Welcome to

# 7. Web Application Security: Defensive, Architecture, Authentication

## Security in Web Development Elective, KEA

Henrik Kramselund he/him han/ham hkj@zencurity.com @kramse

Slides are available as PDF, kramse@Github

7-defensive-architecture-security-in-web.tex in the repo security-courses

# Goals for today


Todays goals:

- Doing defensive
- Authentication on the web

Photo by Thomas Galler on Unsplash

# Plan for today

## Subjects

- Overall securing modern web applications
- Processes involved in creating secure web applications
- Vulnerability management
- Mitigation Strategies

## Exercises

- Github security features
- Password hashing speed
- Architecture exercise related to your exam project

# Time schedule

- 1) 17. Securing Modern Web Applications 45min
- 2) 18. Secure Application Architecture 45min
- 3) OWASP Security by Design Principles 45min
- 4) Architecture exercise 45min

# Reading Summary

*Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

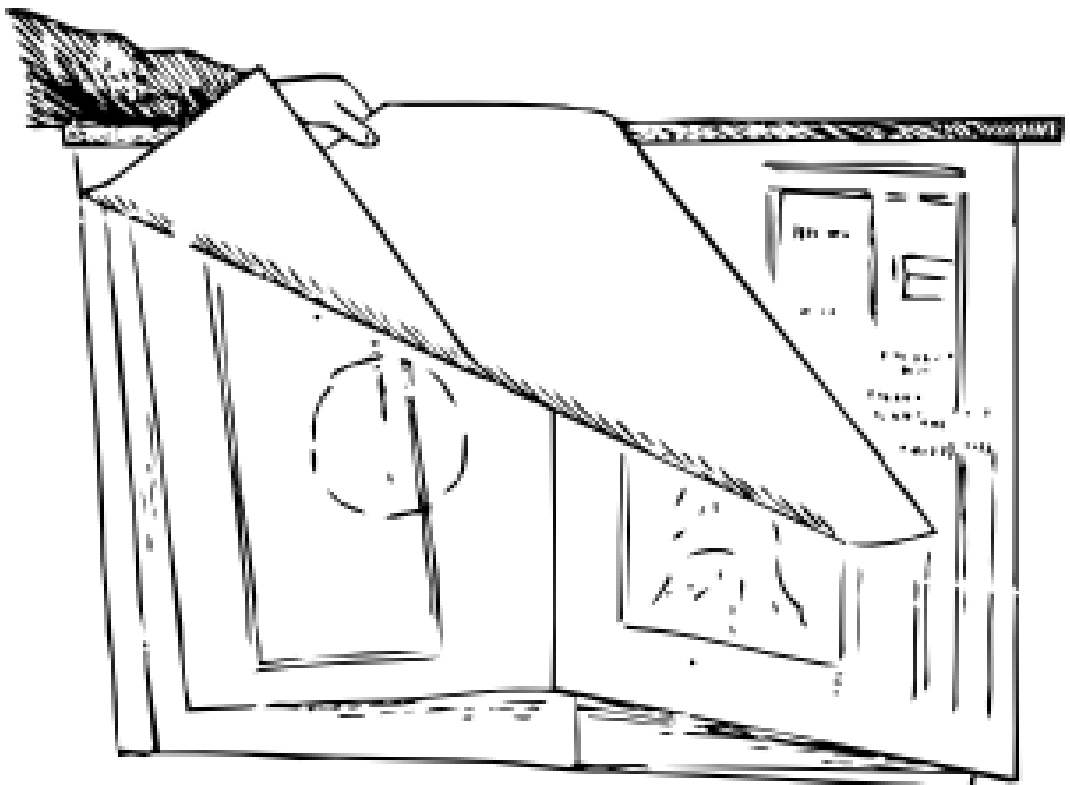
Part III. Defense, chapters 17-18

17. Securing Modern Web Applications

18. Secure Application Architecture

Security by Design Principles, OWASP
https://wiki.owasp.org/index.php/Security_by_Design_Principles

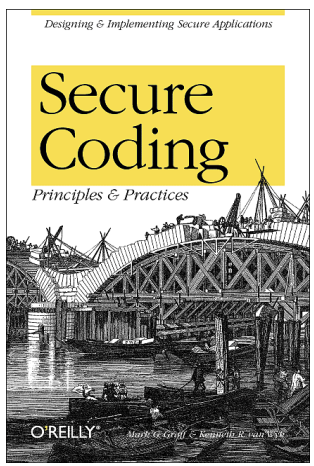
# Design for security - more work



Security is cheapest and most effective when done during design phase.

