EITP20: Secure Systems Engineering — Reference Sheet
Lund University

Karl Hallsby

Last Edited: March 10, 2020

Contents

List of Theorems

1 Threat Analysis

To perform any kind of threat analysis, the system's specification must be considered. The state of the system (new or existing) must also be considered.

- The specification of the system to be analyzed
  - If a completely new system is about to be built, there might not even exist a system specification and it needs to be created.
  - If the analysis is to done on an existing system, the first task is to read existing system specifications, source code and/or make interviews with people familiar with the system.
- The specification must not necessarily be very detailed but can be a high-level description of the system.

Defn 1 (Attack Goal)**.** An *attack goal* is a technology, process, or thing that an attacker would like to gain access to. These are the things we are trying to protect against or mitigate.

1.1 Attack Trees

Defn 2 (Attack Tree)**.** The Schneider *attack tree* method is a straight-forward and step-by-step type of attack analysis. However, it is quite basic, so it might not be the best method to perform this analysis.

The steps involved are:

1. The starting point is a *good* system description.
2. Next, Attack Goals are identified.
3. Attack Goals are then broken down to specific attacks to form a so-called attack tree. Identify different attack vectors for the same Attack Goal with several iterations.
4. Identify dependencies in the Attack Goals.
5. Once the attack tree is created, it is transferred to a table where scores on risks/costs can be added as well as more details.

In a Single Sign-On system (SSO), there any many different points of failure. These will be illustrated with a basic Attack Tree. Some Attack Goals for this system are:

1. Get access to all services provided by all entities in the system
   (a) Get access to all services provided to all users at the ID Provider
   (b) Get access to all services offered through an OIDC Client
   (c) Get access to all services offered at the Hosting Server
2. Get access to all services provided to a single user provided by the ID Provider
3. Get access to all services provided to single user at the OIDC Client
4. Get access to all services provided to a single user at the Hosting Service
5. Get access to all services provided to an OIDC Client at the Hosting Service

Now, we identify dependencies in the various Attack Goals.

- Attack Goal 1 depends on goals 2, 3, 4, and 5 being completed.
- Goal 1.1 depends on goal 2 being completed.
- Goal 1.2 depends on goal 3 being completed.
- Goal 1.3 depends on goals 4 and 5 being completed.
- Goal 5 depends on goal 4 being completed.

We can now break each of the smaller goals down into concrete attack vectors.

1. Access to all services provided to all entities in the system.
   1.1. Access to all services offered by the ID Provider to all users.
      1.1.1. Get KeyCloak admin rights.
         1.1.1.1. Successful phishing attack.
         1.1.1.2. Break administrator authentication.
      1.1.2. Root access on the KeyCloak server.
         1.1.2.1. Utilize an OS vulnerability
         1.1.2.2. Break server isolation (For example, Docker or VMs)
2. Access to all services provided to a single user by the ID Provider
   2.1. Successful end-user password phishing attack.

2.2. Take over the end-user's client device.
  2.2.1. Place malware on the client device.
  2.2.2. Successful network attack on the end-user's client device.
2.3. Break the end-user authentication mechanism.
  2.3.1. Break authentication algorithm. (Difficult, if not impossible)
  2.3.2. Find flaw in authentication protocol.
  2.3.3. Find flaw in authentication algorithm implementation on the client or on the server.

As this "tree" shows, the when accounting for attack vectors, it can become quite large. You should start with the obvious cases, and move onto the les obvious as you go. The most important thing while doing this is you should think like an attacker.

Remark. Depending on the information of the system available, a very detailed attack break-down might be possible or not. The tree can later be complemented when more implementation information is available

Remark. The attack tree is a useful tool, but is not a final or complete solution to a security analysis problem.

