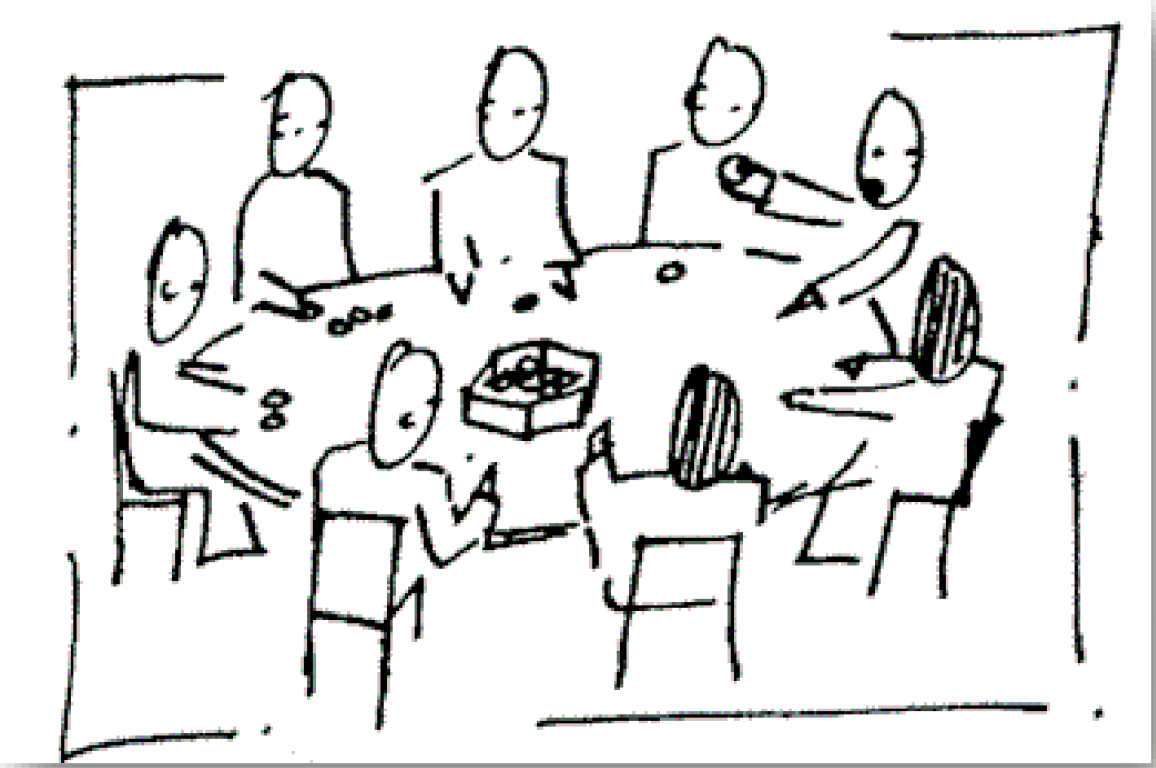
OWASP Stammtisch #37

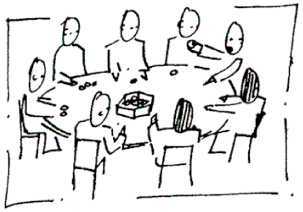
Frankfurt, 20.09.2017

Info

Intro



Intro



- My name is ...
- I work as ...
- I‘m here because ...

Agenda

1. Talk – Johannes: What is OWASP? (why am I here?)
2. Next meetup

OWASP
AppSec EU
Belfast
May 2017

OWASP
AppSec USA
ORLANDO
2017

Talk

OWASP
Open Web Application
Security Project

オワスプ
ジャパン

OWASP
Open Web Application
Security Project

OWASP
2016

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

What is OWASP?

www.owasp.org

Open Web Application Security Project

- worldwide free and open community focused on improving the security of application software
- Promotes secure software development
- Oriented to the delivery of web oriented services
- An open forum for discussion
- A free resource for any development team

What is OWASP?

- Non-profit (501c3), volunteer driven organization
  - All members are volunteers (save 4 employees)
  - All work is donated by volunteers and sponsors
- Provide free resources to the community
  - Publications, Articles, Standards
  - Testing and Training Software
  - Local Chapters & Mailing Lists
- Supported through sponsorships
  - Corporate support through financial or project sponsorship
  - Personal sponsorships from members

OWASP Organization

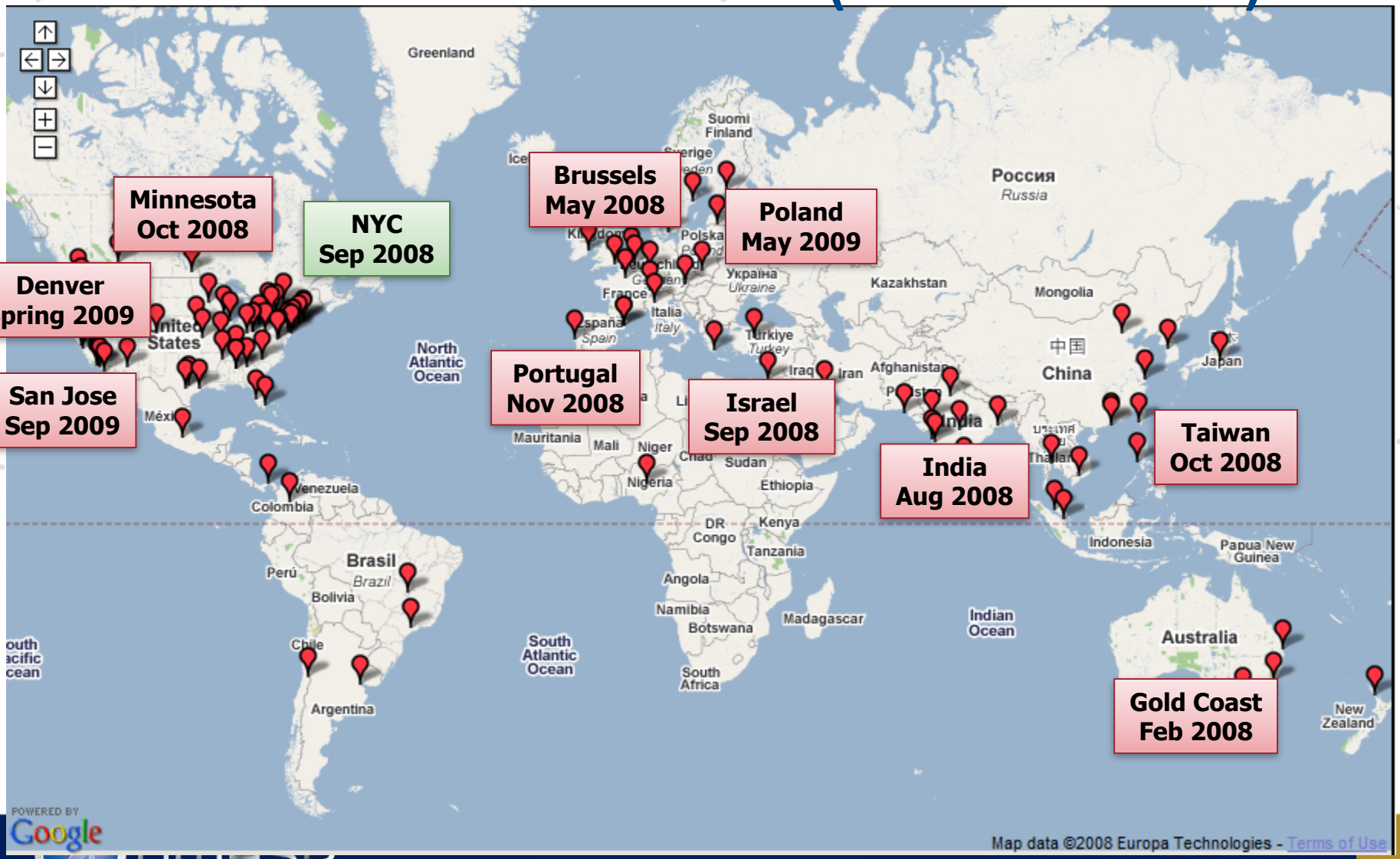
- Global Board
- Global Committees
  - Education
  - Chapters
  - Conferences
  - Industry
  - Projects & Tools
  - Membership
- Employees
- Volunteers

