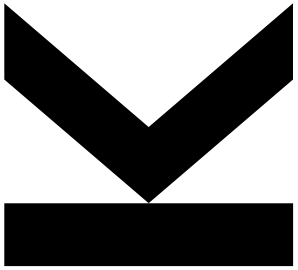
Introduction to Web Security



JOHANNES KEPLER
UNIVERSITÄT LINZ

Michael Sonntag
Institute of Networks and Security

INSTITUTE
OF NETWORKS
AND SECURITY

Why attack web applications/servers?

- Ubiquity: every company has a web server; many a shop, Web 2.0, IoT etc → If not here, try the next few million sites!
- Simple techniques: protocols are simple & attacks are widely known
  - Text input → manipulation is easy and many tools exist/custom programming trivial
  - OS buffer overflow: compare this to SQL injection (find & exploit)!
    - Buffer overflow: write/apply machine (assembly) code, know about OS internals (addresses of calls, how to call them/load libraries, get around ASLR/NX/…)
    - SQL injection: write a brief and simple line of text; try variations
- Anonymity: you don't need to be there; proxies for HTTP(S) are everywhere, see also Tor for anonymization
- Firewall bypassing: web traffic is practically always allowed in both directions: in & out

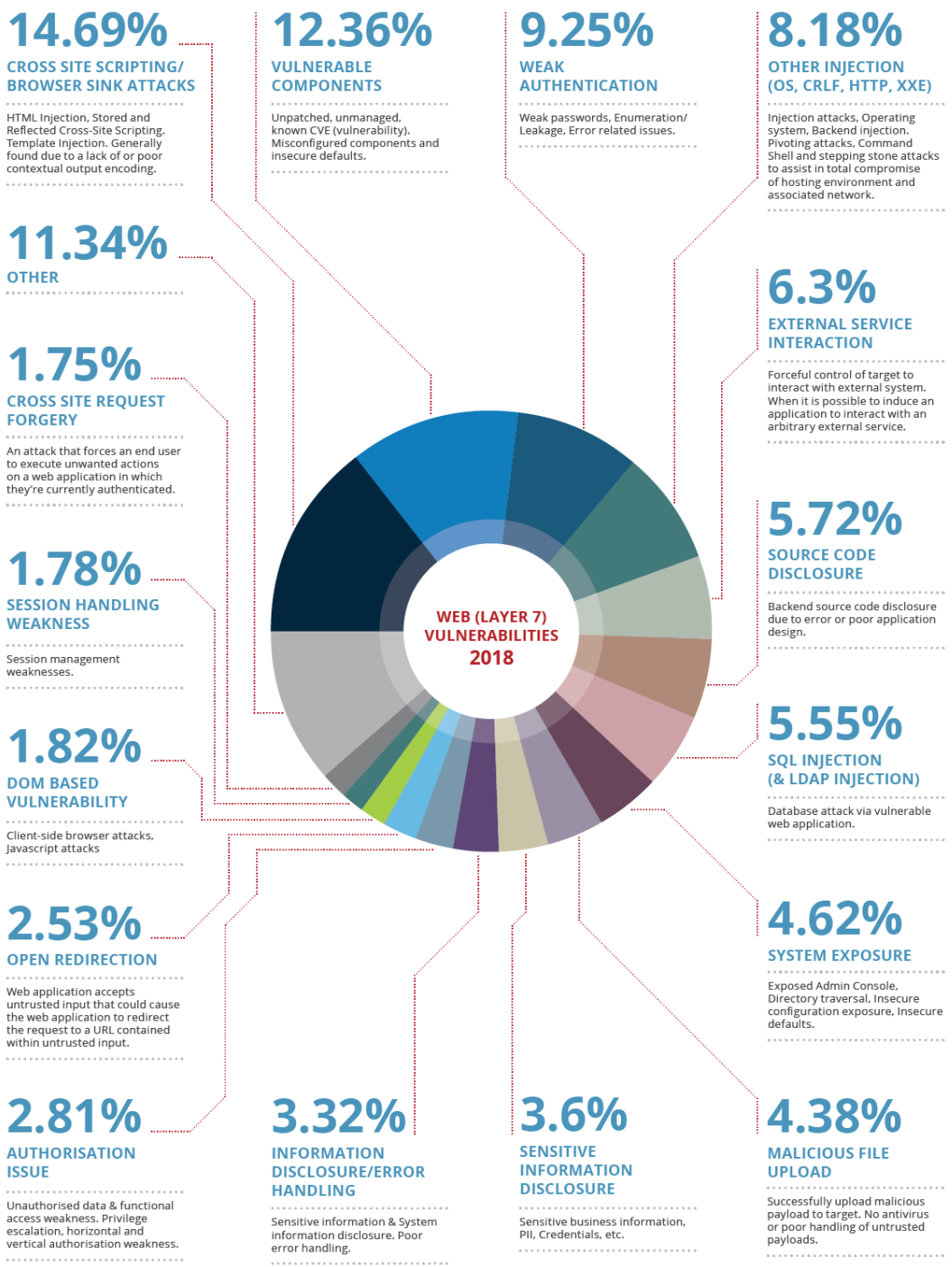
Why attack web applications/servers?

- Custom code: web programming is simple, so "simpletons" do it!
  - No security education for them!
  - Easy to do → simply „hacked“ (+ „security is for later!“)
- Immature security: no sessions, authentication weak: "Do everything yourself“
  - Instead of learning and using a complex framework
    - Complex framework → easy to make a mistake, needs updating…
- Constant change: numerous persons always change the web application and/or the web content
  - OS: Producer; Software: rare updates only
- Lots of websites
  - Keeping all of them continuously updated is a lot of work
    - And not very interesting, bur rather boring
  - Many applications are converted to web services
  - Cloud computing is managed/often based on web pages