1.2 Example Application of Attack Trees

1.3 Microsoft STRIDE Analysis

Defn 3 (STRIDE Analysis)**.** *STRIDE* is an acronym standing for

S Spoofing
T Tampering
R Repudiation
I Information Disclosure
D Denial of Service
E Elevation of Privilege

It is a model to identify computer security threats in software systems or components of a larger software system. It is general enough to analyze any part of a software system.

To perform a STRIDE analysis, it is commonly applied with these steps:

1. Identify the main entities in the system
2. Identify the interactions between/among the main entities.
3. For each entity, perform a STRIDE analysis on the following actions:
   - Processes
   - External Entities
   - Data Flows
   - Data Stores

Remark 3.1 (Incomplete Methodology)*.* Like Attack Trees, STRIDE Analysis is not a complete methodology. It only helps identify typical attacks, but is not a "complete package".

1.3.1 Spoofing

Defn 4 (Spoofing)**.** *Spoofing* is pretending to be something or someone other than yourself. Table 1.1 gives one example, there are several others.

1. Spoofing the identity of an entity across a network. There is not mediating authority that takes reponsibility for telling that something is what it claims to be.
2. Spoofing the identity of an entity, even when there is a mediating authority. For example, a modified `.dll` file is mediated by the operating system to make sure it is the right one. If it's modified, th emediating authority might let it through, even when it has been modified.
3. Pretending to be a specific person, for example, Barack Obama.
4. Pretending to be in a specific role.

1.3.1.1 Spoofing a Process or File on the same Machine If an attacker creates a file before the real process, the process might interpret the file as being good data, and should be trusted. File permissions during a pipe operation, local procedure, etc. can create vulnerabilities.

Spoofing a process or file on a remote machine can work by creating the expected versions, or by pretending to be the expected machine.

Threat	Property Violated	Definition	Typical Victims	Example
Spoofing	Authentication	Pretending to be something or someone other than yourself	Processes, External Entities, People	Falsely claiming to be a police officer.
Tampering	Integrity	Modifying something on disk, on a network, or in memory	Data stores, data flows, processes	Changing spreadsheet, binary contents, database contents, modifying network packets, or the program running.
Repudiation	Non-Repudiation	Claiming that you didn't do something or were not responsible.	Process	Process: "I didn't hit the big red button".
Information Disclosure	Confidentiality	Providing information to someone not authorized to see it	Processes, data stores, data flows	Allow access to file contents/file metadata, network packets, contents of program memory.
Denial of Service	Availability	Deliberately absorbing more resources than needed to provide a service	Processes, data stores, data flows	Program uses all available memory, File fills up disk, too many network connections for real traffic.
Elevation of Privilege	Authorization	Allowing someone to do something they're not authorized to do.	Process	Allow normal user to execute code as administrator. Remote person can run code.

Table 1.1: STRIDE Acronym Definitions

Threat Examples	What the Attacker Does	Notes
Spoofing a process on the same machine	Creates a file before the real process	
	Renaming/Linking	Creating a trojan `su` and altering the `path`
	Renaming something	Naming your process `sshd`
Spoofing a file	Creates a file in the local directory	This can be a library, execcutable, or config file
	Creates a link and changes it	Change should happen between link being checked and being accessed
	Creates many files in expected directory	Automate file creation to fill the space of files possible
Spoofing a machine	ARP Spoofing	
	IP Spoofing	
	DNS Spoofing	Forward or Reverse
	DNS Compromise	Compromise TLD, registrar, or DNS operator
	IP Redirection	At the switch or router level
Spoofing a person	Sets e-mail display name	
	Takes over a real account	
Spoofing a role	Declares themselves to be that role	Sometimes opening a special account with a relevant name

Table 1.2: Spoofing Threats

1.3.1.2 Spoofing a Machine Depending on what has been spoofed, ARP, DNS, IP, anything else in the networking stack, the attacks can vary. Once the attackers have this setup in place, then they can continue spoofing, or perform a man-in-the-middle attack. They could also steal cryptographic keys.