OWASP membership

Membership category	Annual membership fee
Individual Supporters	$50
Organization Supporters	$5,000
Accredited University Supporters	FREE (in exchange of meeting space at least 2x per year)
Lifetime Membership	$500

- Funds OWASP Speakers via OWASP On the Move
- Funds Season of Code projects
- Helps Support Local Chapters

OWASP Conferences (2008-2009)



Talk

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

Write Secure Code → Audit Code, Result → Control Risk

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

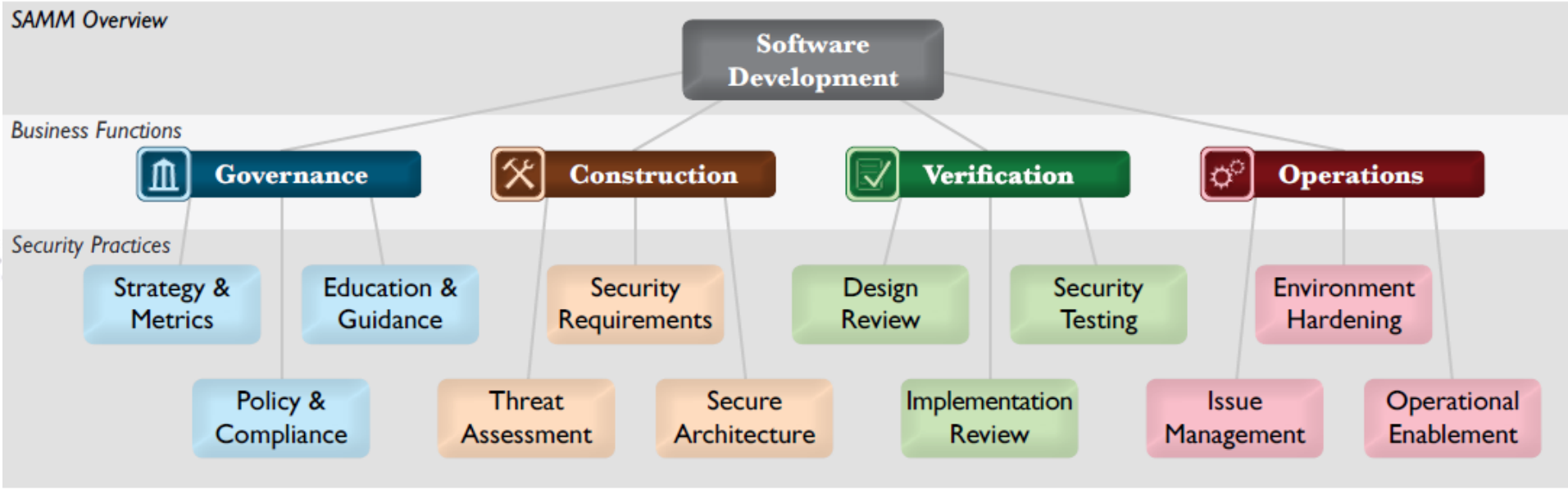
Talk

Write Secure Code → Audit Code, Result → Control Risk

- Software Assurance Maturity Model (SAMM)
- Mobile Application Security Verification Standard (MASVS)
- Cheat Sheet Series
- …

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

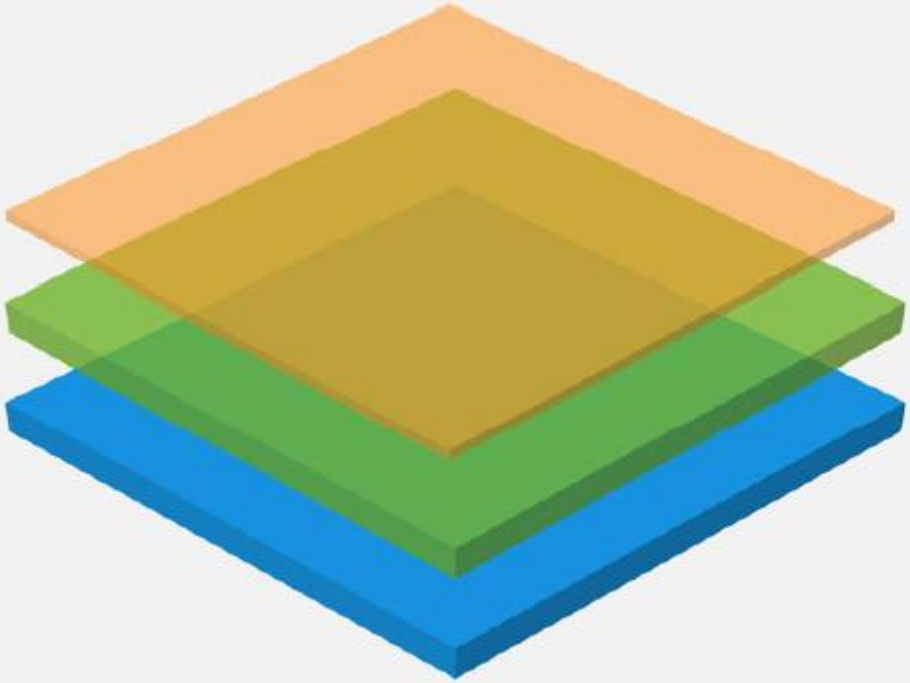
Talk

Write Secure Code → Audit Code, Result → Control Risk

- Software Assurance Maturity Model (SAMM)
- Mobile Application Security Verification Standard (MASVS)
- Cheat Sheet Series
- …

