Software Security

Midterm Exam Preparation

Note: This document contains potential questions for the midterm exam, covering the topics OWASP, Reverse Engineering and Software Analysis.

Be aware, there is no guarantee for the correctness of the answers!

OWASP Advanced Topics

1) Name 5 of the OWASP Top 10

Open Web Application Security Project (OWASP). The OWASP Top 10 represents a broad consensus about what the most critical web application security flaws are (awareness document).

- A1 - Injection
- A2 - Broken Authentication and Session Management
- A3 - Cross Side Scripting (XSS)
- A4 - Insecure Direct Object References
- A5 - Security Misconfiguration
- A6 - Sensitive Data Exposure
- A7 - Missing Function Level Access Control
- A8 - Cross-Site Request Forgery (CSRF)
- A9 - Using components with known vulnerabilities
- A10 - Unvalidated Redirects and Forwards

2) Which other forms of "inputs" (other than user entered input) do you know?

3) Some Code with prepared statement was given, what is the problem with that? (string was directly used in sql string)

When concatenating the SQL string with the user input, the user input can interfere with the program code and even alter it. Use binding of the variables to avoid injection of malicious code:

```
PreparedStatement pstmt =
conn.prepareStatement ("insert into EMP (ENAME) values (?)"); String name =
request.getParameter("name");
pstmt.setString (1, name);
pstmt.execute();
pstmt.close();
```

The prepared statement and the data will be transmitted to the DBMS separately -> the data will never be interpreted as program code but simply as data.

4) What other counter measures against SQL injection than prepared statements can you use while developing?

- Stored procedures
- Escaping of SQL special characters (e.g. mysql_real_escape)
- Validating user input

5) What counter measures against SQL injection can you do on the database side?

- Give least privileges to web application's DB user
- Logging / auditing

6) Software Assurance Maturity Model (SAMM)

- What is the highest level in XY maturity level?
- What have to do with the organizational things in the maturity level?

7) What is Threat Modeling?

Threat modeling allows you to systematically identify and rate the threats that are most likely to affect your system.
-> Identify worst-case scenarios for the software under development and assess the conditions and likelihood that they occur.
Different methodologies:

- Attack trees (by Schneier)
- Microsoft SDL (Secure Development Lifecycle)
    - STRIDE / DREAD

8) What are the common terms in Threat Modeling?

- **Asset**: a resource of value, such as the data in a database or on the file system. A system resource.
- **Threat**: A potential occurrence, malicious or otherwise, that might damager or compromise your assets.
- **Vulnerability**: A weakness in some aspect or feature of a system that makes a threat possible. Vulnerabilities might exist at the network, host or application levels.
- **Attack (or exploit)**: An action taken by someone or something that harms an asset. This could be someone following through on a threat or exploiting a vulnerability
- **Countermeasure**: A safeguard that addresses a threat and mitigates risk

9) What are the steps in the Threat Modeling process with MS SDL?

1. Identify Assets
2. Create an Architecture Overview
3. Decompose the Application
4. Identify the Threats (STRIDE)
5. Document the Threats
6. Rate the Threats (DREAD)

10) What is the difference between Design and Architecture?

Design	Architecture
• Tactical (**How** are we building it?) • How should the code be organized? • Design of components, classes, responsibilities, patterns • Iterations can (and will) happen	• Strategic (**What** are we building?) • Choice of frameworks, languages, goals, development methodology • Highest level of abstraction • changes are hard once established (almost no iterations will happen)

11) What Secure Design Principles do you know?

- **Least Privilege**: only the the necessary level of access rights (privileges) has been given explicitly to it for a minimum amount of time in order for it to complete its operation (e.g. do not use root to run your DB)

- **Separation of Duties**: compartmentalizes software functionality into two or more conditions, all of which need to be satisfied before an operation can be completed (e.g. in SWIFT applications you need 2 admins to generate a new user)
- **Defense in Depth**: layering security controls and risk mitigation safeguards into software design (e.g. do not store passwords in plain text and use a salted brute-force heavy hash functions)
- **Fail Secure**: ensures that the software reliably functions when attacked and is rapidly recoverable into a normal business and secure state in the event of design or implementation failure (e.g. do not display detailed technical error messages to users, might create side channel that leaks information - e.g. crash dump or padding oracle attack)
- **Least Common Mechanisms**: Mechanisms common to more than one user or process are designed not to be shared (e.g. be very cautious when coding functions that can be used by admins and non-admins)
- **Economy of Mechanisms**: keep the design as simple as possible (KISS). E.g. rely on TLS and do not use some client-side crypto to hash a password
- **Complete Mediation**: All accesses to objects are checked to ensure that they are allowed. (e.g. do not solely rely on IDs for access control (which is common with RFID tokens))
- **Open Design**: The security of you software should not be dependent on the secrecy of the design (Kerckhoff's principle, i.e. do not use closed source cryptography)
- **Psychological Acceptability**: Avoid security controls that are hard/impossible to follow (e.g. do not force users to change the password every week)
- **Weakest Link**: Avoid Single-Point-of-Failure in your design (e.g. HTTPS termination if not necessary)
- **Leveraging Existing Components**: Reuse tested and proven libraries for critical functions (e.g. usage of OpenSSL instead of writing your own code)

12) How works CSRF + difference to XSS?

