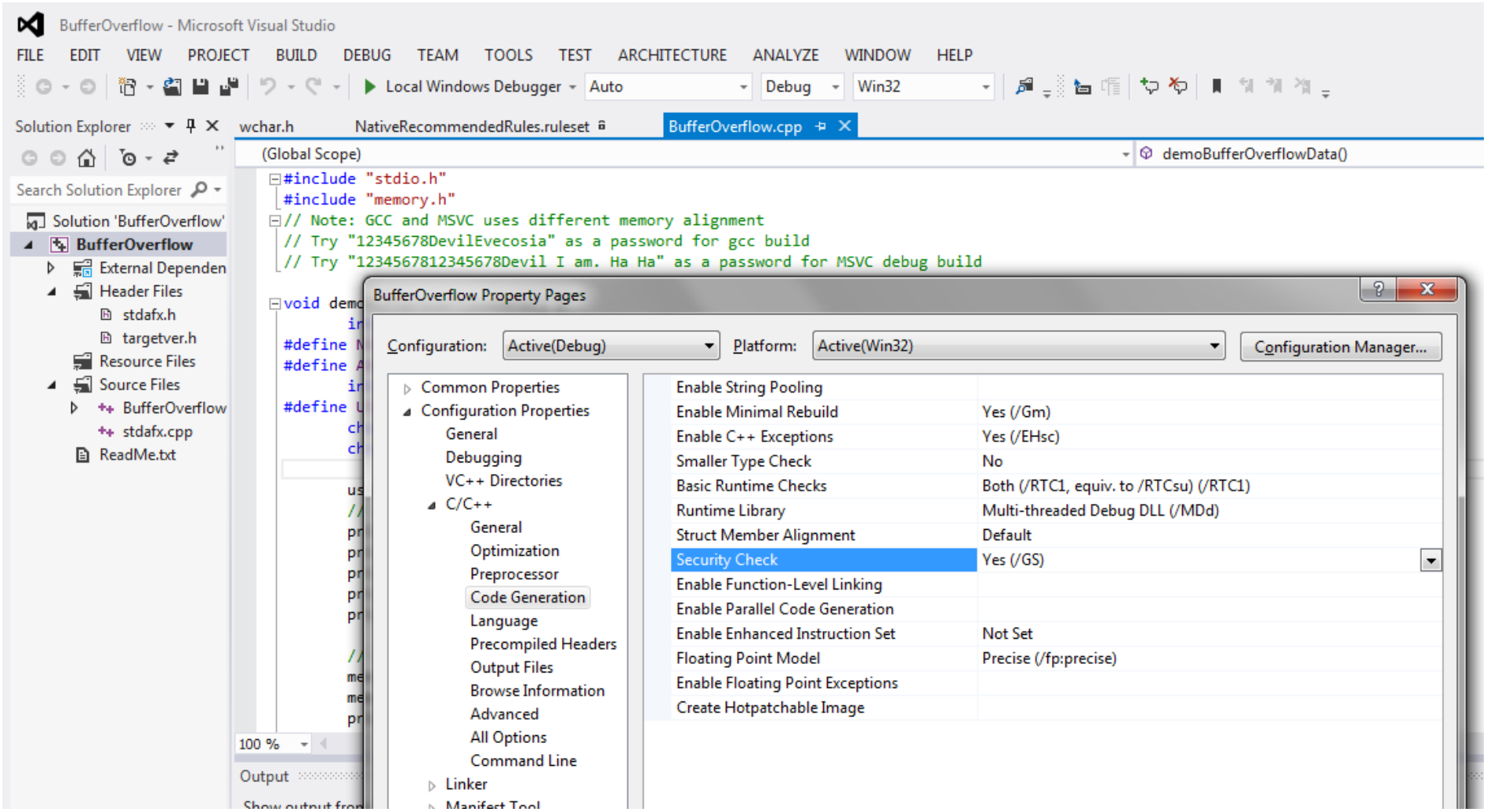
Lab – compiler protections

- GCC (e.g., QT Creator) & MSVC (Visual Studio)
  - list of compiler flags, release mode
- Compile program with/without compiler protection
  - bufferoverflowdemo.cpp::demoBufferOverflowData()
    - download from IS materials
  - return pointer smash behavior (crash, exception)
- Disassembly display of resulting binary
  - instruction-wise mode in IDE (Visual Studio), OllyDbg
  - existence of canary word (function with/without GS buffer)
- Display address of variable, function...,
    - run program multiple times – memory randomization (ASLR)