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



R – Resiliency Against Reverse Engineering and Tampering

L2 – Defense-in-Depth

L1 – Standard Security

#	Description	L1	L2
2.1	System credential storage facilities are used appropriately to store sensitive data, such as user credentials or cryptographic keys.	✓	✓
2.2	No sensitive data is written to application logs.	✓	✓
2.3	No sensitive data is shared with third parties unless it is a necessary part		

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

Write Secure Code → Audit Code, Result → Control Risk

- Software Assurance Maturity Model (SAMM)
- Mobile Application Security Verification Standard (MASVS)
- Cheat Sheet Series
- …

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

OWASP Cheat Sheets

V - T - E	Cheat Sheets [Collapse]
Developer / Builder	3rd Party Javascript Management · Access Control · AJAX Security Cheat Sheet · Authentication (ES) · Bean Validation Cheat Sheet · Choosing and Using Security Questions · Clickjacking Defense · C-Based Toolchain Hardening · Credential Stuffing Prevention Cheat Sheet · Cross-Site Request Forgery (CSRF) Prevention · Cryptographic Storage · Deserialization · DOM based XSS Prevention · Forgot Password · HTML5 Security · HTTP Strict Transport Security · Injection Prevention Cheat Sheet · Injection Prevention Cheat Sheet in Java · JSON Web Token (JWT) Cheat Sheet for Java · Input Validation · JAAS · LDAP Injection Prevention · Logging · Mass Assignment Cheat Sheet · .NET Security · OWASP Top Ten · Password Storage · Pinning · Query Parameterization · Ruby on Rails · Session Management · SAML Security · SQL Injection Prevention · Transaction Authorization · Transport Layer Protection · Unvalidated Redirects and Forwards · User Privacy Protection · Web Service Security · XSS (Cross Site Scripting) Prevention · XML External Entity (XXE) Prevention Cheat Sheet
Assessment / Breaker	Attack Surface Analysis · REST Assessment · Web Application Security Testing · XML Security Cheat Sheet · XSS Filter Evasion
Mobile	Android Testing · IOS Developer · Mobile Jailbreaking
OpSec / Defender	Virtual Patching
Draft and Beta	Application Security Architecture · Business Logic Security · Command Injection Defense Cheat Sheet · Content Security Policy · Denial of Service Cheat Sheet · Grails Secure Code Review · Insecure Direct Object Reference Prevention · IOS Application Security Testing · Key Management · PHP Security · REST Security · Regular Expression Security Cheatsheet · Secure Coding · Secure SDLC · Threat Modeling · Vulnerability Disclosure
All Pages In This Category	

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

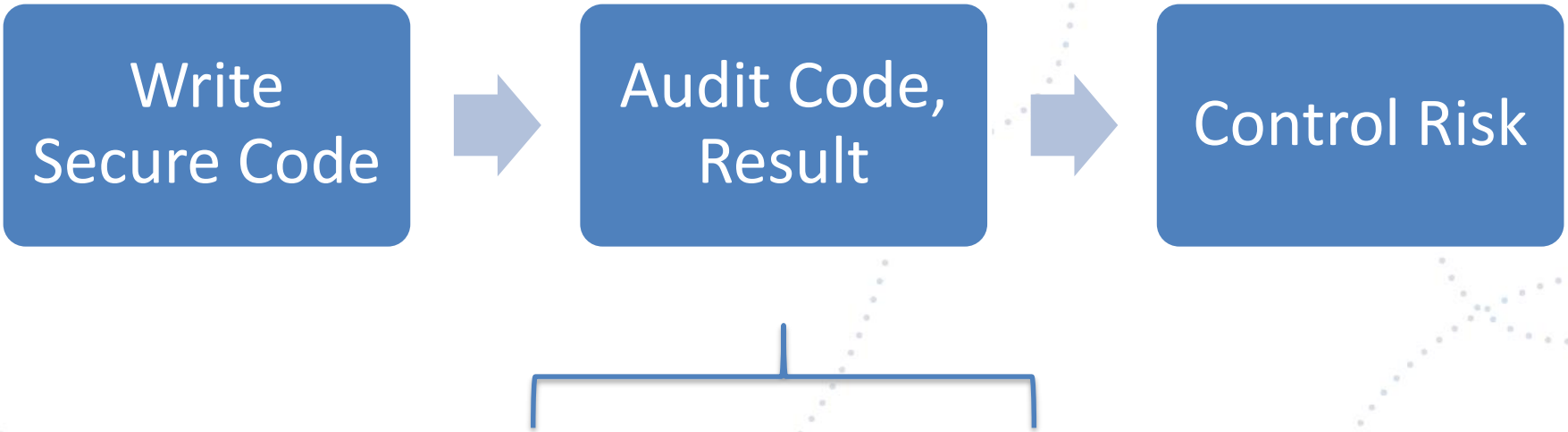
Safe Java Stored Procedure Example

The following code example uses a CallableStatement, Java's implementation of the stored procedure interface, to execute the same database query. The "sp_getAccountBalance" stored procedure would have to be predefined in the database and implement the same functionality as the query defined above.

```
String custname = request.getParameter("customerName"); // This should REALLY be validated
try {
        CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
        cs.setString(1, custname);
        ResultSet results = cs.executeQuery();
        // … result set handling
} catch (SQLException se) {
        // … logging and error handling
}
```


https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



- **OWASP Testing Guide**
- «OWASP Top10» Web, Mobile, IoT…
- OWASP Zed Attack Proxy
- …

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

Testing for Error Handling

Analysis of Error Codes (OTG-ERR-001)

Analysis of Stack Traces (OTG-ERR-002)

Testing for weak Cryptography

Testing for Weak SSL/TLS Ciphers, Insufficient Transp

Testing for Padding Oracle (OTG-CRYPST-002)

Testing for Sensitive information sent via unencrypte

Authorization Testing

Testing Directory traversal/file include (OTG-AUTHZ-001)

Testing for bypassing authorization schema (OTG-AUTHZ

Testing for Privilege Escalation (OTG-AUTHZ-003)

