# Sample exam

Lecture 19

# Format

- 1.5 hours
- Pencil and paper (no calculator)
- 10 questions
  - Quite a bit of work
  - Not much time to “discover” solution on the fly

# Questions

- Basic definitions
- Ethics
- Security principles
  - Definition, and
  - Example
- Vulnerability analysis
  - Strategies to find vulnerabilities
  - Prioritization

# Questions – cont'd

- Assigned reading material
  - Only 3 this year…
- Short vulnerable program/snippet of code
  - Find vulnerability
  - Propose fix
  - Analyze fix
  - Describe exploit
- Shellcode analysis

# Questions – cont'd

- Mitigation techniques
- Longer program (~1 page)
  - Find vulnerabilities, and
  - Propose fix
  - Very simple command line C or servlet Java programs

# Key questions

- Shellcode analysis
- Longer code analysis
- They are worth slightly less than a third of the final grade
  - Make sure you understand what these questions ask
  - Practice!
- Strategies

# LET’S PRACTICE

Sample A22652 No calculator permitted in this examination

**THE UNIVERSITY OF BIRMINGHAM**

**THIS PAGE TO BE REPLACED BY OFFICE**

06 20010

**Secure Programming**

2013 1.5 hours

[Answer ALL questions]

No calculator

1. Give a definition of "risk".

2. (a) Explain and discuss the "Least privilege" security principle.
   (b) Discuss a system that satisfies this principle and one that violates this principles.

3. Suppose you want to identify buffer overflows in a C application. You have a tool that analyzes the source code of the application and automatically locates all the uses of insecure string handling functions, such as `strcpy` and `strcat`. Describe the steps you would take to validate whether a use of one of these functions, identified by the tool, corresponds to an actual vulnerability or not.

4. You are analyzing the code of an application and you find the following function (in Java):

```
1  public Hashtable getUserInfo(final String name, final Connection c)
2  throws Exception {
3      Hashtable res = new Hashtable();
4
5      String q = "SELECT * FROM users WHERE name = '" + name + "'";
6
7      Statement st = c.createStatement();
8      ResultSet rs = st.executeQuery(q);
9      while (rs.next()) {
10         res.put("id", rs.getInt(1));
11         res.put("name", rs.getString(2));
12         res.put("password", rs.getString(3));
13     }
14
15     return res;
16 }
```

Describe under which conditions the function contains a vulnerability.

5. (a) Consider the following code:

```
1 void foo(char *s)
2 {
3     char buf[128];
4
5     strcpy(buf, s);
6
7     printf("%s\n", buf);
8 }
```

Assume that the string s is under user control.

Draw the layout of the stack at the end of the function when s is "test". Then, draw the layout of the stack at the end of the function during a successful buffer overflow attack.

(b) Now assume that the code gets changed as follows:

```
1 void foo(char *s, int len)
2 {
3     char buf[128];    strncpy(buf, s, len);
4     printf("%s\n", buf);
5 }
```

An example invocation is:

```
1 int main(int argc, char **argv) {
2     foo(argv[1], strlen(argv[1]);
3 }
```

Is the vulnerability fixed? If not propose a correct fix.

6. (a) What is the reason for avoiding NULL bytes in shellcode?

(b) Suppose a program defends against buffer overflows by rejecting all inputs that contain the sequence of bytes 0xcd, 0x80 (int $80). Is this an effective defense? If not, what technique could you use to write shellcode that bypasses it?

7. (a) Explain the address space layout randomization (ASLR) defense mechanism.
   (b) Describe one technique to bypass it.