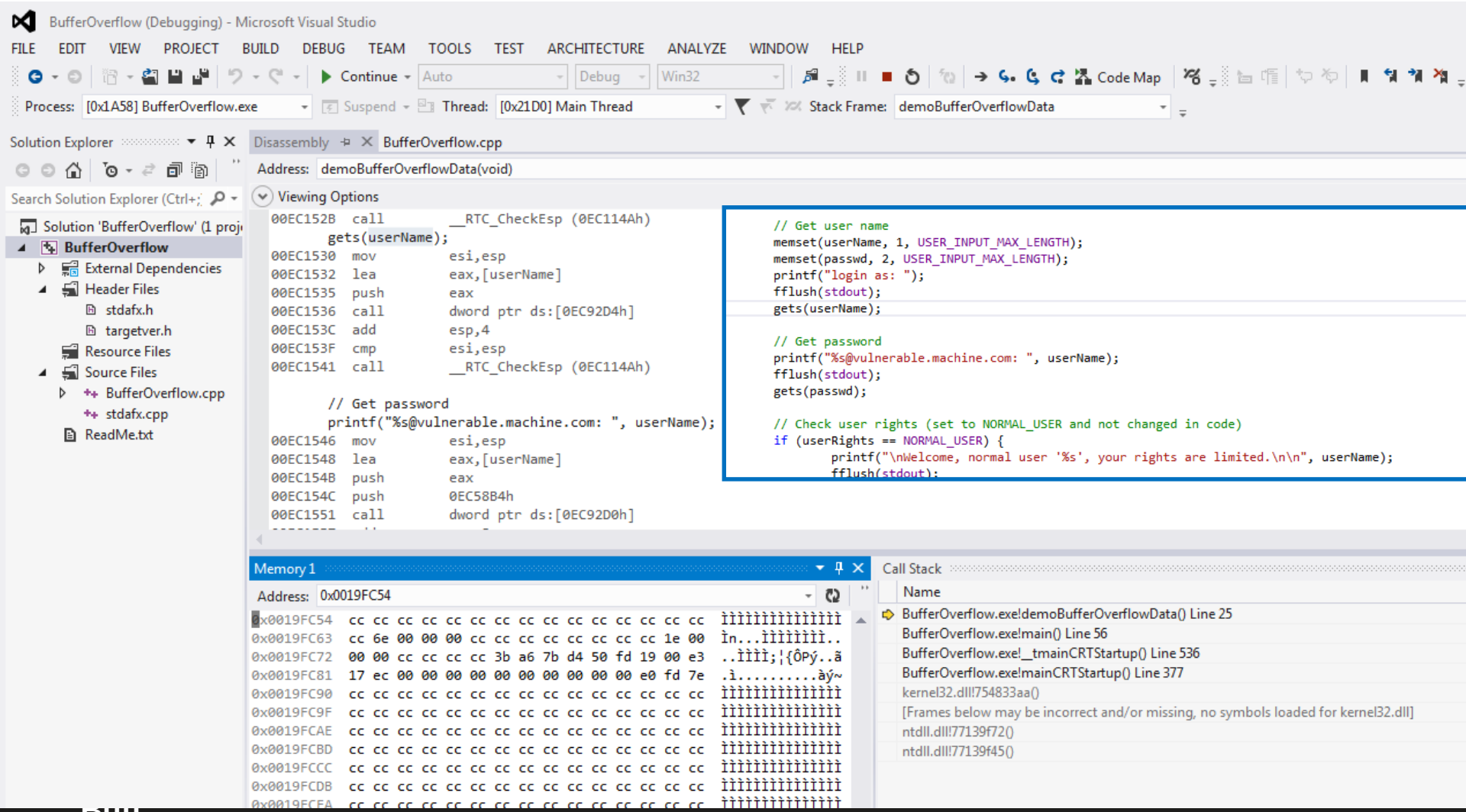
Compiler flags

- Locate all flags discussed during lecture
- Visual Studio Projects Settings
- Observe memory layout for stack frame with and without the flag
  - what is changing?
  - what is missing?