Testing for Insecure Direct Object References (OTG-AUTH

Session Management Testing

Testing for Bypassing Session Management Schema (OTG

Testing for Cookies attributes (OTG-SESS-002)

Identity Management Testing

Test Role Definitions (OTG-IDENT-001)

Test User Registration Process (OTG-IDENT-002)

Test Account Provisioning Process (OTG-IDENT-003)

Testing for Account Enumeration and Guessable User Account (OTG-IDENT-004)

Testing for Weak or unenforced username policy (OTG-IDENT-005)

Authentication Testing

Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)

Testing for default credentials (OTG-AUTHN-002)

Testing for Weak lock out mechanism (OTG-AUTHN-003)

Testing for bypassing authentication schema (OTG-AUTHN-004)

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-sm

Talk

Authorization Testing

Testing Directory traversal/file include (OTG-AUTHZ-001)

Testing Techniques

The next stage of testing is analyzing the input validation functions present in the web application. Using the previous example, the dynamic page called getUserProfile.jsp loads static information from a file and shows the content to users. An attacker could insert the malicious string "../../../../etc/passwd" to include the password hash file of a Linux/UNIX system. Obviously, this kind of attack is possible only if the validation checkpoint fails; according to the file system privileges, the web application itself must be able to read the file.

To successfully test for this flaw, the tester needs to have knowledge of the system being tested and the location of the files being requested. There is no point requesting /etc/passwd from an IIS web server.

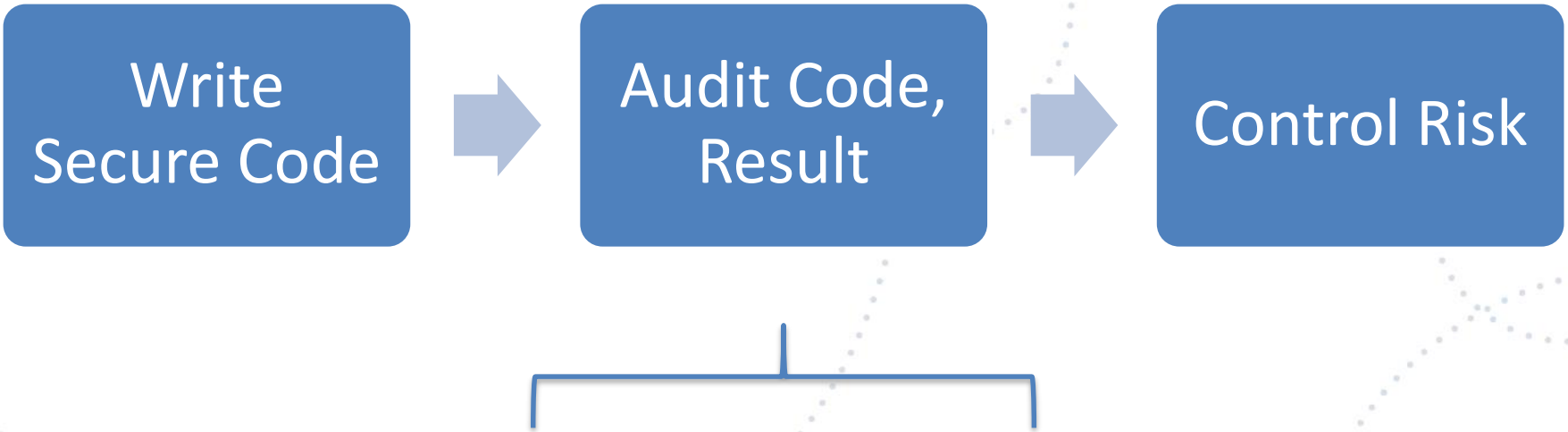
http://example.com/getUserProfile.jsp?item=../../../../etc/passwd

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



- OWASP Testing Guide
- **«OWASP Top10» Web, Mobile, IoT…**
- OWASP Zed Attack Proxy
- …

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

For 2017, the OWASP Top 10 Most Critical Web Application Security Risks (in the Release Candidate) are:

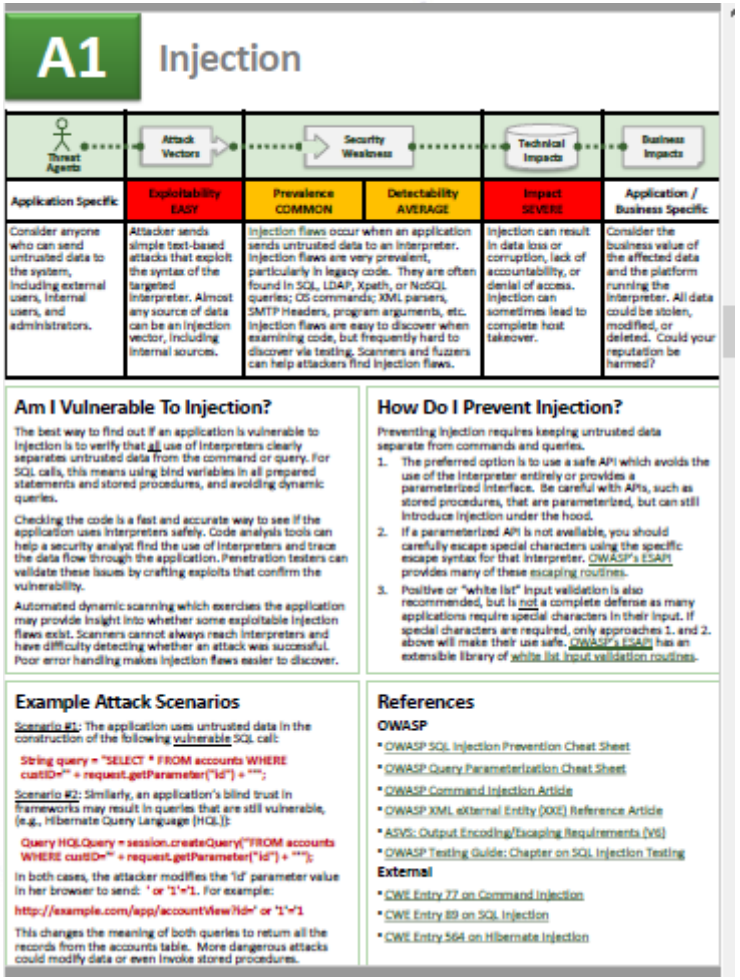
- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting (XSS)
- A4 Broken Access Control (As it was in 2004)
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Insufficient Attack Protection (NEW)
- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Underprotected APIs (NEW)

The OWASP Top 10 IoT Vulnerabilities from 2014 are as follows:

Rank	Title
I1	• Insecure Web Interface
I2	• Insufficient Authentication/Authorization
I3	• Insecure Network Services
I4	• Lack of Transport Encryption/Integrity Verification
I5	• Privacy Concerns
I6	• Insecure Cloud Interface
I7	• Insecure Mobile Interface
I8	• Insufficient Security Configurability
I9	• Insecure Software/Firmware
I10	• Poor Physical Security

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



A1 Injection

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability EASY	Prevalence COMMON	Detectability AVERAGE	Impact SEVERE	Application / Business Specific
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?

