goto; copenhagen

GOTO Copenhagen 2018

Conference Nov. 19 - 21





Click ‘Rate Session’ to rate session and ask questions.



Follow us @gotocph

Pragmatic Web Security

Security training for developers



From the OWASP Top Ten(s) to the OWASP ASVS

Philippe De Ryck

@PhilippeDeRyck – philippe@PragmaticWebSecurity.com



OWASP

OWASP

OWASP Top 10 - 2017

The Ten Most Critical Web Application Security Risks



https://owasp.org

This work is licensed under a
Creative Commons Attribution-ShareAlike 4.0 International License

OWASP
Open Web Application
Security Project

OWASP
PROActive
CONTROLS

FOR DEVELOPERS

2018 **v 3.0**

10 Critical Security Areas That Software Developers Must Be Aware Of

PROJECT LEADERS

KATY ANTON
JIM MANICO
JIM BIRD

OWASP
Open Web Application
Security Project

Application Security Verification Standard 3.0.1

July 2016

- Traveling the world to deliver ***security courses***
  - In-depth web security training for developers
  - Custom training courses with developer-oriented labs
  - Covering web security, API security, Angular/React security

- 15+ years of security experience
  - Founder of **Pragmatic Web Security**
  - Author of ***Primer on client-side web security***
  - Creator of ***Web Security Fundamentals*** on edX

- Course curator of the **SecAppDev course**
  - Yearly security course targeted towards developers
  - More information on ***https://secappdev.org***



DR. PHILIPPE DE RYCK

PH.D. IN WEB SECURITY

GOOGLE DEVELOPER EXPERT

(NOT EMPLOYED BY GOOGLE)

