

Security Testing

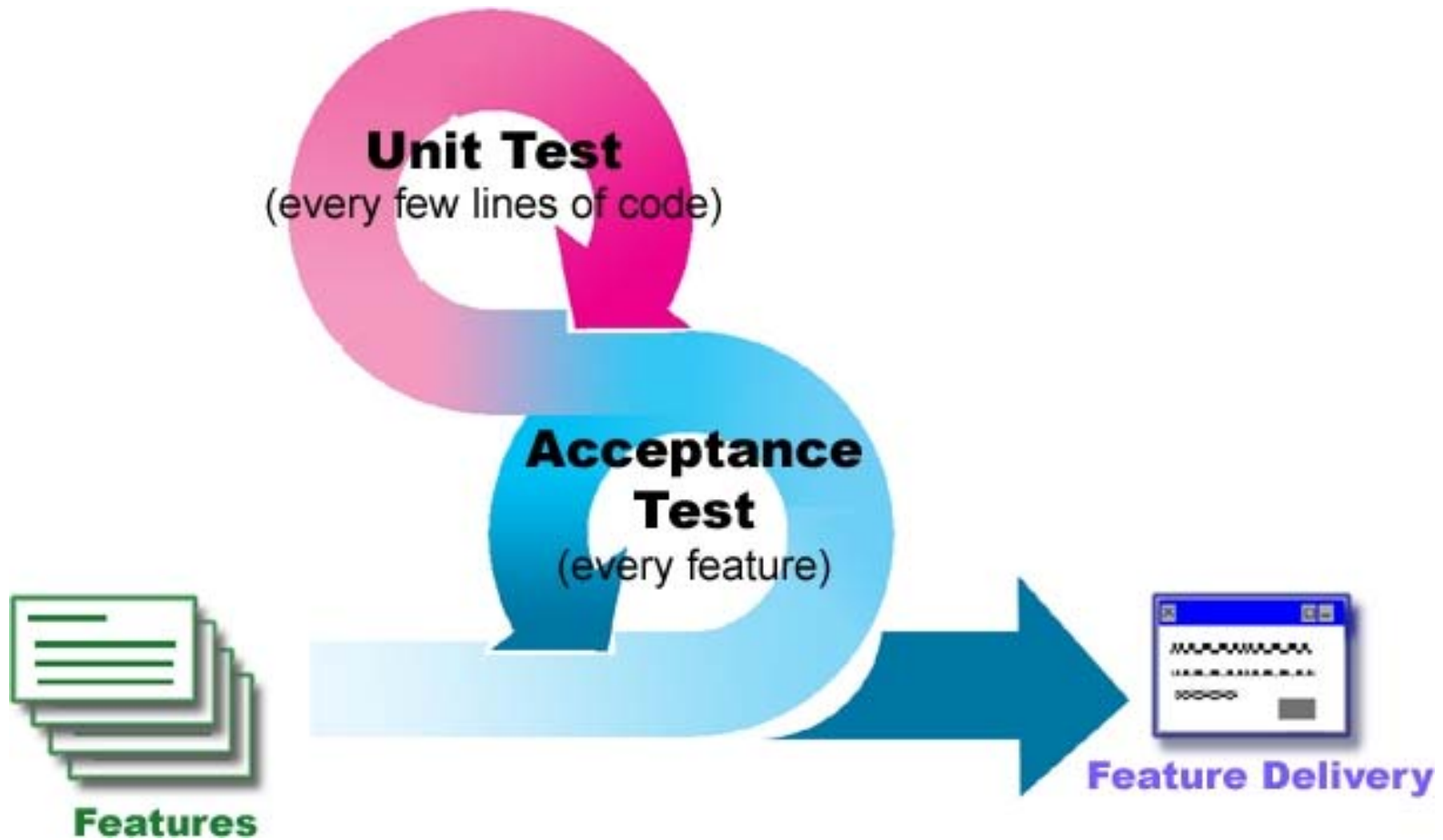
- **How security testing is different**
 - **Types of security attacks**
 - **Threat modelling**
-
- **Note: focus is on security of applications (not networks, operating systems)**
 - **Security testing is about making sure the defensive mechanisms work correctly (e.g. you cannot spoof another user's identify), rather than that the functionality works correctly**

Concepts

- **Threat:** a potential event that will have an unwelcome consequence if it becomes an attack.
- **Vulnerability:** a weakness in a system, such as a coding bug or a design flaw.
- **Attack:** occurs when an attacker has a motive and takes advantage of a vulnerability to threaten an asset.
- **Asset:** also referred to as threat target.