# Secure Coding starts with the design

*Secure Coding: Principles and Practices* af Mark G. Graff, Kenneth R. Van Wyk 2003




Architecture/design – while you are thinking about the application

Implementation – while you are writing the application

Operations – After the application is in production

Ca. 200 pages, very dense.

# Design vs Implementation

Software vulnerabilities can be divided into two major categories:

- Design vulnerabilities
- Implementation vulnerabilities

Even with a well-thought-out security design a program can contain implementation flaws.

# 17. Securing Modern Web Applications

Up to this point, we have spent a significant amount of time analyzing techniques that can be used for researching, analyzing, and breaking into web applications. These preliminary techniques are important in their own right, but also give us important insights as we move into the third and final part of this book: **defense**.

**Today's web applications are much more complex and distributed than their predecessors**. This opens up the surface area for attack when compared to older, monolithic web applications—in particular, those with server-side rendering and little to no user interaction. These are the reasons I structured this book to start with **recon, followed by offense, and finally defense**.

...

All of the skills and techniques we have covered up until this point are synergistic. Improving your mastery of recon, offense, or defense will result in extremely efficient use of your time.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- Offense informs defense, we should know how attackers work

# Defending a Castle

Defending a web application is somewhat akin to defending a medieval castle. A castle consists of a number of buildings and walls, which represent the core application code. Outside of the castle are a number of buildings that integrate with and support the castle's owner (usually a lord) in a way that describes an application's dependencies and integrations. Due to the large surface area in a castle and the surrounding kingdom, in wartime it is essential for defenses to be prioritized as it would be infeasible to maximize the defensive fortifications at every potential entrance point.

In the world of **web application security, such prioritization and vulnerability management is often the job of security engineers in large corporations or more generalized software engineers in smaller companies**. These professionals take on the role of **master defender**, using **software engineering skills in combination with recon and hacking skills** to reduce the probability of a successful attack, mitigate potential damages, and then manage active or past damages.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- Vulnerability Discovery – make it easy to report
  https://github.blog/2021-11-02-blue-teaming-create-security-advisory-process/

# Vulnerability Discovery

- Bug bounty programs
- Internal red/blue teams
- Third-party penetration testers
- Corporate incentives for engineers to log known vulnerabilities
- Letting your users find the bugs, and report them ... may not be the best way

# Vulnerability Management

After **assessing the risk of a vulnerability, and prioritizing it** based on the factors listed, a fix must be **tracked through to completion**. Such fixes should be completed in a timely manner, with deadlines determined based off of the risk assessment. Furthermore, customer contracts should be analyzed in response to an assessed vulnerability to determine if any agreements have been violated. Managing vulnerabilities is an **ongoing process**. Your vulnerability management process should be carefully planned out and written down so that your progress can be recorded. This should result in more accurate timelines as time goes on and time-to-fix burn rates can be averaged.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- Hint: use a bug tracker!

# Mitigation Strategies

Finally, an overall best practice for any security-friendly company is to actively make a good effort to mitigate the risk of a vulnerability occurring in the application code- base. This is a practice that happens all the way from the architecture phase to the regression testing phase. Mitigation strategies should be widespread, like a net trying to catch as many fish as possible. In crucial areas of an application, mitigation should also run deep.

Mitigation comes in the form of:

- secure coding best practices
- secure application architecture
- regression testing frameworks
- secure software development life cycle (SSDL)
- secure-by-default developer mindset and development frameworks.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

(Bullet list created by me)

# Git getting started

**Hints:**
Browse the Git tutorials on https://git-scm.com/docs/gittutorial
and https://guides.github.com/activities/hello-world/

- What is git
- Terminology

Note: you don't need an account on Github to download/clone repositories, but having an acccount allows you to save repositories yourself and is recommended.

# Github secure open source software

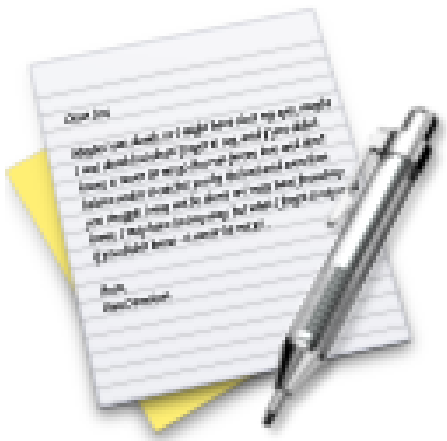
GitHub Advisory Database, vulnerable dependency alerts, and Dependabot One of the key elements of identifying security issues is working with a rich database of vulnerabilities. GitHub's dependency vulnerability detection tools use a combination of data directly from GitHub Security Advisories and the National Vulnerability Database (NVD) to create a complete picture of vulnerabilities in open source. This combined dataset lives in the GitHub Advisory Database and powers Dependabot alerts and security updates. The Advisory Database is also under a Creative Commons Attribution 4.0 license, meaning it's freely available for anyone to use as long as they attribute GitHub as the data source.