8. Using the instruction decoding, the ASCII, and the syscall ID tables below, decode the following shellcode (that is, produce the corresponding x86 instructions), and explain what it does.

```
eb 36 b8 05 00 00 00 5b 31 c9 cd 80 89 c3 b8 03
00 00 00 89 e7 89 f9 ba 00 10 00 00 cd 80 89 c2
b8 04 00 00 00 bb 01 00 00 00 cd 80 b8 01 00 00
00 bb 00 00 00 00 cd 80 e8 c5 ff ff ff 2f 65 74
63 2f 70 61 73 73 77 64 00
```

| Bytes | Instruction | Description |
|---|---|---|
| 31 c9 | xor %ecx,%ecx | |
| 5b | pop %ebx | |
| 89 c2 | mov %eax,%edx | |
| 89 c3 | mov %eax,%ebx | |
| 89 e7 | mov %esp,%edi | |
| 89 f9 | mov %edi,%ecx | |
| b8 *AB CD EF GH* | mov 0xGHEFCDAB,%eax | |
| ba *AB CD EF GH* | mov 0xGHEFCDAB,%edx | |
| bb *AB CD EF GH* | mov 0xGHEFCDAB,%ebx | |
| cd 80 | int $0x80 | |
| e8 *AB CD EF GH* | call 0xGHEFCDAB | |
| eb *AB* | jmp 0xAB | |

| Hex code | Character | Hex code | Character |
|---|---|---|---|
| 2d | - | 6d | m |
| 2f | / | 6e | n |
| 61 | a | 6f | o |
| 62 | b | 70 | p |
| 63 | c | 71 | q |
| 64 | d | 72 | r |
| 65 | e | 73 | s |
| 66 | f | 74 | t |
| 67 | g | 75 | u |
| 68 | h | 76 | v |
| 69 | i | 77 | w |
| 6a | j | 78 | x |
| 6b | k | 79 | y |
| 6c | l | 7a | z |

In particular, the following sequence of bytes corresponds to the NULL-terminated string `/etc/passwd`:

```
2f 65 74 63 2f 70 61 73 73 77 64 00
```

| ID | Syscall name |
|---|---|
| 0x1 | exit |
| 0x3 | read |
| 0x4 | write |
| 0x5 | open |

9. (a) Describe the vulnerability in this code snippet:

```
1 if (!access(fname, W_OK)) {
2         truncate(fname, 0);
3 } else {
4     printf("Cannot truncate %s\n", fname);
5 }
```

   (b) Fix the vulnerabilty.

10. Identify the vulnerabilities contained in the following code (use the line numbers to specify the location of each vulnerability) *and* briefly propose a fix for each of them.

```
1   public class BuggyFriend extends HttpServlet {
2     Connection conn = ... /* initializes the DB connection */;
3     String sanitizeSql(String str) {
4         String cleanStr = str.replace("SELECT", "_SELECT_");
5         cleanStr = cleanStr.replace("OR", "_OR_");
6         return cleanStr;
7     }
8     protected void processRequest(HttpServletRequest request,
9       HttpServletResponse response)
10    throws ServletException, IOException, SQLException {
11        PrintWriter out = response.getWriter();
12
13        String action = request.getParameter("action");
14        // search for a friend whose name contains the given string
15        if ("search".equals(action)) {
16            Statement st = conn.createStatement();
17            ResultSet rs = st.executeQuery("SELECT * "
18              + "FROM friends WHERE name LIKE '%"
19              + sanitizeSql(request.getParameter("q"))
20              + "%'");
21            while (rs.next()) {
22                String f = rs.getString("NAME");
23                out.write(f);
24            }
25            st.close();
26        // view the privacy policy in the language specified by the
27        // user (lang parameter)
28        } else if ("view_privacy_policy".equals(action)) {
29            String lang = request.getParameter("lang");
30            if (lang.matches("[a-z]{2}")) {
31              BufferedReader f = new BufferedReader(
32                new FileReader("policies/privacy/" + lang));
33              String s;
34              while ((s = f.readLine()) != null) {
35                  out.write(s + "\n");
36              }
37            }
38        // send a friendship invitation via email to the address
39        // specified by the user (to parameter)
40        } else if ("email".equals(action)) {
41            String to = request.getParameter("to");
42            String[] cmd = {"/bin/bash", "-c",
43                "cat invitation.txt | "
44                + "mail -s \"Friendship invitation\" " + to};
45            Runtime.getRuntime().exec(cmd);
46        // invalid action
47        } else {
48            out.write("Error: invalid action: " + action);
49        }
50    }
51  }
```