- **Analogy:**
 - A **fault** may remain latent, or may be surfaced as a **failure** when the code is executed
 - A **vulnerability** may remain latent, or may be exploited by an attacker, enabling an **attack**

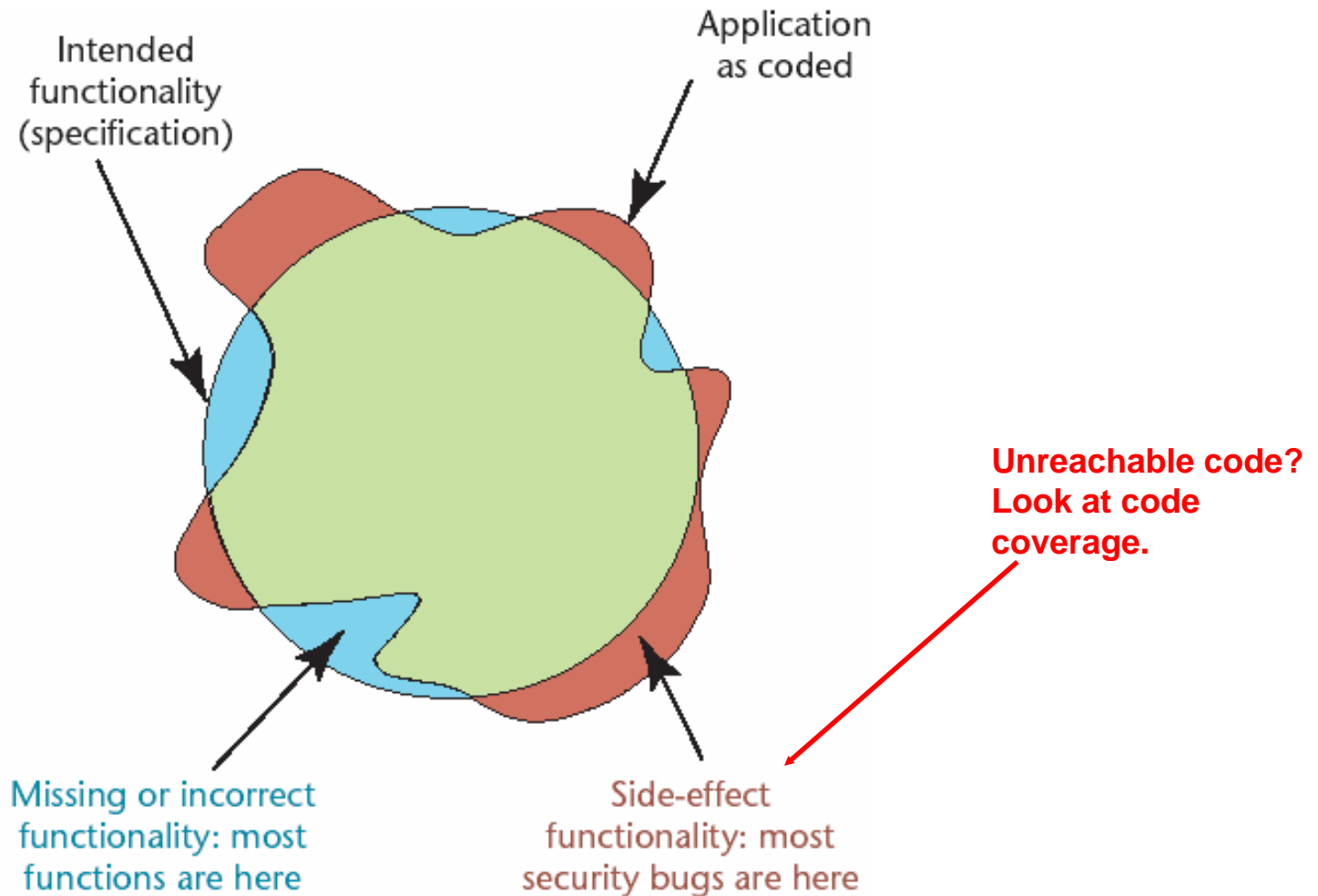
Test Driven Development



General Good Practices

1. The best defense is building security **IN**, not testing it in. [security is an emergent property; having a test-centric methodology helps]
2. Decide how much security is needed.
3. Perform automated testing.
4. Test at every stage.
5. Make a test plan.
6. Test all system components.
 - Consider business impact of failure, compromise or unavailability
 - What about third-party software? Should it be trusted?
7. Follow a specific software development methodology.
8. [Also, use a layered approach to building in and testing for security → Defense in Depth]
9. [Practice Role-based Access Control (RBAC). Minimize the capability given to each “role” → use least privilege]

Beware of Side Effects



Four general classes of security attacks

➤ **Dependencies**

- Application may inherit vulnerabilities from components its dependant upon
- Libraries that contain security service may fail (so application must respond securely – defense in depth!)

➤ **Unanticipated user input**

- Reserved words, escape characters, long strings, boundary values

➤ **Expose design vulnerabilities**

- Esp. Ports open, insecure default values, test instrumentation interwoven with implementation code [that can explicitly bypass security for ease of testing]

➤ **Expose implementation vulnerabilities**

- Developers only understand their piece, may expose data

Example Security Testing Techniques

➤ Spoofing Identity

- Attempt to force the application to use no authentication; is there an option to allow this, which a non-administrator can use?
- Can you view a valid user's credentials on the wire or in persistent storage?
- Can "security tokens" (e.g. a cookie) be replayed to bypass an authentication stage?

➤ Tampering with the data

- Is it possible to tamper with and rehash the data?
- Create invalid hashes and digital signatures to verify they are checked correctly.

➤ Repudiation

- Do conditions exist that prevent logging or auditing?
- Is it possible to create requests that create incorrect data in an event log?

Example Security Testing Techniques II

➤ Information Disclosure

- Attempt to access data that can be accessed only by more privileged users.
- Make the application fail in a way that discloses useful information to an attacker (for example, error messages)
- Kill the process and then perform disk scavenging, looking for sensitive data written to disk.

➤ Denial of Service (Dos)

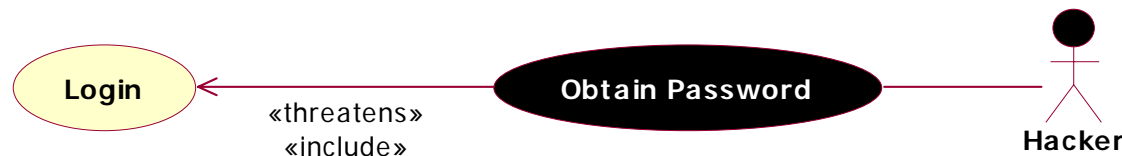
- Flood a process with so much data it stops responding to valid requests.
- Does malformed data crash the process?

➤ Elevation of Privilege

- Can you execute data as code
- Can an elevated process be forced to load a command shell, which in turn will execute with elevated privileges?