Source: Github article

Check out the things Github can provide, examples:

- https://resources.github.com/whitepapers/How-GitHub-secures-open-source-software/
- https://github.blog/2022-01-13-open-source-software-security-summit-securing-the-worlds-code-together/
- https://github.blog/2021-12-06-write-more-secure-code-owasp-top-10-proactive-controls/
- https://github.blog/2021-11-02-blue-teaming-create-security-advisory-process/

# Exercise



Now lets do the exercise

## Github secure open source software 15 min

which is number **36** in the exercise PDF.

# 18. Secure Application Architecture

When building a product, a cross-functional team of software engineers and product managers usually collaborate to find a technical model that will serve a very specific business goal in an efficient manner. In **software engineering, the role of an architect is to design modules at a high level and evaluate the best ways for modules to communicate with each other**. This can be extended to determining the best ways to store data, what third-party dependencies to rely on, what programming paradigm should be predominant throughout the codebase, etc.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- May be more than 100 times as costly to remove a design flaw later on!
- Book references NIST 30-60 times less when removing a vulnerability

# Analyzing Feature Requirements

In order to distribute merchandise under the new MegaMerch brand, MegaBank would like to set up an ecommerce application that meets the following requirements:

- Users can create accounts and sign in.
- User accounts contain the user's full name, address, and date of birth.
- Users can access the front page of the store that shows items.
- Users can search for specific items.
- Users can save credit cards and bank accounts for later use.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- I find it particularly interesting to identify features that *all applications* need

# Data storage

A high-level analysis of these requirements tells us a few important tidbits of information:

- We are storing credentials.
- We are storing personal identifier information.
- Users have elevated privileges compared to guests.
- Users can search through existing items.
- We are storing financial data.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

Most applications today store some sensitive information.

# Risk in Web Applications

A few of the risk areas derived from this analysis are as follows:

- Authentication and authorization: How do we handle sessions, logins, and cookies?
- Personal data: Is it handled differently than other data? Do laws affect how we should handle this data?
- Search engine: How is the search engine implemented? Does it draw from the primary database as its single source of truth or use a separate cached database?

Each of these risks brings up many questions about implementation details, which provide surface area for a security engineer to assist in developing the application in a more secure direction.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

# Goals: Data Security

## Nine principles of data security

(1) **Access control**—Each identifiable clinical record shall be marked with an access control list naming the people or groups of people who may read it and append data to it. The system shall prevent anyone not on the list from accessing the record in any way.

(2) **Record opening**—A clinician may open a record with herself and the patient on the access control list. When a patient has been referred she may open a record with herself, the patient, and the referring clinician(s) on the access control list.

(3) **Control**—One of the clinicians on the access control list must be marked as being responsible. Only she may change the access control list and she may add only other health care professionals to it.

(4) **Consent and notification**—The responsible clinician must notify the patient of the names on his record's access control list when it is opened, of all subsequent additions, and whenever responsibility is transferred. His consent must also be obtained, except in emergency or in the case of statutory exemptions.

(5) **Persistence**—No one shall have the ability to delete clinical information until the appropriate time has expired.

(6) **Attribution**—All accesses to clinical records shall be marked on the record with the name of the person accessing the record as well as the date and time. An audit trail must be kept of all deletions.

(7) **Information flow**—Information derived from record A may be appended to record B if and only if B's access control list is contained in A's.

(8) **Aggregation control**—Effective measures should exist to prevent the aggregation of personal health information. In particular, patients must receive special notification if any person whom it is proposed to add to their access control list already has access to personal health information on a large number of people.

(9) **Trusted computing base**—Computer systems that handle personal health information shall have a subsystem that enforces the above principles in an effective way. Its effectiveness shall be evaluated by independent experts.