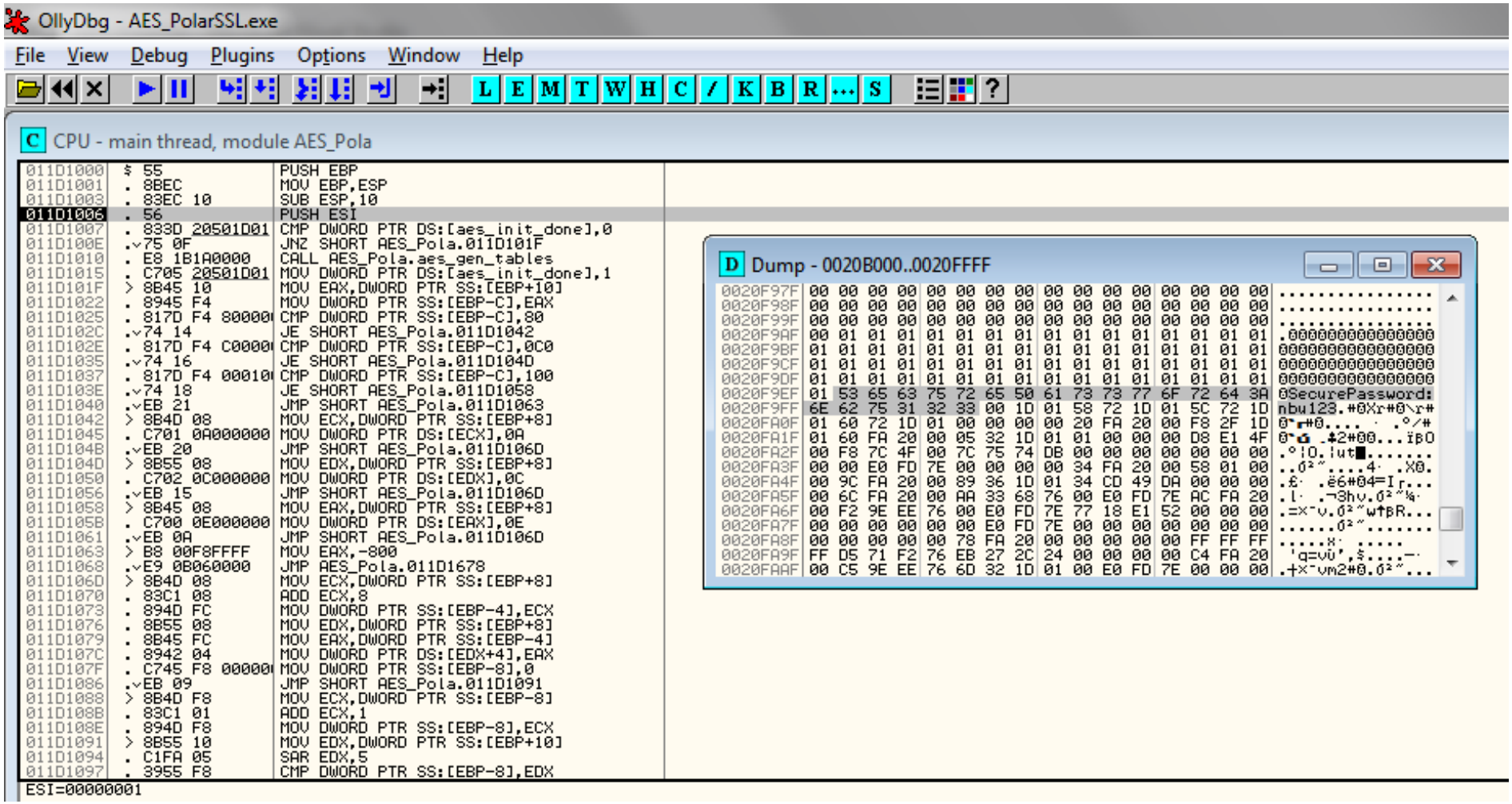
Compiler settings for /DEP and /ASLR



Deeper look into disassembly



Deeper look into disassembly (cont.)