Am I Vulnerable To Injection?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover.

How Do I Prevent Injection?

Preventing injection requires keeping untrusted data separate from commands and queries.

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.
2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP's ESAPI provides many of these escaping routines.
3. Positive or "white list" input validation is also recommended, but is not a complete defense as many applications require special characters in their input. If special characters are required, only approaches 1. and 2. above will make their use safe. OWASP's ESAPI has an extensible library of white list input validation routines.

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:

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

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send: ' or '1'='1. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

References

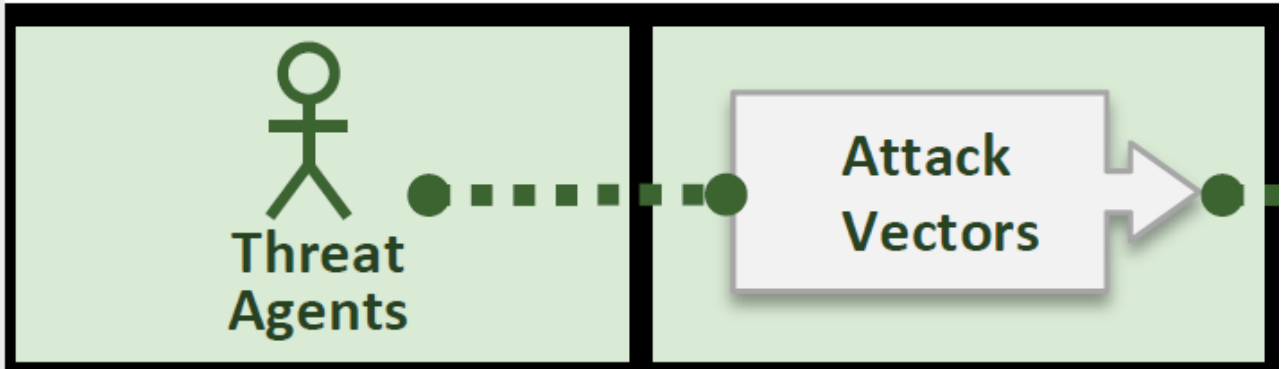
OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Query Parameterization Cheat Sheet
- OWASP Command Injection Article
- OWASP XML eXternal Entity (XXE) Reference Article
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP Testing Guide: Chapter on SQL Injection Testing

External

- CWE Entry 77 on Command Injection
- CWE Entry 89 on SQL Injection
- CWE Entry 564 on Hibernate Injection

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

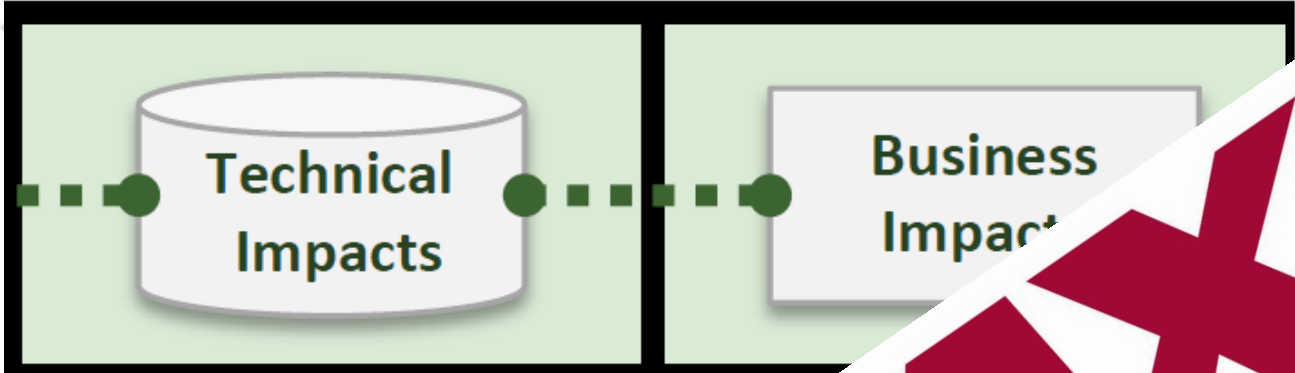


Application Specific	Exploitability EASY
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.

Prevalence	Detectability
COMMON	AVERAGE

Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.

Impact SEVERE	Application / Business Specific
Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?



Impact SEVERE

Injection can
in data los
corrup
acc
ad to
most
er.

value of
ected data
d the platform
running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?

EQUIFAX®

https://media.mnn.com/ass

Talk

Am I vulnerable to injection?
How do I prevent injection attacks?

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:

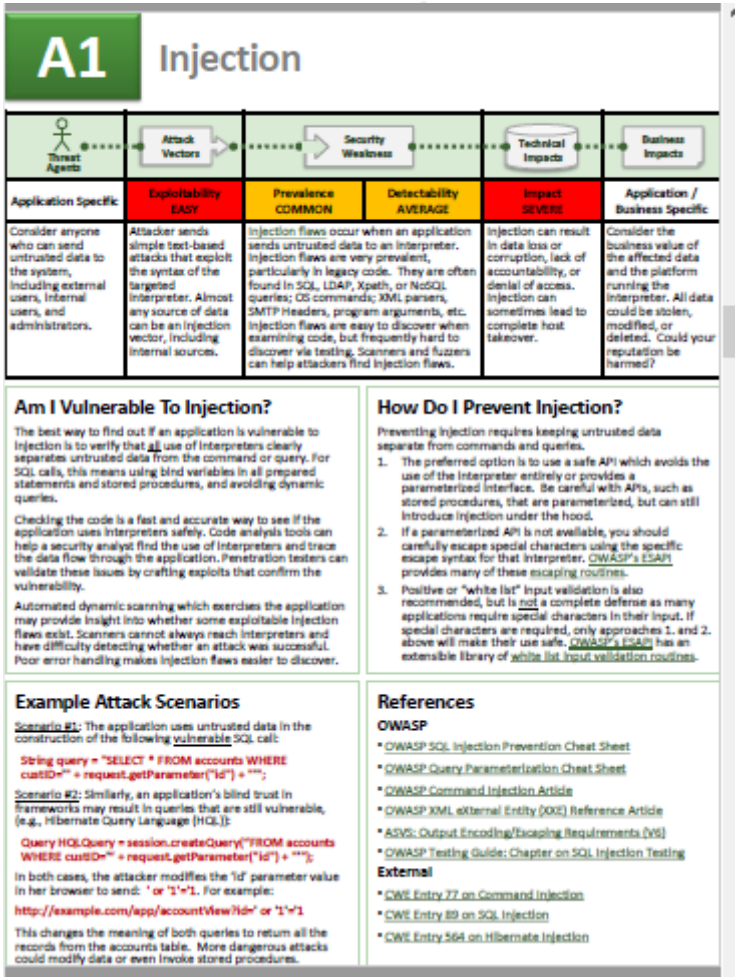
```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

References

OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Query Parameterization Cheat Sheet

Talk



A1 Injection

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability EASY	Prevalence COMMON	Detectability AVERAGE	Impact SEVERE	Application / Business Specific
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?

Am I Vulnerable To Injection?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover.

How Do I Prevent Injection?

Preventing injection requires keeping untrusted data separate from commands and queries.

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.
2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP's ESAPI provides many of these escaping routines.
3. Positive or "white list" input validation is also recommended, but is not a complete defense as many applications require special characters in their input. If special characters are required, only approaches 1. and 2. above will make their use safe. OWASP's ESAPI has an extensible library of white list input validation routines.

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:

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

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send: ' or '1'='1. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

References

OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Query Parameterization Cheat Sheet
- OWASP Command Injection Article
- OWASP XML eXternal Entity (XXE) Reference Article
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP Testing Guide: Chapter on SQL Injection Testing

External

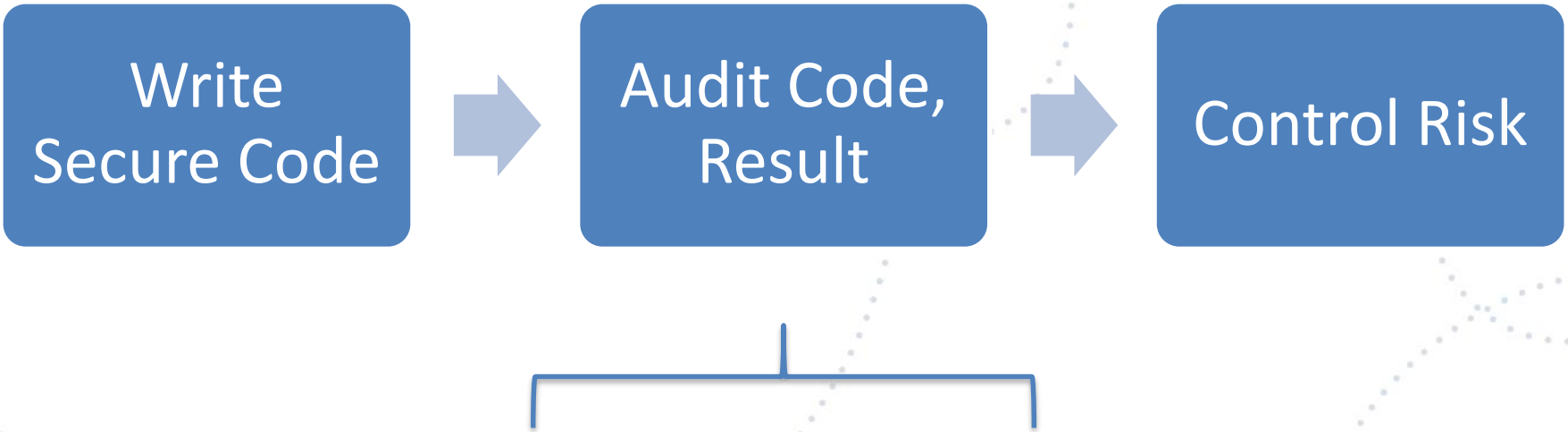
- CWE Entry 77 on Command Injection
- CWE Entry 89 on SQL Injection
- CWE Entry 564 on Hibernate Injection

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

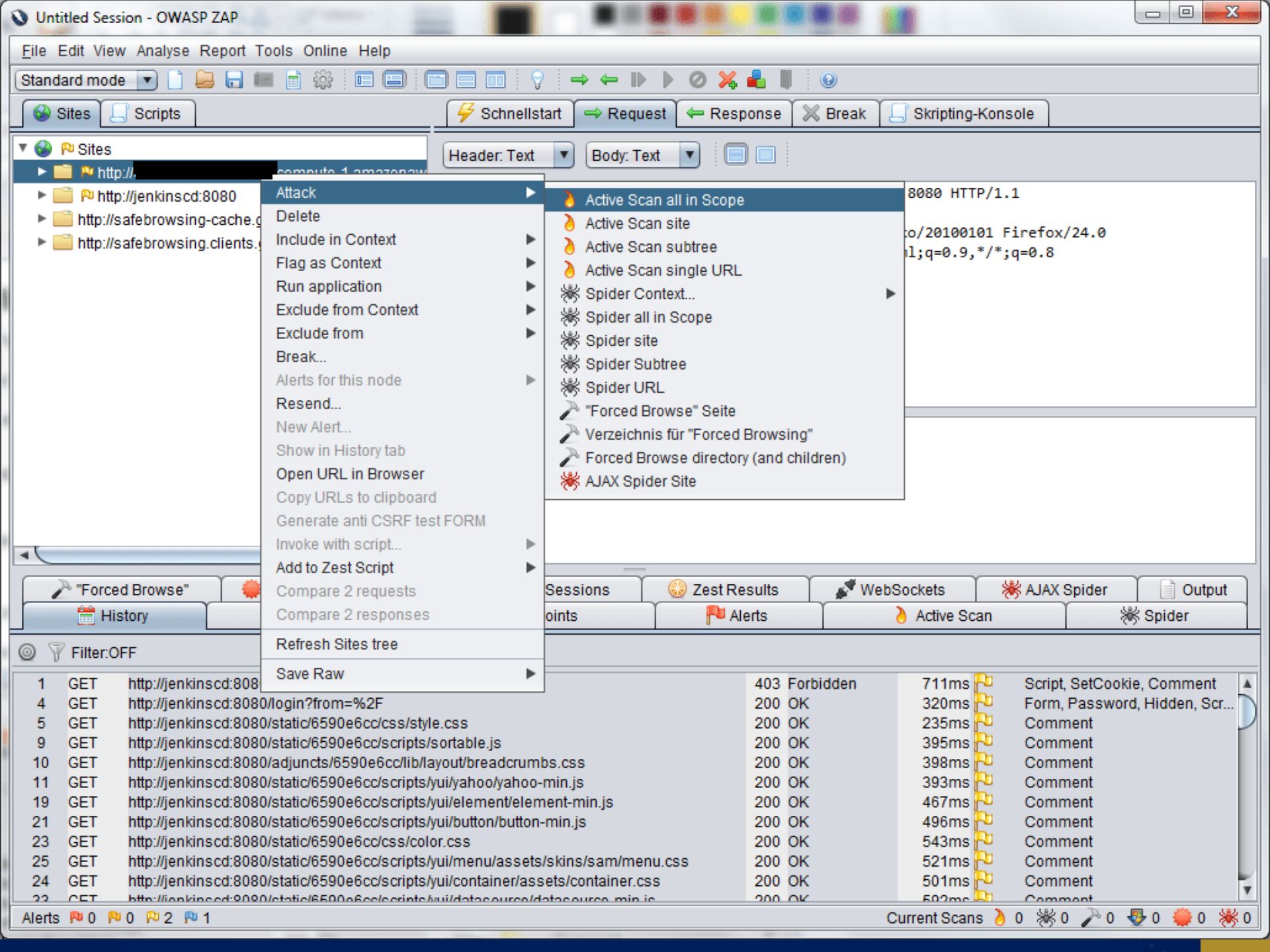
https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



- OWASP Testing Guide
- «OWASP Top10» Web, Mobile, IoT…
- **OWASP Zed Attack Proxy**
- …

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg


Untitled Session - OWASP ZAP
File Edit View Analyse Report Tools Online Help
Standard mode
Sites
Scripts
Schnellstart
Request
Response
Break
Skripting-Konsole
Header: Text
Body: Text
http://jenkinscd:8080
http://safebrowsing-cache.g