With a spoofed machine, they can perform phishing attacks to steal information using sites that appear to be valid.

1.3.1.3 Spoofing a Person Typically, spoofing a person involves having access to the real person's digital account(s), then pretending to be the real person through another account.

1.3.2 Tampering

Defn 5 (Tampering)**.** *Tampering* is modifying something, typically on disk, on a network, or in memory. This can include changing the data in a spreadsheet, changing a binary or config file on dis, modifying a database, etc. On a network, packets can be added, modified, or removed.

Threat Examples	What the Attacker Does	Notes
Tampering with a File	Modifies a file they own and on which you rely	
	Modifies a file you own	
	Modifies a file on a file server you own	
	Modifies a file on their file server	Loads of fun when you include files from remote domains
	Modifies a file on their file server	XML schemas include remote schemas
	Modifies links or redirects	
Tampering with Memory	Modifies your code	Hard to defend against once attacker is running code as same user.
	Modifes data they've supplied to your API	Pass by value, not by reference when crossing a trust boundary
Tampering with a Network	Redirect flow of data to their machine	Often stage 1 of tampering
	Modifies data flowing over entwork	Even easier when the network is wireless (WiFi, 3G, LTE, etc.)
	Enhances spoofing attacks	

Table 1.3: Tampering Threats

1.3.2.1 Tampering with a File Attackers can create/modify files if they have write permission. If the code you wrote relies on files other things (people or files) wrote, there's a possibility it was written maliciously.

This can commonly happen when a website has been compromised and is serving malicious JavaScript. They can also modify links that the website gives you to be compromised as well. This same link manipulation can happen on your local disk as well.

1.3.2.2 Tampering with Memory Attacker can modify your code, if they are running at the same or higher privilege level. This is difficult to find and handle, since the bad code is running at the same privilege and under the same user as the good code, and it is near impossible to tell them apart.

If you handle data in a pass-by-reference fashion, the attacker can modify it after security checks have been performed.

1.3.2.3 Tampering with a Network There are a variety of tricks for this. The attacker can forward some data that is intact and some that is modified.

Other tricks use the radio communication used for wireless data transmission. Software-Defined Radio has made this type of attack trivial.

1.3.3 Repudiation

Defn 6 (Repudiation)**.** *Repudiation* is claiming you didn't do something, or were not responsible for what happened. This can be done honestly or deceptively.

Typically, this is done above any technical layer, and is instead in the business logic of buying products, for example.

Repudiation often deals with logs and logging, which tracks everything that happened in order to find who/what was/is responsible.

Threat Examples	What the Attacker Does	Notes
Repudiating an action	Claims to have not clicked	Maybe they really did
	Claims to have received	Maybe they did, but didn't read. Maybe cached. Maybe pre-fetched.
	Claims to have been a fraud victim	
	Uses someone else's account	
	Uses someone else's payment instrument without authorization	
Attacking the logs	Notices you have no logs	
	Puts attacks in logs to confuse logs, log-reading code, or a person	

Table 1.4: Repudiation Threats

1.3.3.1 Attacking the Logs If there is no logging, we don't retain logs, cannot analyze them, repudiation actions are difficult to dispute. There needs to be log centralization and analysis capabilities. There needs to be a clear definition of what is logged.

1.3.3.2 Repudiating an Action More useful to talk about "someone" than "an attacker" in this context. Since many people who perform some action that may need to have repudiation are usually people that have been failed by technology or a process.

1.3.4 Information Disclosure

Defn 7 (Information Disclosure)**.** *Information disclosure* is about allowing people to see information they are not authorized to see.

1.3.4.1 Information Disclosure from a Process A process will disclose information that helps further attacks. This can be done by leaking memory addresses, extracting secrets from error messages, or extracting design details from errors messages.

1.3.4.2 Information Disclosure from a Data Store Since data stores store data, they can leak it multiple ways:

- Failure to properly use security mechanisms
- Not setting permisions
- Cryptographic keys being found/leaked
- Files read over the network are readable over the network
- Metadata (filenames, author, etc.) is also important
- OS-level leaks when swapping things out
- Data extracted from things using OS under attacker control.

1.3.4.3 Information Disclosure from a Data Flow Data flows are particularly susceptible. Data flows on a single machine that is shared among multiple mutually distrustful uses of a system is a main susceptibility. Attackers can redirect information to themselves. Attackers can gain information even when the traffic is encrypted.

1.3.5 Denial of Service

Defn 8 (Denial of Service)**.** *Denial-of-service* attacks consume a resource that is needed to provide a service.

These attacks can be split into attacks that work while the attacker is actively attacking, and ones that occur persistently. These can also be amplified or unamplified. Amplified attacks are relatively small attacks that have big effects.

1.3.6 Elevation of Privilege

Defn 9 (Elevation of Privilege)**.** *Elevation of privilege* is allowing someone to do something they are not authorized to to. For example, allowing a remote user to execute code as an administrator, or allowing a remote person without privileges to run code.

Threat Examples	What the Attacker Does	Notes
Information disclosure against a process	Extract secrets from error messages	
	Reads error messages from username/passwords to entire database tables	
	Extract machine secrets from error messages	Makes defense against memory corruption less useful
	Extract business/personal secrets from error cases	
Information disclosure against data stores	Takes advantage of inappropriate or missing ACLs	
	Takes advantage of bad database permissions	
	Finds files protected by obscurity	
	Finds cryptographic keys on disk or in memory	
	Sees interesting information in metadata	
	Reads files as they traverse the network	
	Gets data from logs or temp files	
	Gets data from swap or other temp storage	
	Extracts data by obtaining device, changing OS	
Information disclosure against data flows	Reads data on network	
	Redirects traffic to enable reading data on network	
	Learn secrets by analyzing traffic	
	Learn who's talking to whom by watching DNS	
	Learn who's talking to whom by social network info disclosure	

Table 1.5: Information Disclosure Threats

Threat Examples	What the Attacker Does	Notes
Denial of service against a process	Absorbs memory	
	Absorbs storage	
	Absorbs CPU cycles	
	Uses process as an amplifier	
Denial of service against a data store	Fills data store up	
	Makes enough requests to slow down the system	
Denial of service against a data flow	Consumes network resources	

Table 1.6: Denial of Service Threats

1.3.6.1 Elevate Privileges by Corrupting a Process

Corrupting a process can involve:

- Smashing the stack
- Exploiting information on the heap
- Other techniques

These attacks allow the attacker to gain influence or control over a program's control flow.

Threat Examples	What the Attacker Does	Notes
Elevation of privilege against a process by corrupting the process	Send inputs that the code doesn't handle properly	These are very common and are usually high impact
	Gains access to read/write memory inappropriately	Writing memory is bad, but reading memory allows further attacks.
Elevation through missed authorization checks		
Elevation through buggy authorization checks		Centralizing such checks makes bugs easier to manage
Elevation through data tampering	Modifies bits on disk to do things other than what the authorized user intends	

Table 1.7: Elevation of Privilege Threats

1.3.6.2 Elevate Privileges through Authorization Failures If we don't check authorization on every path through a program, this can be a problem. If there are buggy checks, the attacker can take advantage of them. If a program relies on other programs, configuration files, or datasets being trustworthy, those are possible targets as well.

1.3.7 STRIDE-per-Interaction

This variant of STRIDE Analysis involves applying the traditional STRIDE Analysis to interations between elements in a system, rather than considering the system as a whole.

1.4 MITRE TARA Analysis

A Complex Numbers

Complex numbers are numbers that have both a real part and an imaginary part.

$$z = a \pm bi \tag{A.1}$$

where

$$i = \sqrt{-1} \tag{A.2}$$

Remark (i vs. j for Imaginary Numbers). Complex numbers are generally denoted with either i or j. Since this is an appendix section, I will denote complex numbers with i, to make it more general. However, electrical engineering regularly makes use of j as the imaginary value. This is because alternating current i is already taken, so j is used as the imaginary value instad.

$$Ae^{-ix} = A\left[\cos\left(x\right) + i\sin\left(x\right)\right] \tag{A.3}$$

A.1 Complex Conjugates

If we have a complex number as shown below,

$$z = a \pm bi$$

then, the conjugate is denoted and calculated as shown below.

$$\overline{z} = a \mp bi \tag{A.4}$$

Defn A.1.1 (Complex Conjugate)**.** The conjugate of a complex number is called its *complex conjugate*. The complex conjugate of a complex number is the number with an equal real part and an imaginary part equal in magnitude but opposite in sign.

The complex conjugate can also be denoted with an asterisk ($*$). This is generally done for complex functions, rather than single variables.

$$z^* = \overline{z} \tag{A.5}$$

A.1.1 Complex Conjugates of Exponentials

$$\overline{e^z} = e^{\overline{z}} \tag{A.6}$$

$$\overline{\log(z)} = \log(\overline{z}) \tag{A.7}$$

A.1.2 Complex Conjugates of Sinusoids

Since sinusoids can be represented by complex exponentials, as shown in Appendix B.2, we could calculate their complex conjugate.

$$\begin{aligned} \overline{\cos(x)} &= \cos(x) \\ &= \frac{1}{2}\left(e^{ix} + e^{-ix}\right) \end{aligned} \tag{A.8}$$

$$\begin{aligned} \overline{\sin(x)} &= \sin(x) \\ &= \frac{1}{2i}\left(e^{ix} - e^{-ix}\right) \end{aligned} \tag{A.9}$$

B Trigonometry

B.1 Trigonometric Formulas

$$\sin(\alpha) + \sin(\beta) = 2\sin\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right) \tag{B.1}$$

$$\cos(\theta)\sin(\theta) = \frac{1}{2}\sin(2\theta) \tag{B.2}$$

B.2 Euler Equivalents of Trigonometric Functions

$$e^{\pm j\alpha} = \cos(\alpha) \pm j\sin(\alpha) \tag{B.3}$$

$$\cos(x) = \frac{e^{jx} + e^{-jx}}{2} \tag{B.4}$$

$$\sin(x) = \frac{e^{jx} - e^{-jx}}{2j} \tag{B.5}$$

$$\sinh(x) = \frac{e^{x} - e^{-x}}{2} \tag{B.6}$$

$$\cosh(x) = \frac{e^{x} + e^{-x}}{2} \tag{B.7}$$

B.3 Angle Sum and Difference Identities

$$\sin(\alpha \pm \beta) = \sin(\alpha)\cos(\beta) \pm \cos(\alpha)\sin(\beta) \tag{B.8}$$

$$\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta) \tag{B.9}$$

B.4 Double-Angle Formulae

$$\sin(2\alpha) = 2\sin(\alpha)\cos(\alpha) \tag{B.10}$$

$$\cos(2\alpha) = \cos^2(\alpha) - \sin^2(\alpha) \tag{B.11}$$

B.5 Half-Angle Formulae

$$\sin\left(\frac{\alpha}{2}\right) = \sqrt{\frac{1-\cos(\alpha)}{2}} \tag{B.12}$$

$$\cos\left(\frac{\alpha}{2}\right) = \sqrt{\frac{1+\cos(\alpha)}{2}} \tag{B.13}$$

B.6 Exponent Reduction Formulae

$$\sin^2(\alpha) = \frac{1-\cos(2\alpha)}{2} \tag{B.14}$$

$$\cos^2(\alpha) = \frac{1+\cos(2\alpha)}{2} \tag{B.15}$$

B.7 Product-to-Sum Identities

$$2\cos(\alpha)\cos(\beta) = \cos(\alpha-\beta) + \cos(\alpha+\beta) \tag{B.16}$$

$$2\sin(\alpha)\sin(\beta) = \cos(\alpha-\beta) - \cos(\alpha+\beta) \tag{B.17}$$

$$2\sin(\alpha)\cos(\beta) = \sin(\alpha+\beta) + \sin(\alpha-\beta) \tag{B.18}$$

$$2\cos(\alpha)\sin(\beta) = \sin(\alpha+\beta) - \sin(\alpha-\beta) \tag{B.19}$$

B.8 Sum-to-Product Identities

$$\sin(\alpha) \pm \sin(\beta) = 2\sin\left(\frac{\alpha \pm \beta}{2}\right)\cos\left(\frac{\alpha \mp \beta}{2}\right) \tag{B.20}$$

$$\cos(\alpha) + \cos(\beta) = 2\cos\left(\frac{\alpha + \beta}{2}\right)\cos\left(\frac{\alpha - \beta}{2}\right) \tag{B.21}$$

$$\cos(\alpha) - \cos(\beta) = -2\sin\left(\frac{\alpha + \beta}{2}\right)\sin\left(\frac{\alpha - \beta}{2}\right) \tag{B.22}$$

B.9 Pythagorean Theorem for Trig

$$\cos^2(\alpha) + \sin^2(\alpha) = 1^2 \tag{B.23}$$

B.10 Rectangular to Polar

$$a + jb = \sqrt{a^2 + b^2}e^{j\theta} = re^{j\theta} \tag{B.24}$$

$$\theta = \begin{cases} \arctan\left(\frac{b}{a}\right) & a > 0 \\ \pi - \arctan\left(\frac{b}{a}\right) & a < 0 \end{cases} \tag{B.25}$$

B.11 Polar to Rectangular

$$re^{j\theta} = r\cos(\theta) + jr\sin(\theta) \tag{B.26}$$

C Calculus

C.1 Fundamental Theorems of Calculus

Defn C.1.1 (First Fundamental Theorem of Calculus)**.** The *first fundamental theorem of calculus* states that, if f is continuous on the closed interval $[a, b]$ and F is the indefinite integral of f on $[a, b]$, then

$$\int_a^b f(x)\,dx = F(b) - F(a) \tag{C.1}$$

Defn C.1.2 (Second Fundamental Theorem of Calculus)**.** The *second fundamental theorem of calculus* holds for f a continuous function on an open interval I and a any point in I, and states that if F is defined by

$$F(x) = \int_a^x f(t)\,dt,$$

then

$$\begin{aligned} \frac{d}{dx}\int_a^x f(t)\,dt &= f(x) \\ F'(x) &= f(x) \end{aligned} \tag{C.2}$$

Defn C.1.3 (argmax)**.** The arguments to the *argmax* function are to be maximized by using their derivatives. You must take the derivative of the function, find critical points, then determine if that critical point is a global maxima. This is denoted as

$$\underset{x}{\operatorname{argmax}}$$

C.2 Rules of Calculus

C.2.1 Chain Rule

Defn C.2.1 (Chain Rule)**.** The *chain rule* is a way to differentiate a function that has 2 functions multiplied together.
If

$$f(x) = g(x) \cdot h(x)$$

then,

$$\begin{aligned} f'(x) &= g'(x) \cdot h(x) + g(x) \cdot h'(x) \\ \frac{df(x)}{dx} &= \frac{dg(x)}{dx} \cdot g(x) + g(x) \cdot \frac{dh(x)}{dx} \end{aligned} \tag{C.3}$$

D Laplace Transform

Defn D.0.1 (Laplace Transform)**.** The *Laplace transformation* operation is denoted as $\mathcal{L}\{x(t)\}$ and is defined as

$$X(s) = \int_{-\infty}^{\infty} x(t)e^{-st}dt \tag{D.1}$$