Reverse Engineering

1) What is Code Obfuscation? What reasons for it might exist?

A technique to obscure the control flow of software as well as data structures that contain sensitive information (e.g. cryptographic keys). It is used to mitigate the threat of reverse engineering.
Reasons to apply code obfuscation:

- Protecting secrets in software (System Architecture, Cryptographic keys, Algorithms)
- Tamper proofing (prevent people from tampering with your system e.g. cheats in games)
- Malware (prevent analysis / detection)

2) What makes disassembling so hard?

- Structure is lost (Data types, names and labels are lost)
- No one-to-one mapping
  - The same code can be compiled into different (equivalent) assembler blocks
  - An assembler block can be the result of different code snippets
- Overlapping instructions / functions
  - x86 instructions have variable length
  - Start address of instructions are not known in advance

    - Depending from which byte you start disassembling you can obtain different instructions
- Desynchronization
    - How to distinguish data from code?

3) What is the difference between linear sweep and recursive traversal?

Two ways of disassembling a binary executable into into its assembler instructions:

Linear Sweep	Recursive Traversal
• Start at the beginning of code (.text) section • Disassemble **one instruction after the other** • Examples: objdump, gdb, windbg	• Start at the program entry point • Disassemble one instruction after the other **until you find a control flow instruction**, e.g. jmp • **Recursively follow the targets** of e.g. jmp • Examples: IDA, OllyDbg, radare2 • Pros: better at interleaving data/code • Cons: coverage! - what to do with indirect jumps?

4) Build a Control Flow Graph (CFG) for the following program:

```
int gcd(int x, int y) {
   int temp;
   while (true) {
      if (x%y == 0) break;
      temp = x%y;
      x = y;
      y = temp;
   }
}
```

5) How can reverse engineering be made more difficult?

- Obfuscate the Control Flow Graph (CFG)
- Try to introduce additional branches, which are never used
- Change the structure to reflect a different layout

6) Explain opaque predicates, how are they used?

An opaque predicate is a condition which the result is known in advance by the programmer and that cannot be resolved statically (by a compiler, for example) and must be resolved dynamically.

Other definition: An opaque predicate is referred to as a branch that always executes in one direction, which is known to the creator of the program, and which is unknown a priori to the analyzer.

By introducing opaque expressions which always evaluate to the same value (known to the programmer in advance) static program analysis tools can be misled. (Conditional branches which are either never or always taken)

-> Obfuscation Method to make static program analysis harder.

7) What are Indirect Jumps and how can they be used in the context of making reverse engineering harder?

Indirect Jumps (aka JMP [EAX]) do not reveal their jump target until program execution because the jump address is generated during runtime. It can be used for software obfuscation in order to make it difficult for static analyzers to reconstruct the Control Flow Graph.

8) What is a Branching Function and what is it used for?

Replace CALL and JMP instructions with calls to generic function, which decides at runtime where to jump. For a static analyzer it is difficult to calculate the jump target without executing the software. -> Making static disassembly harder (confusing the disassembler)

9) How can the strength of an obfuscation be measured?

Collberg's Metric:

- describes how much more difficult the obfuscated program is to understand for humans (potency).
- strength against automatic de-obfuscators (resilience)
- cost: computational overhead (memory, time)

10) Difference, pros and cons of static analysis vs. dynamic analysis, examples.

During the Static Analysis the machine code is analyzed by the disassembler but is not executed. During Dynamic Analysis the machine code is executed on some inputs and the execution is monitored by an external tool (e.g. debugger).

Static Analysis	Dynamic Analysis
+ fast + process the entire file at once - theoretically: undecidable (Halting Problem) - cost: often does not scale to real world programs - needs to understand the effect of library or syscalls (how to model environment?) - problematic instructions (Indirect memory addressing, jumps, compiler optimizations) - anti-analysis tricks - fail to analyze programs compressed with a packer	- can be slow: proportional to the number of instructions executed during runtime - coverage: only disassembles a slice of the program (those instructions that were executed) - anti-debugging/-analysis tricks (binary check that it is running inside a VM, debugger) - Malware needs a sandbox

11) What is a Packer, and what is problematic about it?

A Packer is a compression tool to reduce the size of executables on disk.
When malware is compressed with a packer, it can evade anti-virus software and benefit from anti-debugging techniques included in packers.

Software Analysis

1) What is the difference between Software- and Hardware Breakpoint?

Software Breakpoints	Hardware Breakpoints
• Break the execution of the program at a specific address Implementation: • The debugger saves the first byte of the target address and replaces it with an INT 3 instruction (0xCC) • When the instruction is executed, the debugger traps the exception and replaces the trap with its original byte • When the user continues the execution, the debugger steps to the next instruction, restores the breakpoint, and then continues the program execution Characteristics: • Can be created in an unlimited number • They require the debugger to change the program code (impossible for self-modifying code -> breaks modified code) • The INTO 3 instruction can be detected	• Take advantage of the CPU to stop execution at a certain point Implementation: • The target address is set in one of the debugger registers (DR0-DR3) • The type of breakpoint is configured in DR7 (Break on execution, on R, on R/W; length of data item to be monitored -> 1, 2 or 4 bytes) • When the address on the bus matches those stored in the debug registers, a breakpoint signal, interrupt one (INT1) is sent and the CPU halts the process • DR6 is used as a status register to allow the debugger to know which debug register has triggered