Source: *Clinical system security: Interim guidelines*, Ross Anderson, 1996

# Authentication and Authorization

Because we are storing credentials and offering a different user experience to guests and registered users, we know we have both an authentication and an authorization system. This means we must allow users to log in, as well as be able to differentiate among different tiers of users when determining what actions these users are allowed. Furthermore, because we are storing credentials and support a login flow, we know there are going to be credentials sent over the network. These credentials must also be stored in a database, otherwise the authentication flow will break down. This means we have to consider the following risks:

- How do we handle data in transit?
- How do we handle the storage of credentials?
- How do we handle various authorization levels of users?

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

Generic recommendation is NOT to build it yourselves, but for the exam project we need you to demonstrate parts. So you will realize this is quite hard to get right.

# Secure Credentials and Hashing Credentials

- Checking against top one thousand password list and reject popular passwords
- Hashing – of course
- I recommend looking into https://haveibeenpwned.com/ and then using their services and perhaps API for improving password security. You can even download the list of passwords, and check before allowing use of a password.
  https://haveibeenpwned.com/Passwords
- Book describes bcrypt https://en.wikipedia.org/wiki/Bcrypt and PBKDF2 https://en.wikipedia.org/wiki/PBKDF2
- See also for alternatives the Password Hashing Competition
  https://en.wikipedia.org/wiki/Password_Hashing_Competition
- Book mentions 2FA, but too short. Recommended read https://en.wikipedia.org/wiki/Multi-factor_authentication

This part of the book recommend Transport Layer Security and Let's Encrypt, which I agree too.

# Attacking Authentication

Passwords are NOT chosen randomly

**The 50 Most Used Passwords**

| | | | | |
|---|---|---|---|---|
| 1. 123456 | 11. 123123 | 21. mustang | 31. 7777777 | 41. harley |
| 2. password | 12. baseball | 22. 666666 | 32. f*cky*u | 42. zxcvbnm |
| 3. 12345678 | 13. abc123 | 23. qwertyuiop | 33. qazwsx | 43. asdfgh |
| 4. qwerty | 14. football | 24. 123321 | 34. jordan | 44. buster |
| 5. 123456789 | 15. monkey | 25. 1234...890 | 35. jennifer | 45. andrew |
| 6. 12345 | 16. letmein | 26. p*s*y | 36. 123qwe | 46. batman |
| 7. 1234 | 17. shadow | 27. superman | 37. 121212 | 47. soccer |
| 8. 111111 | 18. master | 28. 270 | 38. killer | 48. tigger |
| 9. 1234567 | 19. 696969 | 29. 654321 | 39. trustno1 | 49. charlie |
| 10. dragon | 20. michael | 30. 1qaz2wsx | 40. hunter | 50. robert |

Source: https://wpengine.com/unmasked/

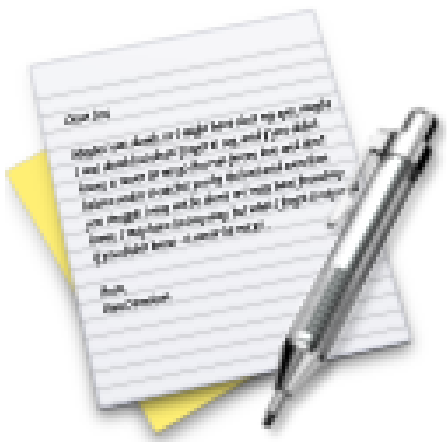
If doing password attacks, there is a nice RockYou password list on Kali Linux

# Demo/exercise: hashcat or John the Ripper

```
user@KaliVM:~$ john --test
Will run 4 OpenMP threads
Benchmarking: descrypt, traditional crypt(3) [DES 256/256 AVX2]... (4xOMP) DONE
Many salts: 32784K c/s real, 8460K c/s virtual
Only one salt: 27082K c/s real, 6873K c/s virtual
...
```

If you want to try cracking passwords yo can grab sample hashes from your local system or https://hashcat.net/wiki/doku.php?id=example_hashes

# Exercise



Now lets do the exercise

## PHP Passwords 15 min

which is number **37** in the exercise PDF.

# Summary: Secure Application Architecture

At the beginning of this chapter, I included the estimate from NIST that a security flaw found in the architecture phase of an application could cost 30 to 60 times less to fix than if it is found in production. This can be because of a combination of factors, including the following: ...

- Deep architecture-level security flaws may require rewriting a significant number of modules, in addition to the insecure module. For example, a complex 3D video game with a flawed multiplayer module may require rewriting of not only the networking module, but the game modules written on top of the multiplayer net- working module as well. This is especially true if an underlying technology has to be swapped out to improve security (moving from UDP or TCP networking, for example).

...