http://safebrowsing.clients.
Attack
Delete
Include in Context
Flag as Context
Run application
Exclude from Context
Exclude from
Break...
Alerts for this node
Resend...
New Alert...
Show in History tab
Open URL in Browser
Copy URLs to clipboard
Generate anti CSRF test FORM
Invoke with script...
Add to Zest Script
Compare 2 requests
Compare 2 responses
Refresh Sites tree
Save Raw
Active Scan all in Scope
Active Scan site
Active Scan subtree
Active Scan single URL
Spider Context...
Spider all in Scope
Spider site
Spider Subtree
Spider URL
"Forced Browse" Seite
Verzeichnis für "Forced Browsing"
Forced Browse directory (and children)
AJAX Spider Site
8080 HTTP/1.1
o/20100101 Firefox/24.0
l;q=0.9,*/*;q=0.8
"Forced Browse"
Sessions
Zest Results
WebSockets
AJAX Spider
Output
History
oints
Alerts
Active Scan
Spider
Filter:OFF
1 GET http://jenkinscd:808 403 Forbidden 711ms Script, SetCookie, Comment
4 GET http://jenkinscd:8080/login?from=%2F 200 OK 320ms Form, Password, Hidden, Scr...
5 GET http://jenkinscd:8080/static/6590e6cc/css/style.css 200 OK 235ms Comment
9 GET http://jenkinscd:8080/static/6590e6cc/scripts/sortable.js 200 OK 395ms Comment
10 GET http://jenkinscd:8080/adjuncts/6590e6cc/lib/layout/breadcrumbs.css 200 OK 398ms Comment
11 GET http://jenkinscd:8080/static/6590e6cc/scripts/yui/yahoo/yahoo-min.js 200 OK 393ms Comment
19 GET http://jenkinscd:8080/static/6590e6cc/scripts/yui/element/element-min.js 200 OK 467ms Comment
21 GET http://jenkinscd:8080/static/6590e6cc/scripts/yui/button/button-min.js 200 OK 496ms Comment
23 GET http://jenkinscd:8080/static/6590e6cc/css/color.css 200 OK 543ms Comment
25 GET http://jenkinscd:8080/static/6590e6cc/scripts/yui/menu/assets/skins/sam/menu.css 200 OK 521ms Comment
24 GET http://jenkinscd:8080/static/6590e6cc/scripts/yui/container/assets/container.css 200 OK 501ms Comment
Alerts 0 0 2 1
Current Scans 0 0 0 0 0 0

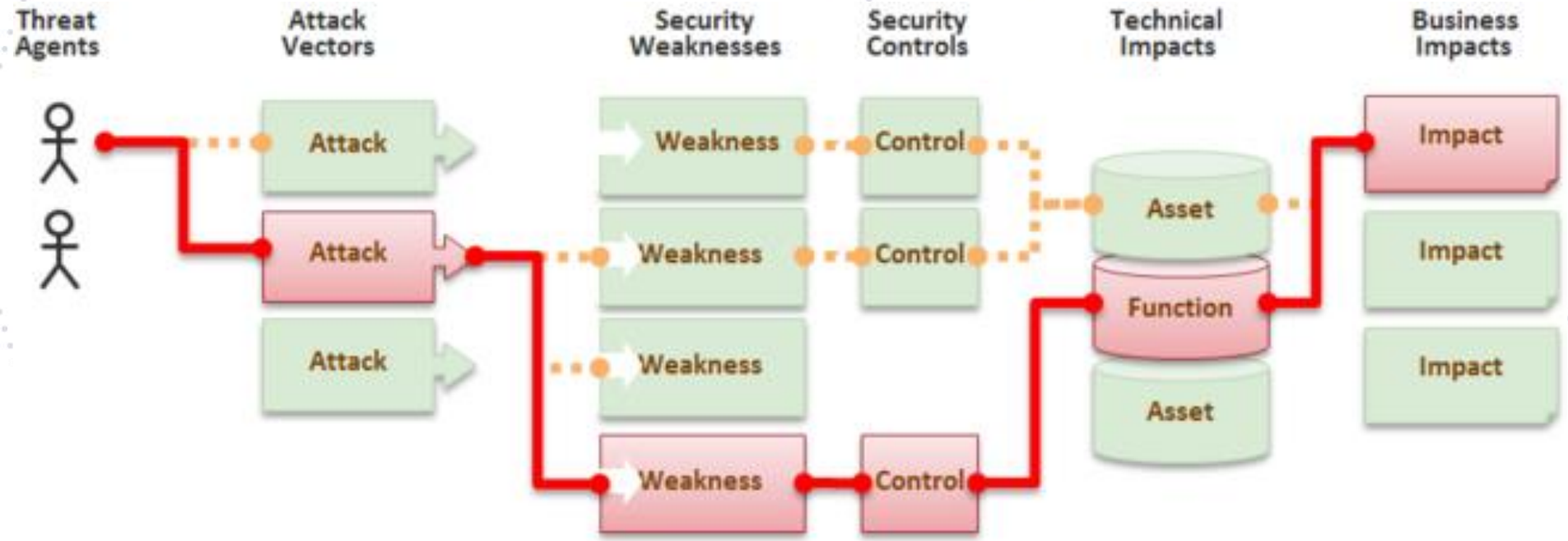
Talk

Write Secure Code → Audit Code, Result → Control Risk

- OWASP Risk Rating Methodology
- Threat Risk Modeling
- OWASP Application Security Guide For CISOs Project
- …

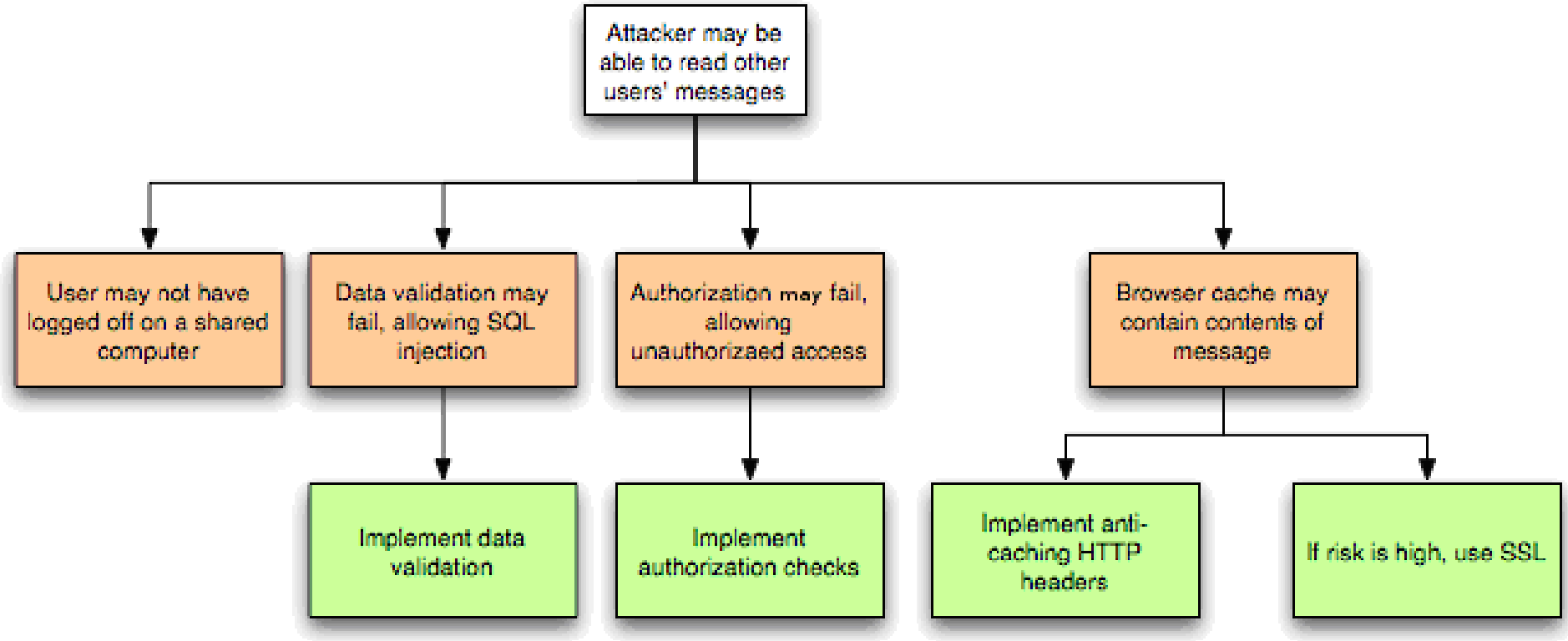
https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk



https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

	Likelihood							
	Threat agent factors					Vulnerability factors		