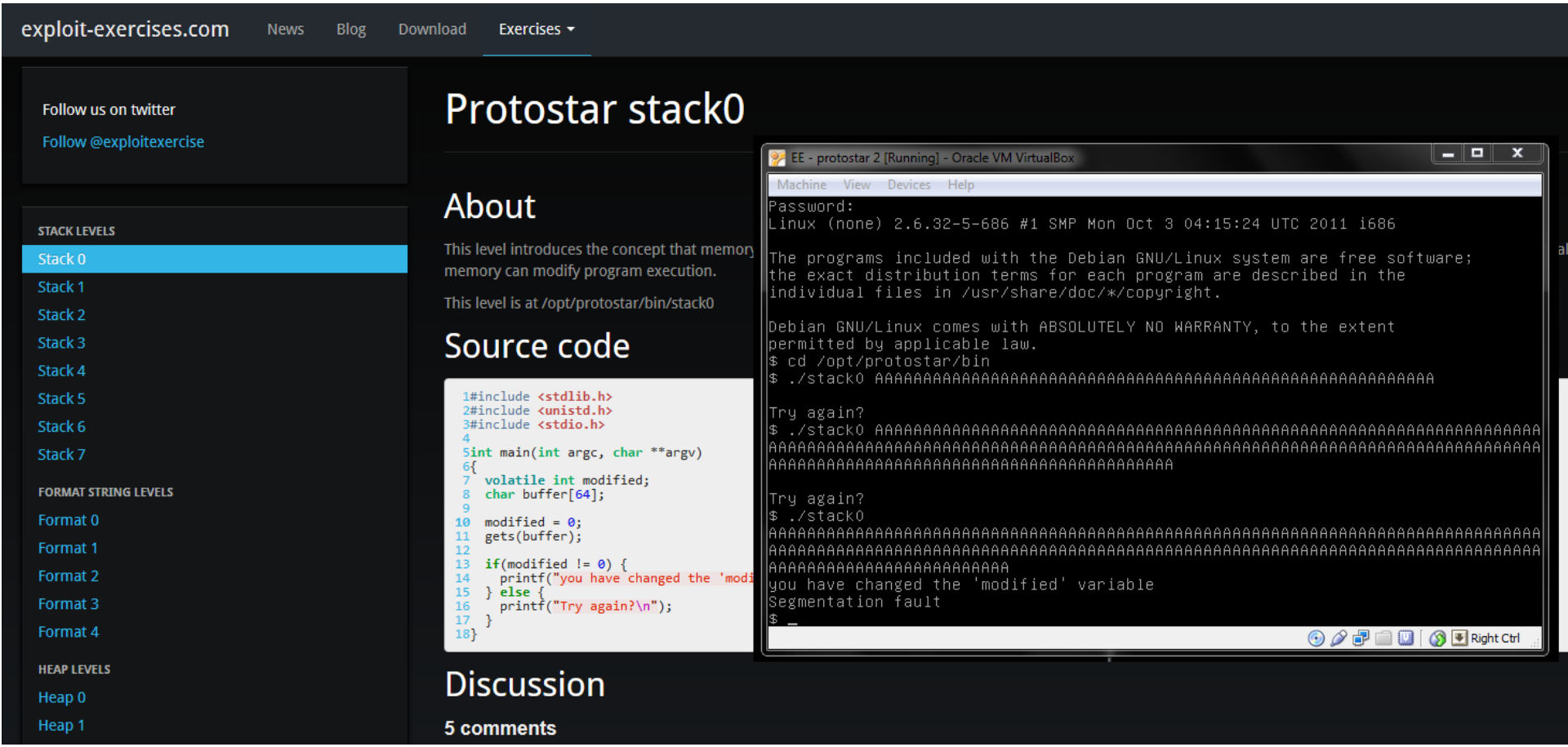
BinScope Binary Analyzer

- Download Microsoft SDL's Binscope
  - https://www.microsoft.com/en-us/download/details.aspx?id=11910
- Run BinScope Binary Analyzer (cmd or GUI)
  - **`binscope.exe`**
  - **`binscope.exe /o results.xml targetApp.exe`**
- Run on the binaries produced with different compiler settings
  - /GS...

Lab – exploiting exercises

- Protostar image (http://exploit-exercises.com)
  - pre-prepared virtual machine
  - http://exploit-exercises.com/protostar (task description)
- Important: site now not available, use this link:
  - https://web.archive.org/web/20140922114755/http://exploit-exercises.com/protostar
  - Or protostar.zip in IS
- Login credentials: user / user; root / godmode
- Challenges stored in /opt/protostar/bin/ directory
  - stack0-7
- Run it, supply malformed input leading to crash
- Think about how to fix the source code

Protostar virtual image with exercises



Lab - Homework

- Finish exploit exercises (Protostar, stack0-4)
  - submit txt file with inputs causing corruption
- Fix problems from these exercises (stack0-4)
  - submit corrected code that will not contain vulnerable constructions (save functions, proper arguments checking...)
- Bonus (+3 points):
  - Finish exploit exercise (Protostar, stack5-7)
- Upload your solution to IS repository
  - (homework vaults)
- Deadline: one week (22.10.2015 23:59 / 27.10.2015 23:59 depending on your seminar group)