Statistics

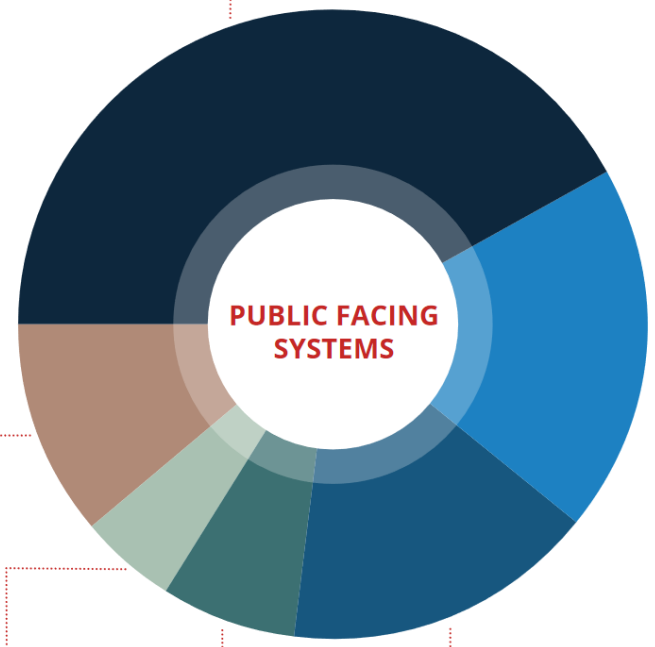
- Biggest problems:
  - 14,7% CSS
  - 12,4% Unpatched/unmanaged
  - 11,3% Mixed other
  - 9% Authentication problem
  - 8% Injection
  - 6,9% Information disclosure
  - 6% Force interaction with other external system
  - 5,7% Source code disclosure
  - 5,6% SQL injection
  - 4,6% Bad design/configuration
  - 4,4% File upload problems
  - 2,8% Authorization problems
  - 2,5% Redirects
  - 1,8% Client-side attacks
  - 1,8% CSRF
- Web vulnerabilities only!
- Source: Edgescan 2019 Vulnerability Statistics Report
- https://www.edgescan.com/?smd_process_download=1&download_id=2057

JKU INSTITUTE OF NETWORKS AND SECURITY



Statistics

- Note: All public facing systems, not solely web related
- But (almost) web-only are:
  - ☐ SQL injection
  - ☐ XSS
  - ☐ PHP



42% SQL INJECTION

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.

A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.

SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

19% CROSS-SITE SCRIPTING (XSS)

Cross site Scripting (XSS) attacks are a type of injection problem, in which malicious scripts are injected into web sites. Cross site scripting flaws are the most prevalent flaw in web applications today. Cross site scripting attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser-side script, to a different end user.

The 'stored' variant is considered a "Critical" vulnerability as it persists across all users who access an infected page and has the potential to infect a wide user base of the web application or site.

11% OTHER

5% SENSITIVE FILE DISCLOSURE

This is the result of leaving unprotected files on a hosting environment, systems using inadequate authorization or poorly deployed systems which result in directory listing and sensitive data disclosure.

A recent trend in such a vulnerability, are exposed AWS S3 buckets which are misconfigured, resulting in publicly exposed database back up files, internal files, configuration files and other private information being left available on the public Internet.

7% REMOTE CODE EXECUTION

Remote code execution (RCE) is used to describe an attacker's ability to execute arbitrary commands or code remotely across the Internet or network on a target machine.

This is achieved by exploiting a vulnerability which generally, if known about, could be mitigated via a patch or configuration change.

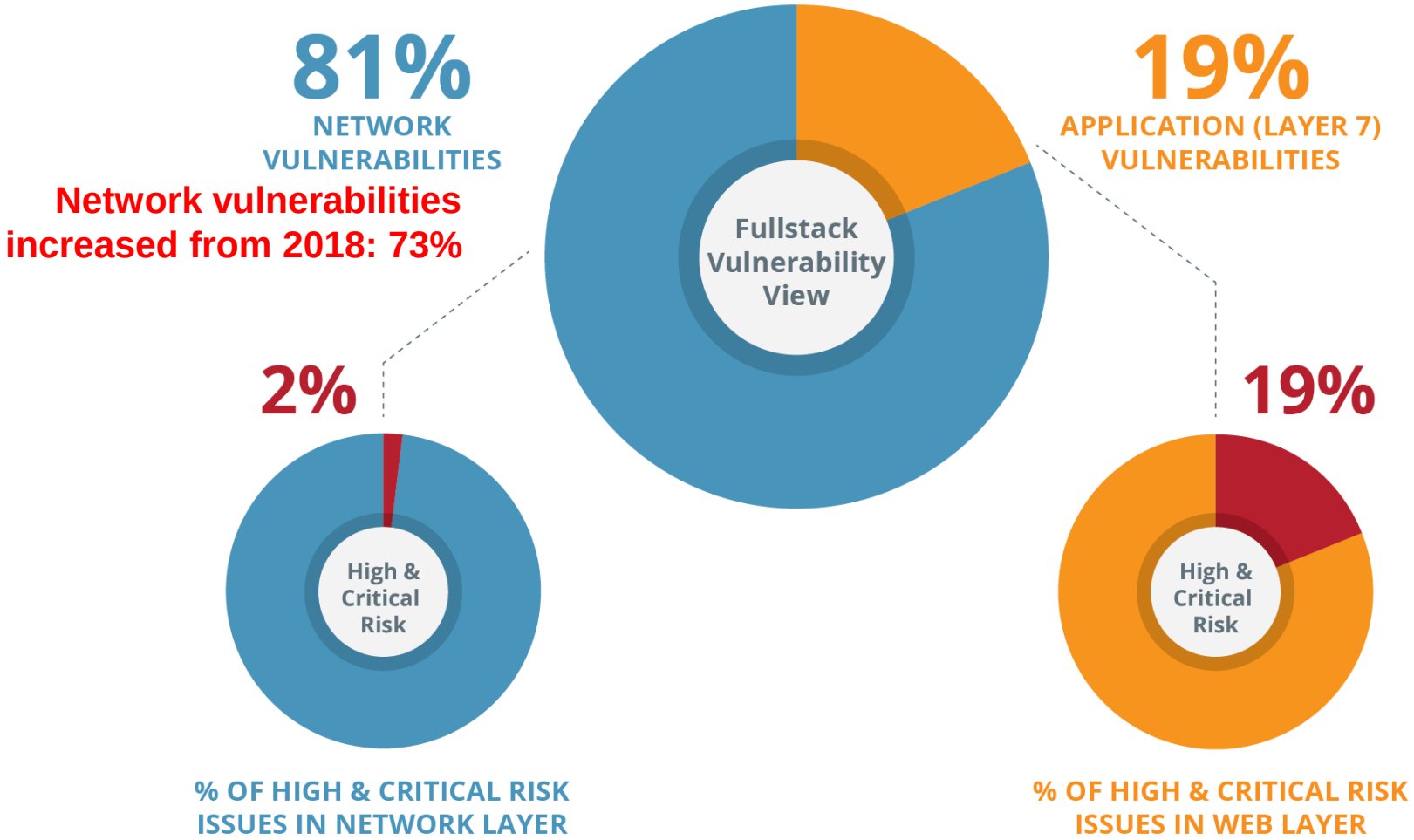
16% PHP MULTIPLE VULNERABILITIES

Many PHP vulnerabilities were discovered with ratings including both high and critical risk. Many PHP deployments have multiple vulnerabilities concurrently. PHP is still a widely used programming language but loosing popularity. Millions of sites on the Internet use PHP and will for some time to come.

CVE-2016-7411, CVE-2016-7412, CVE-2014-9425, CVE-2014-9709, CVE-2015-1351, CVE-2015-1352, CVE-2015-8383, CVE-2015-8386, CVE-2015-8387, CVE-2015-8389, CVE-2015-8390, CVE-2015-8391, CVE-2015-8393, CVE-2015-8394, CVE-2015-8865, CVE-2016-3141, CVE-2016-3142, CVE-2016-4070, CVE-2016-4071, CVE-2016-4072, CVE-2016-4073, CVE-2016-4537, CVE-2016-4539, CVE-2016-4540, CVE-2016-4542, CVE-2016-5385, CVE-2016-5399, CVE-2016-6207, CVE-2016-6289, CVE-2016-6290, CVE-2016-6291, CVE-2016-6292, CVE-2016-6293, CVE-2016-6294, CVE-2016-6295, CVE-2016-6296, CVE-2016-6297, CVE-2016-7124, CVE-2016-7125, CVE-2016-7126, CVE-2016-7127, CVE-2016-7128, CVE-2016-7129, CVE-2016-7130, CVE-2016-7131, CVE-2016-7132

- Source: Edgescan 2020 Vulnerability Statistics Report
- https://info.edgescan.com/vulnerability-stats

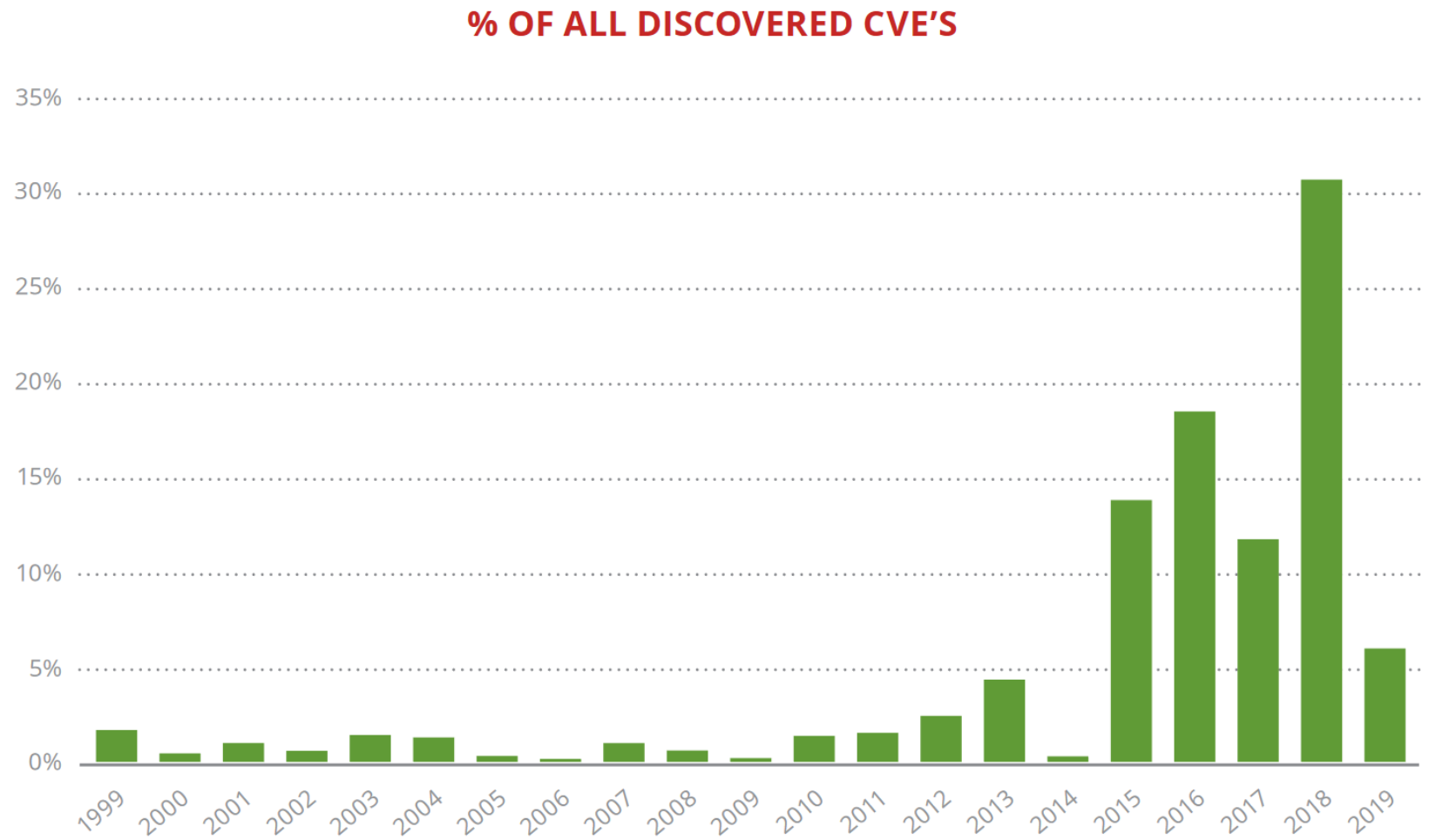
Statistics



Network vulnerabilities increased from 2018: 73%

- Source: Edgescan 2019 Vulnerability Statistics Report
- https://www.edgescan.com/?smd_process_download=1&download_id=2057

Statistics



- Security problems may linger…
- Source: Edgescan 2020 Vulnerability Statistics Report

Example

Dear Customer,

This message is to inform you that your account has been temporarily locked until you can validate your account details.

This is security measure to protect your iCloud Account from unapproved use.
We apologize for the inconvenience.

You won't be able to access Apple service or the iTunes & App Store until you verify your Apple Account ownership, we urge you to complete validation as soon as possible. Failure to validate your details within a 24 hours can result in termination of your Apple/iCloud Account to safeguard our system.

How do I validate my Apple Account and unlock my Apple ID?
Just proceed to the link below to verify your account. Login using your Apple Account and password, then follow the prompts.

Verify Your Account

https://vk.com/away.php?to=http://x.co/6nhA4

While using Apple devices and web services, you'll still login with your e-mail address as your Apple login.

If you have question and need support, please see the Apple ID Support Site.

Apple Support

- Note the actual link on hovering the mouse
  - This might be a HTML-Mail/PDF phishing mail, but where does it actually lead to? Is it secure (https)? “vk.com” is very trustworthy!
    - vk.com? httpS? Click on it, as it leads to vk.com?

What can go wrong?

- Denial of Service (DoS): your webshop is not accessible
  - → Direct losses
  - Political party/company website down: reputation loss etc.
- Defacement: the content of the website is changed
  - Shop: price modifications for you/all customers → Expensive!
  - Reputation loss, shutdown by government (e.g. illegal content), added to blacklists etc.
- Data loss: data from you (or your customers/employees/…!) is stolen
  - This is usually no immediate problem – you still have it
  - But the consequences can be dire: trade secrets lost, fines/compensations to pay, bad reputation/customer loss…
  - You might have to make it public and/or have to pay for credit monitoring for **all** affected persons…
    - Data breach notification duty
    - Dependent on business sector: critical infrastructure has notification obligations too now

What can go wrong?

- Extortion: ransomware; threat of other attacks, e.g. DoS
- Service stealing: whatever service/data you provide is used for free
  - Especially problematic if you are providing digital products
    - Which can then be distributed by P2P, sharehosters…
- "Piggybacking": using your resources, e.g. to send spam, host own data, phishing, infect visitors…
  - Liability: damages for affected persons
  - Increased costs: data transfer/CPU use (cloud services!)
  - Lower performance: something else is going on too
  - Blacklists: customers/business partners won't receive any E-Mails you send
  - Legal problems: the police might shut you down/confiscate equipment (only for a few years!) until you prove your innocence
  - Etc!

Where to attack?

- Operating System: not covered here
  - (Remote) attacks are rare and difficult
    - But see Spectre and Meltdown; Heartbleed (=web server)!
- Transport: HTTP / TLS sniffing
  - Extremely difficult if not on transmission path
  - Trivial if on the path & no encryption (intelligence agencies!)
  - Complicated to do unnoticed if on the path & encryption
    - Certificate warnings/pinning
- Web Server: the server itself and any necessary applications or languages
  - PHP, Python, Ruby…; server plugins/modules
- Web application platform: basic frameworks used by the application
  - Spring, JSF, Ruby on Rails, Struts, Tapestry, Cold Spring, Node.js… (WordPress, Drupal, Typo3…)
- Cloud assets, e.g. access tokens for backend services

Where to attack?

- Database: typically only an indirect target
  - Note: every year a significant number of DBs are found publicly exposed to the Internet... → This should never be necessary!
  - PostgreSQL, MySQL, MS SQL Server, DB2, Oracle; any non-relational ones (e.g. recently Redis)
    - MongoDB: 2015 more than 40.000 were completely open on the Internet (no security → read&write access without authentication)
      - Reason: Setup guide did not mention to activate access control, authentication, and transfer encryption; all were disabled by default
- **Web application: server-side → See later!**
- **Web Client: client-side → See later!**
- Availability: sending enormous amount or size (amplification attacks) of requests (few variations)
  - Any kind of packets (network-level attack; simple to filter out) or fully legitimate connections (application-level attack; very difficult/resource intensive)

How to attack: Profiling

- Gathering information on potential vulnerabilities (or excluding non-working ones)
  - Basic information: what OS, web server, application framework, language, load balancers, local time&timezone, proxies, web application firewall, services running…
  - User information: who owns it? Names, E-Mail addressed, IP addresses (web server, DNS, internal servers etc)
  - Website information: complete mirror of pages, known accounts, static/dynamic pages, form pages, directory layout, presence of common files, source code (accessible, OS repositories…) etc.
  - Vulnerability information:
    - Common profiles for applications/frameworks
    - Automated scanners for testing known ones
    - Manually looking for potential problems
  - Gathering data on vulnerabilities: how to exploit it, working code, assembling payload, preparing server for further code/control…

How to attack: Executing the attack

- When will the system be most likely un-/less supervised?
- Exploiting vulnerability or testing for any probable one
  - Or just testing anything, perhaps we are lucky!
  - Repeating the tests – some are not deterministic
- Hiding traces of the attack while in progress (logs)
- Hiding the source of the attack (IP)
- Injecting first "foothold": typically some root/admin shell
  - Almost always fragile: only in memory, very small, almost no functionality, not accessible from outside (only: in→ out)
- Expanding the foothold: connecting out, loading more code, privilege escalation, installing permanently…
  - Has often to be done blindly, e.g. by a pre-defined script!
- Gaining access: installing backdoor/command receiver and testing it

Hiding: Various traces (1)

- Source (IP): use another computer as proxy
  - Commercial, hacked, Tor network…
  - Or third parties, e.g. through SPAM
    - Let them try; if the attack is successful, the hacked computer will "phone home" – but to you, not to the third party!
- Source: don't mix legitimate and "attack" traffic
  - Log in with your real account and later trying to hack → Bad idea!
  - Use different IP addresses and do not reuse connection data (e.g. session IDs!)
  - Different systems and different times
  - Use different credentials
- Evading IDS (Intrusion Detection Systems)
  - E.g. inserting packets into a stream which are physically addressed to IDS (MAC address only; IP is for attacking connection!): Sees different data stream than recipient
    - Especially IP&TCP layers allow numerous "errors" to confuse listeners

Hiding: Various traces (2)

- Progress: rare tries (“Stealth”)
  - Not a single barrage of requests, but one every few hours split over several days – The server is going nowhere (and updates are rare and take a long time to be implemented)!
- Progress: very frequent tries (“Overwhelm”)
  - Try to fill up the log and crash the computer after the attack
- Progress: log evasion (staying out of the log files)
  - Long URLs: logs may limit entry size (to avoid DoS attacks!)
    - So add harmless in front of "real" params (depends on server used)!
    - E.g. http://www.victim.com/sh_prod.asp?uid=<4096 random chars> &uname=**' or 1=1; --**
  - Encoding: encode everything as URLEncode
    - Makes it harder to see the attack unless explicitly looking
    - E.g. uname%3D'%20or%201%3D1%3B%20--%20
    - Or: %75%6e%61%6d%64%3d%27%20%6f%72%20%31%3d%31%3b%20%2d%2d%20

How to attack: Exploiting

- Install some malware: typically a hidden "remote control" application
- Depending on the intention, various avenues are open:
  - Political/personal gain: Deface website, download for free etc
    - Today rare!
  - "Terrorist“/”State actor”: delete data, crash system
    - Very rare, except if you are an “interesting” target
  - Espionage: steal specific confidential information
  - **Commercial crime: steal any data perhaps worth something**
    - Credit card/identity numbers, account credentials, E-Mail addresses
    - Introduce slight modifications into data: bank account for payments
    - Insert adware to target user of system/remote users (server)
    - Encrypt data and request payment for decryption key
- Often the actual user of the illegal access is someone else
  - "Renting" computers (botnets)
  - Selling raw data for exploitation by others
  - Selling the software itself (without any hacked computers!)

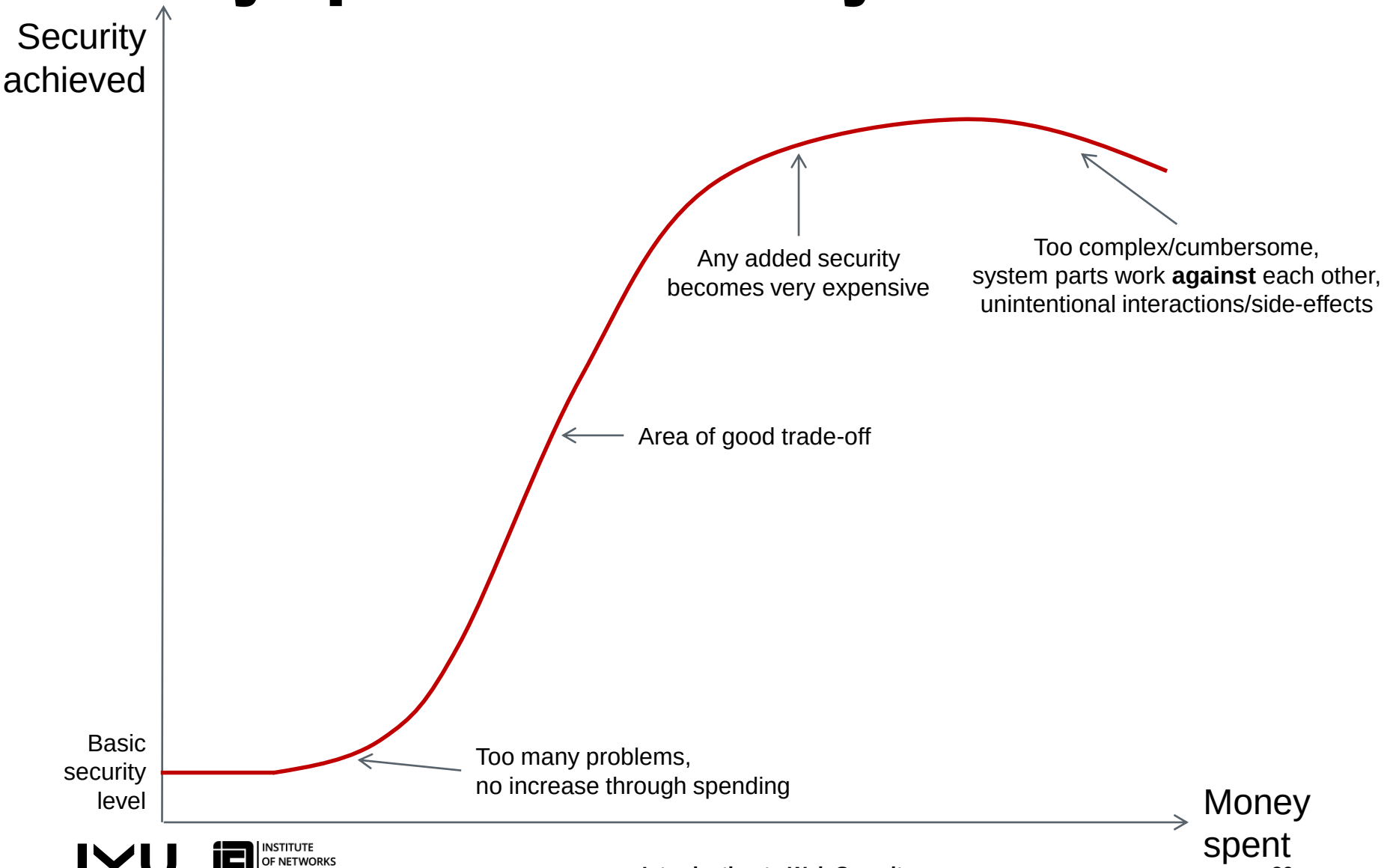
Who is responsible for web security?

- More persons than you probably thought!
  - Developer: writing the application so it is secure
    - Or at least: can be configured/used in a secure way
  - Webserver operator: don't add insecurity through configuration
    - Make sure web server, framework, and application are installed securely and up to date
  - Network operator(s): prevent attacks on infrastructure
    - E.g. DNS attacks are very dangerous!
  - End user: use up-to-date clients as well as common sense
- Why? Whole server + Transport + Client must be secure
  All of them simultaneously!
  - Server insecure: others can modify data on it…
  - Transport insecure: eavesdropping, MitM…
  - Client: some attacks can only be prevented by the client, like phishing!

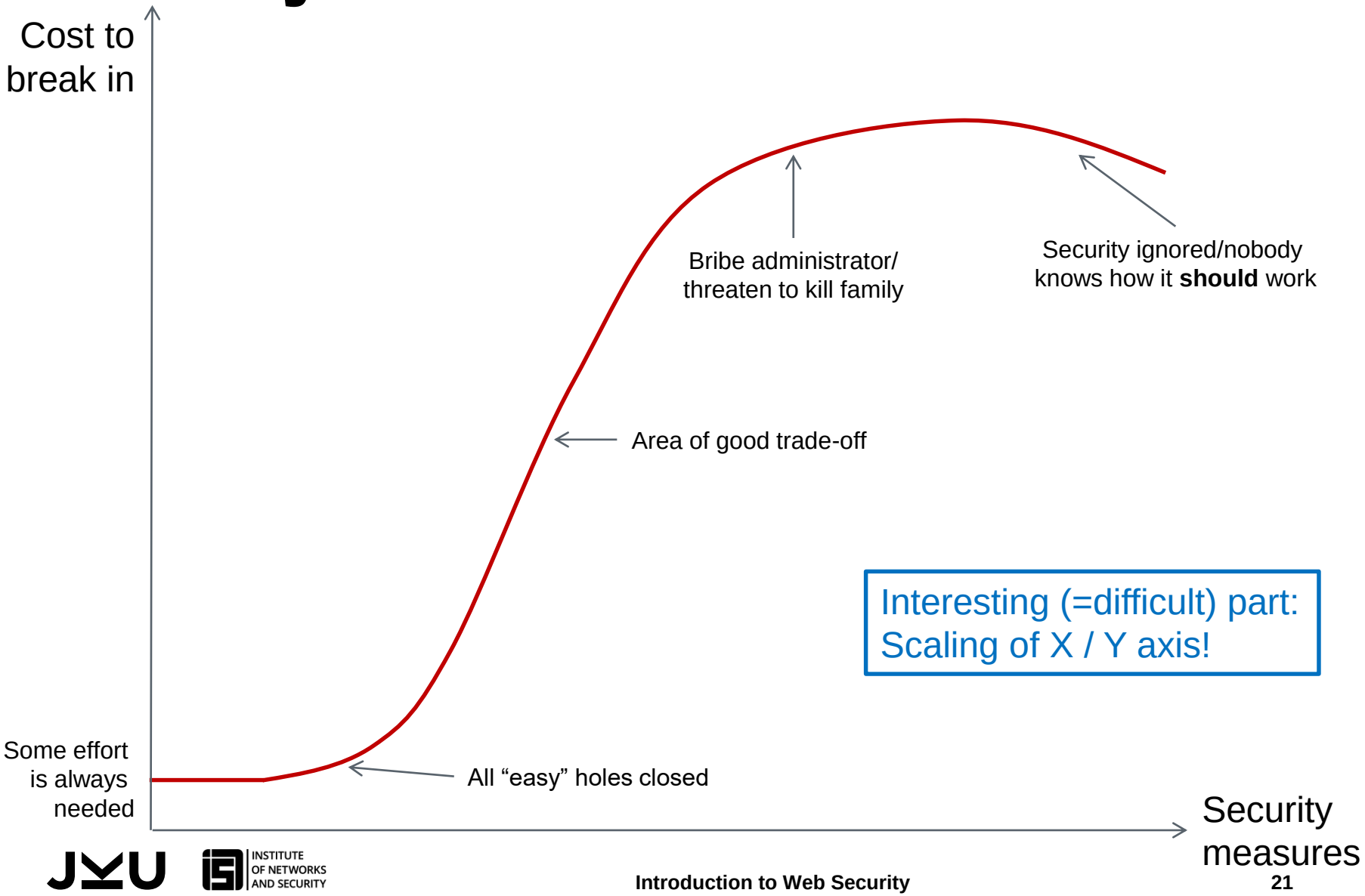
What do you have to do?

- Generally for security and specifically for web security:
  - Authenticate users: is the person really who she claims to be?
  - Authorize users: restrict users to what they are allowed to do
    - Accessing/modifying/deleting data (files, webpages, DB content…)
    - Check content: even if from you, you shouldn't distribute infected webpages/files
  - Prevent eavesdropping: nobody else should be able to access information being transmitted (stored → authorization!)
  - Ensure availability: DoS, resource exhaustion/overload etc
    - Including technical problems, force majeure…
  - Tracing: ensure that enough logging exists to be able to identify the source of attacks or any undesirable behaviour of the system (might also be legally required)
- Practical 80/20 rule: protect 20% of system which are high-impact and high-risk areas first
  → Get rid of 80% of all incidents!

Money spent vs. Security achieved



Security measures vs. Cost to break in



How to do it: Web applicat. guidelines

- Build security in from the start ("good enough security")
  - Especially add "hooks" to improve/add security later!
  - Investigate: what are the main assets to protect and who are the potential/probable attackers?
- Test security: not only functional, but also security tests
  - Input which is deliberately wrong or strange; don't bank on purely random/ape/fuzzing tests to find it!
    - „One-off“, explicitly trying common attacks like SQL injection, direct access without login, direct object access etc
- Keep it simple and centralized
  - E.g. one point where every request must pass through
  - Give out only as much information as necessary (especially error messages; but: local logs)
- “Externalize” security
  - Not distributed everywhere, but single “security configuration” file

How to do it: Web applicat. guidelines

- Store all data unescaped and raw and escape all data when creating output (according to location) or using it (e.g. as commands, DB query content)
  - Because you don't know where it will end up, so you can't appropriately escape input!
  - **Note: another school of thought exists!**
    - Escape immediately on input and store escaped value
    - Unescape (and encode differently) if absolutely necessary
    - Which approach is better is unclear/depends on priorities/mindset/… and whether there is “one output” with very frequent exceptions or a variety of uses/locations for data
- Don't do it yourself: cryptography, authentication etc
  - Should be part of framework used → use it securely
    - And actually do use it & make sure you use it all the time/everywhere!
    - Make sure to configure it correctly to exclude the old/weak ciphers/modes/… (which are there for compatibility)

Classes/Types of attacks

- Very coarse classification:
  - Information leakage: not an actual attack, but profiling for one
    - Becomes real attack in combination with trusting client/input validation
  - Attacks against cryptography: typically not breaking, but circumventing it (get key, disable encryption etc)
    - "Adding TLS" is not going to help one bit if the key is static!
  - Incorrect code: forgetting security or implementing it wrong
    - Note: code is perfectly working (=functionality) & might even be tested!
    - Good algorithm + bad implementation vs. correct impl. of bad algorithm
  - Trusting the client: protecting clients from themselves
    - Plus input validation (see below!)
  - Input validation problems: the trusted user sends data from his/her client computer…
    - … but it turns out to be not that harmless at all (or not that user ☺)!
    - Can be anything: data, programs/scripts, commands (shell, DB,…) etc
    - This is the main and most important type!

OWASP Web Security Reports

- Lists every few years what the most common and/or dangerous attacks
- Based on reports from businesses and academia
- Content/Ranking: occurrence, exploitability, detectability, and impact
- What is common: many attack types are
  - very old
  - very well known how they work
  - very well known countermeasures exist
- So: It is **very** rarely the surprising new attack, but rather a “new implementation” of a boring old problem!
  - Similar attacks in new software, introduced in old software through changes or patches
  - “Teaching” the “developers” (=programmer, but also CSS designer!) is very important

Immune against these common attacks → You can sleep peacefully!

Web Security Report 2010

OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross-Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross-Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A8 – Insecure Cryptographic Storage	A7 – Insecure Cryptographic Storage
A10 – Failure to Restrict URL Access	A8 – Failure to Restrict URL Access
A9 – Insecure Communications	A9 – Insufficient Transport Layer Protection
<not in T10 2007>	A10 – Unvalidated Redirects and Forwards (NEW)
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

Source: OWASP

Web Security Report 2013

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

Source: OWASP

Web Security Report 2017

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Source: OWASP

BSI IT Grundschutz 2021

- IT-Grundschutz by BSI introduces in its 2021 version concepts/ requirements for the development of web applications: CON.10
- General dangers:
  - Circumventing authorization
  - Insufficient input validation and output escaping
  - Missing or incorrect handling of errors
  - Insufficient logging of security-relevant events
  - Disclosure of security-relevant information
  - Misuse of a web application through automated clients
  - Insufficient session management
- Requirements for developers, who are the ones responsible for securing against these dangers!) are split in three groups
  - Basic: Must-have for every web application
  - Standard: This is state of the art and everyone should have this
  - Advanced: If increased security is needed

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic - Authentication

- Secure and appropriate authentication of users MUST be performed before access to protected data or functions
  - An appropriate method MUST be selected and the selection process MUST be documented
- Authentication MUST be performed by a central component
  - This SHOULD be based on an established standard component, like frameworks or libraries
- If authentication data is stored on a client, the user MUST be informed about the risks (→opt-in)
- The application MUST allow defining limits for the number of failed authentication tries
- The application MUST inform users immediately when their password was reset

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Access control

- An authorisation component MUST be included, so users can perform only functions they are allowed to
- Any access to protected content and functions MUST be verified before being provided/executed
- Authorisation MUST cover every and all resource and content of the web application
- If the authorization component fails, access MUST be denied
- Access control for URLs and object references MUST be present
  - This probably means REST calls, directly entering an URL (e.g. via a bookmark) etc

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Secure session management

- Session-IDs MUST be appropriately secured
- Session-IDs MUST be created randomly with sufficient entropy
  - If the framework supports this, this function MUST be used
  - Security-relevant configuration options of the framework MUST be considered
- Session-IDs MUST be sufficiently protected, if they are transmitted or stored by clients
  - This applies e.g. to all session IDs in Cookies → TLS
- Users MUST be able to explicitly terminate a session
  - Can be problematic → Closing the browser is the “minimum”
- After authentication, an existing session ID MUST be replaced by a new one
- Sessions MUST have a maximum time of validity (timeout)
  - Inactive sessions MUST be invalidated after some time
  - After session expiry, all sessions data MUST be invalid and deleted

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Including content

- Developers MUST ensure that only data/content is included and delivered to users that is intended for them
- Redirections MUST be restricted so users are only sent to trustworthy sites
- If users leaves the trust domain, they MUST be informed about this
  - A bit unclear: redirection to other servers of the same company are OK, but to external ones should show some previous information (hint at link → redirection?), interstitional…

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Upload functionality

- It MUST be ensured that uploaded files can only be stored in the predetermined location
- End-user influence on the location MUST be prevented
- The operator MUST be able to configure uploads later
  - This probably applies to both the location as well as further restrictions, e.g. file size and/or file type

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Protection from autom. use

- Protections against automated access MUST be implemented
- When designing these security mechanisms their impact on usage options of authorized users MUST be considered
- Note: This is quite unclear
  - Every access via browser is “automated use”
  - It may be very difficult to distinguish manual from programmatic access
  - Many users will want to automate access sometimes
  - The best possibility is probably some kind of rate-limiting

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Protecting confidential data

- Confidential data may ONLY be sent via Post from client to server
- No confidential data may be stored on the client; this MUST be ensured through directives
  - No caching headers
- Forms may NEVER show confidential data in cleartext
  - Passwords must be in password fields, not input fields
- The application SHOULD ensure, that confidential data is not stored by the browser unexpectedly
  - You can't really do this → Browsers may always store passwords, but will not do this without asking the user
- All access data MUST be protected on the server by secure cryptographic algorithms (salted hash)
- The source code of the application MUST be protected against unauthorized access

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Input validation & escaping

- Developers MUST handle all data sent to the web application as potentially dangerous
  - All such data MUST be suitably filtered
  - All input data (session IDs, form input, uploads, third-party data etc) MUST be validated on the server
- Incorrect data SHOULD not be corrected automatically (sanitized)
  - If unavoidably, sanitizing MUST be implemented securely
- Output data MUST be encoded in a way that malicious code will not be interpreted or executed on the target system

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Protect. against SQL injection

- If a database is used, Stored Procedures or Prepared SQL Statements MUST be used
- If these are not available, other protection measures MUST be used to secure the queries
  - This probably (=should) applies to NoSQL… database too
    → use whatever is their equivalent

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Basic – Limited disclosure of security relevant information

- It MUST be ensured, that no information is disclosed that will provide attackers with hints how to circumvent security measures by
  - Web pages
  - Replies
  - Error messages
- Note that we don’t get any information here, what exactly that data would be. The developer will have to find out!
  - Generally → as little as possible

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Standard – Software architecture

- The developers SHOULD document the architecture of the application, including all components and dependencies
  - I.e. all libraries and other necessary software (recursively!)
  - It SHOULD be created and update already during development
  - It should be designed to be used during the development and so decisions can be evaluated
    - Documentation of the process that led to important decisions
- The documentation SHOULD mark all elements necessary for the use, which are not part of the web application
  - E.g. the database → necessary, but not part of the application itself (except e.g. an in-process DB)
- The documentation should describe
  - Which components implement which security measures
  - How the application is integrated into an existing infrastructure
  - Which cryptographic functions/modes/algorithms/… are used

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Standard – Verifying essential changes

- It SHOULD be ensured, that changes to important configuration options require user verification by entering the password again
  - If this is impossible, another suitable method SHOULD be used to authenticate the user
- Users SHOULD be informed about important configuration changes through communication outside of the web application
  - E.g. E-Mail, SMS

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Standard – Error handling

- Errors while a web application runs SHOULD be handled in a way so that the application remains in a consistent state
- Errors SHOULD be logged
- If an action causes an error, this action SHOULD be terminated
- In case of errors access to a resource requested SHOULD be denied
- Previously reserved resources SHOULD be freed during error handling
- Errors SHOULD be handled by the web application itself if possible

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Standard - Secure HTTPS configuration

- Appropriate HTTP response headers SHOULD be set, at least
  - Content-Security-Policy → restrict what the page can do
  - Strict-Transport-Security → always use httpS
  - Content-Type → what are we sending
  - X-Content-Type-Options → prevent content-type guessing
  - Cache-Control → determine whether/how long to cache
- The headers used SHOULD be set according to the application
  - Note: What else? E.g. setting a header with an invalid value is not going to be very helpful…
- The headers SHOULD be set as restrictive as possible
  - = least privilege
- Cookies SHOULD be marked as Secure, SameSite, and HTTPOnly

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Standard – Preventing CSRF

- Developers SHOULD implement security mechanisms to distinguish intentional requests by users from unintentionally forwarded commands of third persons
  - They SHOULD at least verify whether a secret token should be used in addition to the session ID

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

Standard – Multi Factor Authentication

- Multi factor Authentication SHOULD be implemented

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html

BSI IT Grundschutz 2021 - Advanced

- Only two additional requirements, which are rather weak
- Preventing resource exhaustion
  - To protect against DoS resource-intensive operations SHOULD be avoided
  - If they are necessary, they SHOULD be especially secured
  - Web applications SHOULD monitor and prevent possible overflow of log data/logs
- Cryptographic protection of confidential data
  - Secure cryptographic algorithms SHOULD be employed to protect confidential data
    - What exactly does that mean? Its up to you!
    - Using only secure algorithms and actual cryptography should (in my opinion) be a basic requirement…

Difficulties of prot. against attacks (1)

- You need to know about the attacks (at least classes/types) to be able to protect against them
- Many examples & newbie guides are still often "bad“ (but gets better)
  - And this is rarely even mentioned/corrected (e.g. SQL injection)
- Protect **everything every time against anyone → “zero trust”**
  - The attacker can choose, wait, and try again
- Problems are often not easily "localized“
  ="this is the one and only single erroneous statement"
  - Mostly such statements are correct, but should not be used at this locations && while this activity is going on && while the DB is in a certain state (emergence!)
- A defect might not be exploitable because of the system design, but on extension/modification/… it suddenly is
- You may not be able to fix it: defect in library
  - Might even be open source!

Difficulties of prot. against attacks (2)

- Vulnerabilities are often downplayed by the vendor
- Whose job is it (would it be) to fix it?
  - Browser vs. plugin, framework vs. OS vs. application, vendor 1 vs. vendor 2…
- Technological limitations:
  - Some procedures/functions just cannot be used securely
  - Protocols or standards might have flaws, but must still be implemented and/or used "as defined“/”as provided”
    - Because of various reasons, e.g. compatibility or legal
- Disclosure process varies:
  - 0-day attacks; sent to developer but ignored/delayed there; multiple mailing lists;…
- Marketing: "This system is secure!" → but against what in which circumstances?

Luckily…

This does NOT mean it is hopeless!

Summary

- Web applications **will** be attacked – sooner or later
  - Automated software kits used by "script kiddies“
  - Delay: some seconds to at most a few hours
- Getting it secure is very difficult, but "reasonable" security is not hard!
  - A bit of Floriani principle/not-in-my-backyard (“NIMBY”), however…
- Security must be built in from the beginning and needs at least some monitoring
- As user you cannot depend on the provider
  - Some things he cannot do, some things he won't
- Prepare for incidents or unpatched vulnerabilities
  - Modules for restricting access/filtering URLs etc. should be in there from the start
  - Backups, alternate versions (e.g. static copy of website) etc should be prepared

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Michael Sonntag
michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at