Consider your project

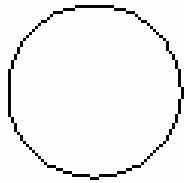
- Think about your types of users and your database project. What rights (read, write, update) do each type of user have for each table?
- Think about your project and possible “misuse cases” – write two security scenario tests.
 - A misuse case is a use case from the point of view of an actor hostile to the system; the actor is a hacker deliberately threatening the security of the system and/or the privacy of the users of the system [Alexander]. Done in requirements phase.
 - **Every misuse case must have one or more tests.**



Threat modeling

- Security-based analysis that helps people determine the highest level risks posed to the product and how threats manifest themselves
- Decompose the application (data flow diagram, DFD) to investigate the components, or assets or **threat targets**, of the application and how data flows between them.
 - Treat targets are: every data source, process, data flow, and interactor
- Using STRIDE, identify threats for each threat target. These serve as the roots for the threat trees; there is one tree per threat goal.
 - Threats should have at least one test case in the test plan [realistically, the higher-risk threats]
- Build one or more threat trees for each threat target, as appropriate (risk-based choice).
- Using DREAD (or some other threat ranking method, determine the security risk for each threat tree.

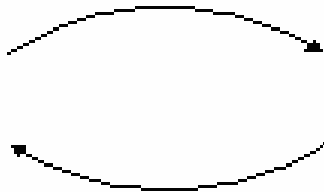
Data Flow Diagram (DFD): Symbols



A process. Transforms or manipulates data.



Interactor. Input into the system.



Data Flow. Depicts data flow from data stores, processes or interactors.



Data Store. A location that stores temporary or permanent data.



Boundary. A machine, physical address space or trust boundary.