Ultimately, the ideal phase to catch and resolve security concerns is always the archi- tecture phase. Eliminating security issues in this phase will save you money in the long run, and eliminate potential headaches caused by external discovery or publica- tion later on.

-

# OWASP top ten


The OWASP Top Ten provides a minimum standard for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are.

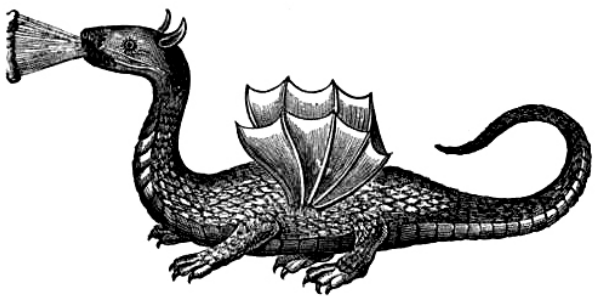
The Open Web Application Security Project (OWASP)

OWASP produces lists of the most common types of errors in web applications

http://www.owasp.org

Create Secure Software Development Lifecycle

# Goals: Secure Software



Here be dragons

- Software is insecure
- How do we improve quality
- Higher quality is more stable, and more secure
- Make sure to test specifically for security issues

We talked about security design with Qmail and Postfix recently.

# Software Development Lifecycle

A full lifecycle approach is the only way to achieve secure software.
–Chris Wysopal

- Often security testing is an afterthought
- Vulnerabilities emerge during design and implementation
- Before, during and after approach is needed

# Secure Software Development Lifecycle

- SSDL represents a structured approach toward implementing and performing secure software development
- Security issues evaluated and addressed early
- During business analysis
- through requirements phase
- during design and implementation

# Functional specification needs to evaluate security

- Completeness
- Consistency
- Feasibility
- Testability
- Priority
- Regulations

Source: The Art of Software Security Testing Identifying Software Security Flaws Chris Wysopal
ISBN: 9780321304865

# Phases of SSDL

- Phase 1: Security Guidelines, Rules, and Regulations
- Phase 2: Security requirements: attack use cases
- Phase 3: Architectural and design reviews/threat modelling
- Phase 4: Secure coding guidelines
- Phase 5: Black/gray/white box testing
- Phase 6: Determining exploitability

# Application Software Security

CIS Control 18:
Application Software Security
Manage the security life cycle of all in-house developed and acquired software in order to prevent, detect, and correct security weaknesses.

Source: Center for Internet Security CIS Controls 7.1 `CIS-Controls-Version-7-1.pdf`

CIS Control 16:
Application Software Security
Manage the security life cycle of in-house developed, hosted, or acquired software to prevent, detect, and remediate security weaknesses before they can impact the enterprise.

Source: Center for Internet Security CIS Controls v8 `CIS_Controls_v8_Guide.pdf`

# OWASP: Security by Design Principles

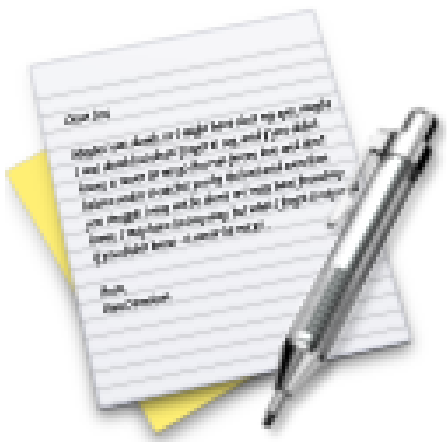
Security architecture Applications without security architecture are as bridges constructed without finite element analysis and wind tunnel testing. Sure, they look like bridges, but they will fall down at the first flutter of a butterfly's wings. The need for application security in the form of security architecture is every bit as great as in building or bridge construction.

Application architects are responsible for constructing their design to adequately cover risks from both typical usage, and from extreme attack. Bridge designers need to cope with a certain amount of cars and foot traffic but also cyclonic winds, earthquake, fire, traffic incidents, and flooding. Application designers must cope with extreme events, such as brute force or injection attacks, and fraud. The risks for application designers are well known. The days of "we didn't know" are long gone. Security is now expected, not an expensive add-on or simply left out.

We will now continue one the guide from OWASP

https://wiki.owasp.org/index.php/Security_by_Design_Principles

# Exercise

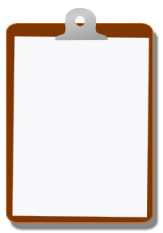


Now lets do the exercise

## Create an Architecture Drawing 45 min

which is number **38** in the exercise PDF.

# For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools