Exam overview

These concepts are particularly important for the exam. When grading the exam, we will judge your answers based on these criteria:

- Completeness – did you mention all the most important / relevant points in the answer?
- Correctness – is your answer actually right?
- Clarity – are you using fancy words and terminology in a way that makes sense, or does it seem like you have no idea what you're talking about?'
- Relevance – did you answer the question, or are you mostly saying (perhaps eloquent and correct) stuff above irrelevant things?

If you are uncertain about the meaning of a question, please state how you're interpreting it, so we can understand your thinking.

For most of the questions, we expect fairly short answers (a few sentences, up to a paragraph or so). It's sometimes difficult to describe complicated things concisely, so it's ok to give a longer answer if you have to – but don't overdo it and write pages and pages, you (and we) won't have time for that. Feel free to use bullet points or lists and such to make things clearer if you like.

The exam is similar to previous years – there's a 7 pts multiple choice + four 12 pts and one 5 pts exercises with text answers (asking you to explain something or analyse some scenario). The weights are approximate (it's sometimes difficult to estimate the difficulty precisely), and for the final result we add (up to) 40 pts from the assignments.

Exam Attachments

Together with the exam, you'll find a simplified version of the overview below, as well as PDF 'printouts' of the 2021 OWASP Top Ten and the OWASP Web Security Cheat Sheet Series (from 2015, but it should still be relevant for the exam). The PDFs are attached as 'resources' that get opened in a separate tab. But: there are several hundred pages in total, so you can easily spend the whole exam reading – make sure you don't fall into that trap!

Overview of important terms/concepts

Auth*

Authentication - proving who you are, for example with a username and some authentication factor(s) like a password (something you know), a fingerprint (something you are) or a one-time code (something you have). OWASP A07:2021

Authorization – access control – what resources or data an authenticated user is permitted to read / change / delete / share / etc. OWASP A01:2021

Crypto, signatures, certificates OWASP A02:2021

Public-key cryptography – each party has a secret *private key* and a non-secret *public key* – both are huge numbers with a special relationship; to encrypt a message, you use your private key and the recipient's public key – the recipient will then be able to decrypt the message using your public key and their private key.

Public-key Infrastructure (PKI) – secure communication with public-key cryptography requires knowing the correct public key for the recipient (e.g., Bob). It's safe to share this key publicly, but if someone else (e.g., Eve) tricks you into using their public key instead of Bob's, Eve can intercept the messages and pretend to be Bob. PKI is a trusted way to connect public keys with identities.

Digital signature – uses public key cryptography to sign a message or document; the signature proves that the message was signed by someone who knows the private key belonging to a particular public key, and that the message hasn't been changed after signing.

Digital or Public-key certificate – ties a public key to the identity of its owner, by having the key and identity signed by some trusted party (the Certificate Authority). Web pages typically have such certificates, either based on checking actual credentials of the server operator (identity documents, company ownership, etc), or by proving to the CA that you have control over the server (used for free certificates). Can also be used for encrypted email, logging in to a remot server, securely connecting two servers, and other cases where proving identity is important.

General security concepts

- Threat modelling

CIA-T

Confidentiality *(konfidensialitet)* – not leaking/exposing private or secret data

Integrity *(integritet)* – data is protected against unauthorized or unintended changes; data is accurate, consistent and complete. A08:2021

Availability *(tilgjengelighet)* – the system is available / working properly when needed by its intended users

Traceability *(sporbarhet)* – tracking access to, changes to, and availability of the system; who accessed this patient record? why is my bankaccount suddenly full of money? what happened when the server went down? A09:2021; note that useful logs often contain sensitive information (e.g., a user mixing up username and password would have the password in plain text in the log file, probably quite close to the username), and logs might be shared unintentionally (e.g., by server misconfiguration or in a bug report).

Non-repudiation *(ikke-benekting)* – making sure that a party can't deny or disown their actions (or non-actions); proving that someone sent a message, made a bank transaction, failed to deliver an assignment on time, actually authored a document, etc

STRIDE threat model

Spoofing – a party impersonating or pretending to be someone else; e.g., false authentication, tricking a user or a system into thinking they're someone else; A01:2021

Tampering – making uauthorized changes (violating integrity)

Repudiation *(benekting)* – dispute authorship or validity of some action or claim; e.g., “My dog sent my tweet”

Information disclosure – violating confidentiality

Denial of service – violating availability; e.g., by overloading a server, crashing it, or tricking it into blocking a valid user

Elevation of privilege – bypassing authorization; e.g., running code as administrator after gaining access as a user, or breaking out of a container / virtualized machine / sandbox (e.g., a script accessing the users files from within a browser would be breaking the browser's sandbox and elevating its privilege)

DREAD risk assessment

Damage – how bad would an attack be?

Reproducibility – how easy is it to reproduce the attack?

Exploitability – how much work is it to launch the attack?

Affected users – how many people will be impacted?

Discoverability – how easy is it to discover the threat?

Web terms

Cookie – small bit of data stored in the browser and sent with each request to a particular server (according to the SameSite security setting)

Cross-site Request Forgery (CSRF) – getting a user to click on a form/link on your site, leading the browser to submit data to the another site with the user's credentials

CSRF token – a random, short-lived value added to forms to avoid reusing the form in a CSRF attack

Cross-site scripting (XSS) – injecting script code into a web site, tricking the browser into running the code in the context of that page; for example, by putting script code in user content or in a link with request parameters; this is a very common vulnerability, and lets an attacker read or tamper with the user's data on the vulnerable web page.

Cross-origin Resource Sharing (CORS) – to prevent misuse of data and various cross-site attacks, the browser will prevent loading certain resources or making calls to site that don't share the same origin – this could, for instance, be problematic when the user has private data on the other site, particularly with a lax cookie SameSite policy. With the CORS header, web servers can permit such cross-origin requests in cases where it's safe (e.g., accessing data that's public anyway).

Origin – where a web page was loaded from; the origin is the same if the scheme (e.g., https), host (e.g., www.example.com) and port number (e.g., 443) are the same.

Same-origin policy - scripts can only access data (e.g., accessing the document tree) from an other page if they share the same origin

OAuth 2.0 – an authorization system that works by a web application (the client app) sending the user (the resource owner) to another web page (the resource server); the resource server authenticating the user and then asking (or having previously asked) the user for permission to share some resource with the client app; and then redirecting the user back to the original site with a special token that lets the client app make requests to the resource server on behalf of the user. Often used for authentication instead, with OpenID Connect. Since the user interacts with the resource server directly in the web browser, the usual security context (session cookies for logged in users; stored passwords for the site) is available, and the client app has no access to the users credentials on the resource server (thanks to the same-origin policy).

OpenID Connect (OIDC) – authentication built on top of OAuth 2.0; the user authorizes the client app to access their identity information (username, perhaps email) from another server (the identity provider; e.g., Google, Facebook, etc) – the fact that the identity provider grants access to the user's identity proves that the user is properly authenticated against the identity provider (and/or the provider sends a signed response certifying the user's identity).

SameSite cookie policy – whether cookies should be sent with requests made to other domains (third-party cookies); a Strict

SameSite policy helps prevent CSRF attacks

HTTP – the protocol used to access webpages – the client sends the host name, method and requested path, along with a number of optional headers, and gets headers and a payload in response. A misconfigured web server might allow access to important data such as password lists, log files, confidential files.

HTTPS – HTTP over an encrypted channel (TLS) – the server needs a certificate signed with a Certificate Authority (CA) to prove its identity, the browser comes with a list of trusted CAs. Some browser features may only be available on pages loaded over https; sites can also specify that they will only ever be available over https (preventing someone from setting up a fake http site); you should avoid mixing content served over https and http. Misconfiguring the web server can defeat the purpose of https by allowing the use of older, insecure algorithms or older versions of the protocol.

Attacks

See the OWASP Top Ten list.

Buffer overflow – overwriting data in memory, e.g., by inputting more data than expected; very dangerous, but less common with type and memory safe languages – in such languages there is no way to write code that reads or writes memory outside declared variables (for example, in Java, you can refer to objects, but there is no way to refer to or directly access the memory where the object is stored, or talk about the memory locations before or after an object in memory.)

Stack smashing – a form of buffer overflow where data in the stack frame is overwritten; for example, changing the return address so program flow is redirected; with return oriented programming, you can trick the program into running more or less arbitrary code. A stack canary helps protect against stack smashing; modern OS and CPU-level features can also prevent such attacks by detecting if the chain of return addresses has been changed, or if the program is jumping to code that isn't supposed to be the target of a jump.

Injection attacks, **SQL Injection** – very common and dangerous attack where user-supplied data isn't validated properly, and mixed with queries, code, styles or similar, so that a program is tricked into running user-supplied code. SQL injections are particularly famous. Prevented by keeping validating data, and making sure user-supplied strings aren't combined with code that will get executed (such as SQL queries, or HTML code – or even printf formatting strings). SQL prepared statements avoid this since the prepared statement string is parsed separately from the supplied data.

Programming terms

Encapsulation *(innkapsling)* – when implementations are hidden and only accessible through a well-defined interface. For example, objects have hidden state (fields) and are bundled with methods (the implementation) that manipulate the state - any interaction happens by calling the methods and other parts of the program see only the method signatures (a well-defined interface)

Immutability *(immutabilitet)* – preventing the modification of object state after construction; makes it easier to reason about a program, particularly when there are multiple references to objects; requires strong encapsulation to avoid unintentional modification of object state.

Type safety *(typetrygghet)* – whether the programming language prevents type errors (statically or dynamically)

Type error *(typefeil)* – misinterpreting data; for example, an integer used as a pointer/reference, trying to call methods that an object doesn't support, passing the wrong kind of object/data as an argument

Memory safety *(minnetrygghet)* – (part of type safety) whether the programming language prevents reading/writing data to/from arbitrary memory locations

Static (or lexical) typing *(statisk typing)* – most or all type errors are discovered at compile time; some runtime checks may also be needed, e.g., in the case of casts in Java / SSL / TLS

Duck typing *(gakketyping)* – an object is considered to be of the correct type if it supports the operations/methods you try to use on it (“if it looks like a duck, walks like a duck and quacks like a duck, it's probably a duck”). For example, Python, and C++ templates are duck typed.

Dynamic typing *(dynamisk typing)* – most or all type errors are discoverd at run time, e.g., by throwing TypeError exceptions or MethodNotFound or similar; it may still be possible to predict or eliminate some of the errors at compile time, though; the actual typing rules could be stronger or weaker than with a static language (e.g., Python is stricter than Java about some type errors, JavaScript is very ‘flexible’ with its typing rules)

Dynamic programming language *(dynamisk programmeringsspråk)* – where it's easy to inspect, load or change code while the program is running; typically dynamically typed. For example, Python, Lisp or JavaScript (Java and C, etc. *can* do this, but it's not convenient)

Call stack – contains the stack frames (with local variables, return addresses and such) and saved temporary data for all currently active functions/methods. (*Nested* functions may need to have access to more than one stack frame, and *closures* (using functions with bound variables as values) requires activation records that are store on the heap, since the closure may live longer than the current function call.)