Data Flow Diagram (DFD): Sample

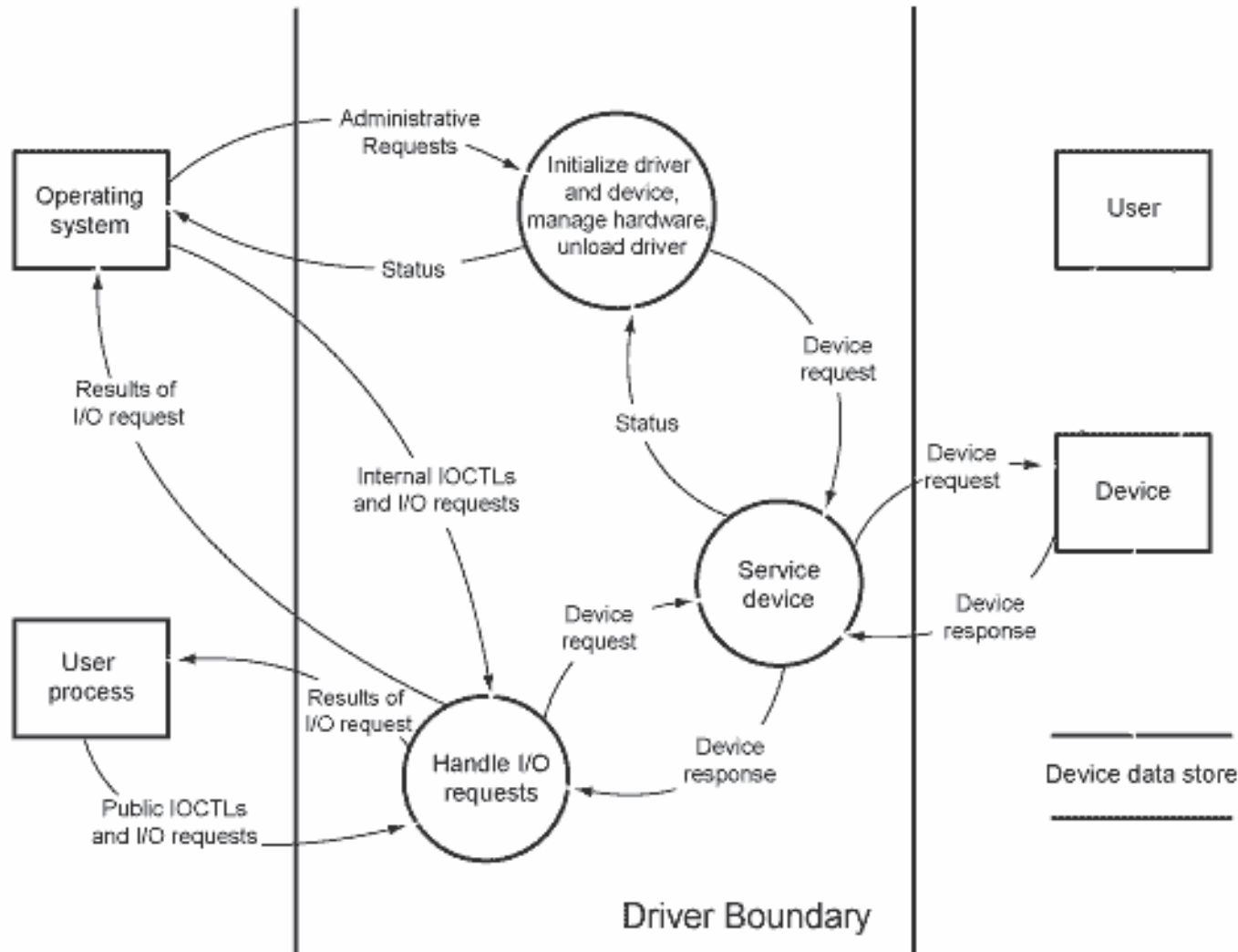


Figure 1. Sample data flow diagram for hypothetical kernel-mode driver. (modified)

Simple rules for DFD entities:

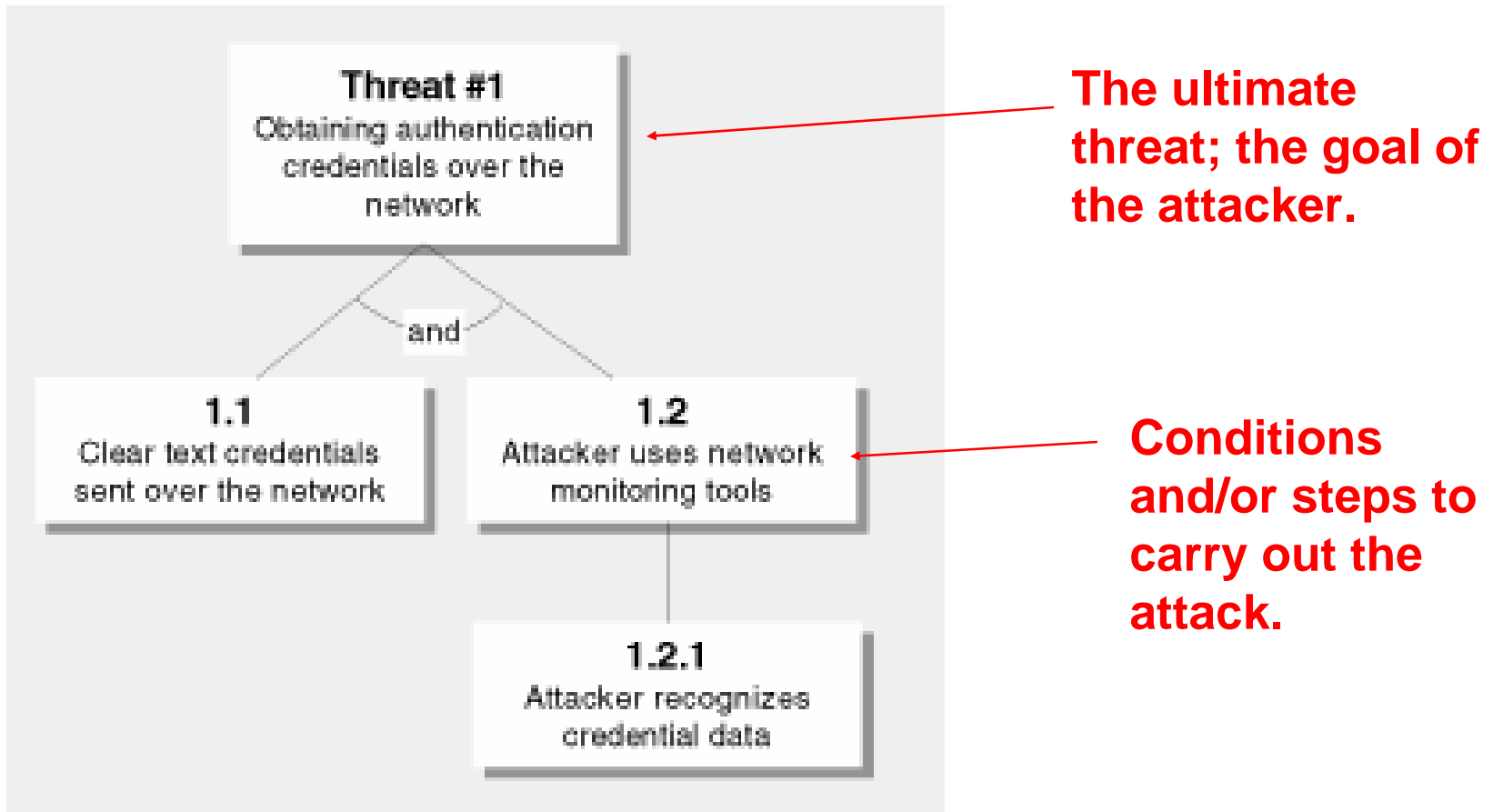
- A process must have at least one data flow entering and one data flow exiting.
- All data flows start or stop at a process.
- Data stores cannot connect together, they must pass through a process.
- **Process** names are verbs and nouns or verb phrases (e.g. Process stock symbol, evaluate exam grade)
- **Data flow** names are nouns or noun phrases (e.g. Stock price, Exam score)
- **Interactor** names are nouns (e.g. Stock broker)
- **Data store** names are nouns (e.g. Realtime stock data, Exam result data)