	Skill level	Motive	Opportunity	Size		Ease of discovery	Ease of exploit	Awareness	Intrusion detection
	4 - Advanced computer user	1 - Low or no reward	access or resources required	5 - Partners		3 - Difficult	3 - Difficult	4 - Hidden	3 - Logged and reviewed
	Overall likelihood:				3,375	MEDIUM			

2 -
3 -
4 - Hidden
5 -
6 - Obvious
7 -
8 -
9 - Public knowledge

	Technical Impact								
	Loss of confidentiality	Loss of integrity	Loss of availability	Loss of accountability		Financial damage	Reputa dama		Privacy violation
	2 - Minimal non-sensitive data disclosed	0 -	0 -	9 - Completely anonymous		1 - Less than the cost to fix the vulnerability	1 - Minimal damage	0 -	5 - Hundreds of people
	Overall technical impact:		2,750	LOW		Overall business impact:		1,750	LOW
					Overall impact:	2,250	LOW		

Overall Risk Severity = Likelihood x Impact				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH

Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Talk

https://media.mnn.com/assets/images/2016/02/bionic-arm.jpg.653x0_q80_crop-smart.jpg

Feedback

Next meetup
29.11.2017
???

Spread the word

- Mailinglisten
  - OWASP Deutschland
    - https://lists.owasp.org/pipermail/owasp-germany/
  - Stammtisch Frankfurt
    - https://lists.owasp.org/mailman/listinfo/owasp-frankfurt

- **Meetup**
  - **Stammtisch Frankfurt**
    http://www.meetup.com/de/IT-Security-Stammtisch-Frankfurt-OWASP-u-w/

- OWASP Germany
  - https://www.owasp.org/index.php/Germany
  - https://www.owasp.org/index.php/OWASP_German_Chapter_Stammtisch_Initiative
  - https://www.owasp.org/index.php/Germany/Projekte
  - https://twitter.com/#!/search/OWASP_de

Outro


Danke!

Quellen

- http://www.presseportal.de/bild/269113-preview-pressemitteilung-pol-w-alle-jahre-wieder-wir-wollen-dass-sie-sicher-ankommen.jpg
- http://www.open-forum.de/events/storytelling-open-forum-rundgespraeche-procedere-FES-2008_html_m2da1a877.gif
- http://www.luxusteich.de/img/idee.jpg
- http://www.breaksec.com/wp-content/uploads/2014/08/DoSCover.jpg
- http://static.one-schweiz.ch/1335769457/roundtable_shutterstock.jpg
- http://www.couchpotatoshop.de/images/produkte/i80/801-801-801-Brillenputztuch-Wiedersehen-0.jpg
- http://www.bier-fibel.de/wp-content/uploads/2008/10/bier_gesund.jpg
- http://t.qkme.me/3rdit3.jpg
- Intro_to_OWASP_Rochester_v5.ppt
- Owasp.org
- https://www.google.de/search?q=owasp&source=lnms&tbm=isch&sa=X