Microsoft STRIDE (six) threat categories

- **S**poofing identity – pose as another user
- **T**ampering with data – malicious modification of data
- **R**epudiation – can the action (prohibited action) be traced?
- **I**nformation disclosure – disclose of information to individuals who aren't supposed to have it
- **D**enial of service – deny access to valid users (e.g. consume all the CPU time)
- **E**levation of privilege – unprivileged user gains privileged access (becomes part of the trusted system)

Attack Tree

Describes the decision-making process an attacker would go through to compromise the component.



Microsoft Security Risk Management: DREAD

Table 3.6 Thread Rating Table

	Rating	High (3)	Medium (2)	Low (1)
D	Damage potential	The attacker can subvert the security system; get full trust authorization; run as administrator; upload content.	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be reproduced every time and does not require a timing window.	The attack can be reproduced, but only with a timing window and a particular race situation.	The attack is very difficult to reproduce, even with knowledge of the security hole.
E	Exploitability	A novice programmer could make the attack in a short time.	A skilled programmer could make the attack, then repeat the steps.	The attack requires an extremely skilled person and in-depth knowledge every time to exploit.
A	Affected users	All users, default configuration, key customers	Some users, non-default configuration	Very small percentage of users, obscure feature; affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable.	The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use.	The bug is obscure, and it is unlikely that users will work out damage potential.

After you ask the above questions, count the values (1-3) for a given threat. The result can fall in the range of 5-15. Then you can treat threats with overall ratings of 12-15 as High risk, 8-11 as Medium risk, and 5-7 as Low risk.

Consider your project

- Draw a DFD diagram.
 - Draw 2-3 attack trees, maybe based upon your scenario (misuse case).
 - Create test cases for those attack trees.
-
- **Resource:**
<http://www.microsoft.com/whdc/driver/security/threatmodel.mspx>

References

- Graff, Mark G. and van Wyk, Kenneth R., *Secure Coding: Principles and Practices*, O'Reilly, 2003.
- Howard, M. and LeBlanc, D., *Writing Secure Code*, Microsoft Press, 2003.
- Thompson, H., *Why Security Testing is Hard*, IEEE Security and Privacy, July/Aug 2003, pp. 83-86.
- Whittaker, J., *How to Break Software Security*, Addison Wesley, 2004.