OWASP TOP 10



The Ten Most Critical
Web Application Security Risks

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

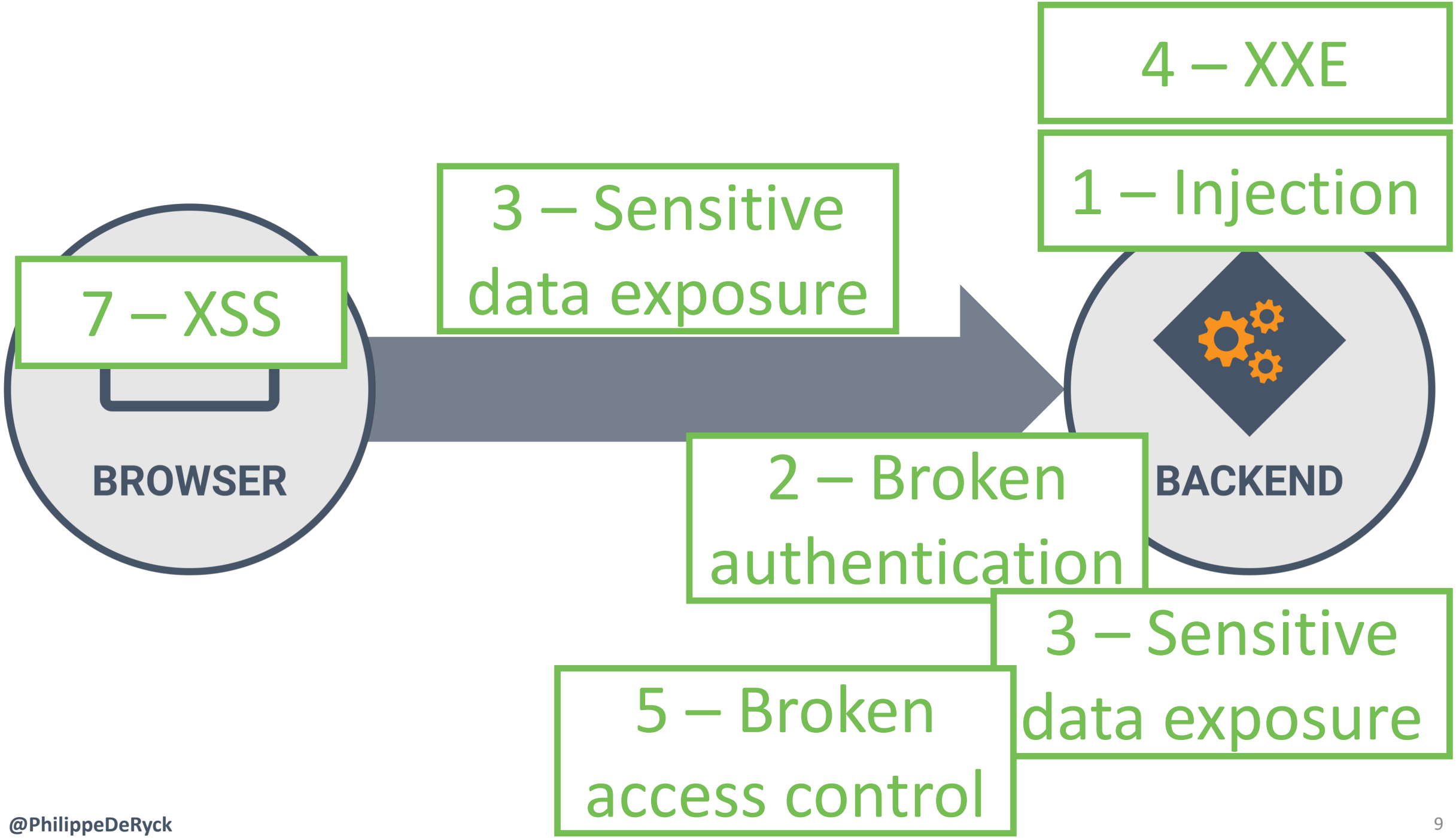
A1 :2017

Injection

7



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	


4 – XXE
1 – Injection
3 – Sensitive data exposure
7 – XSS
BROWSER
BACKEND
2 – Broken authentication
3 – Sensitive data exposure
5 – Broken access control

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

AWARENESS

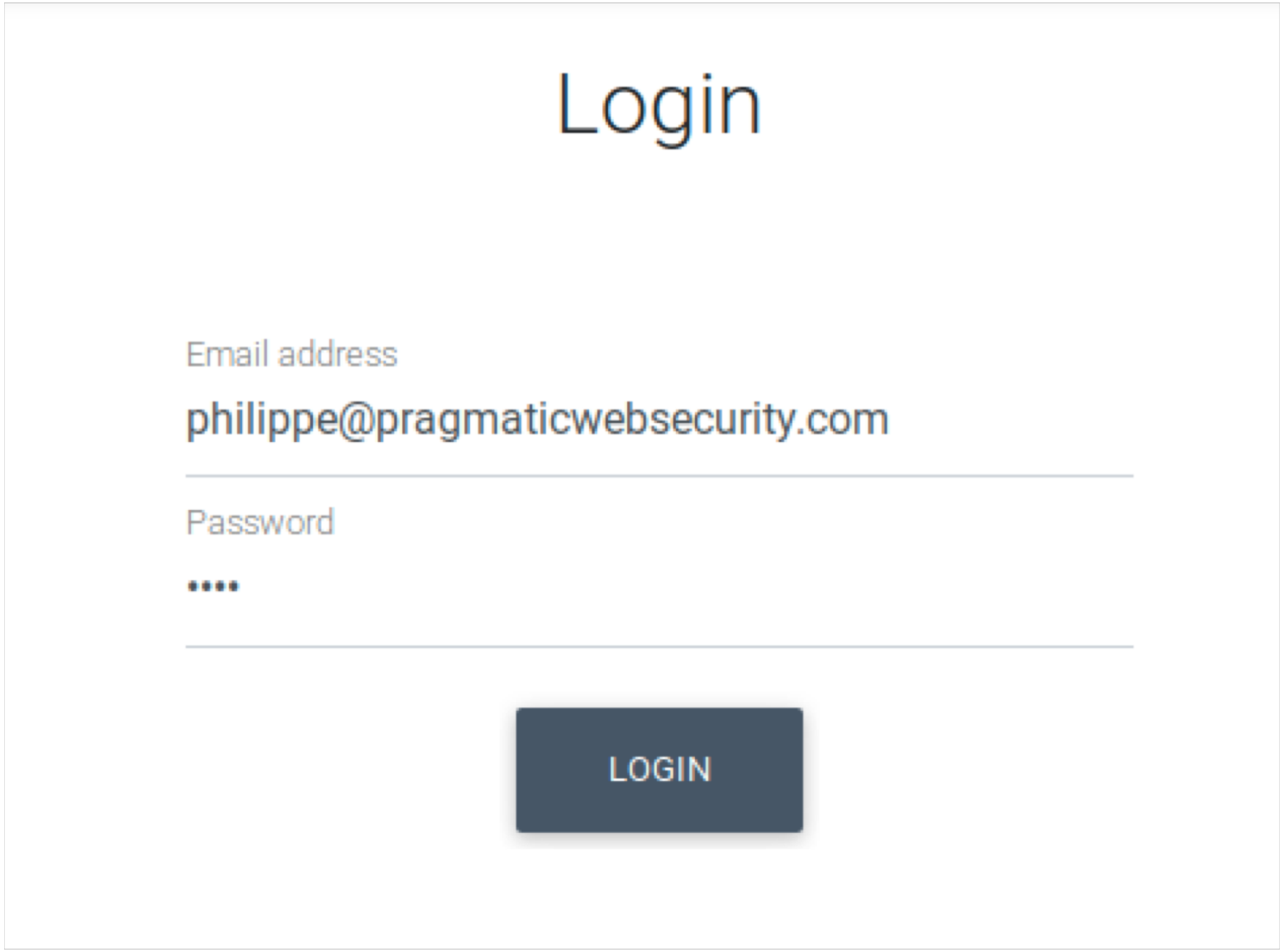
A2 :2017

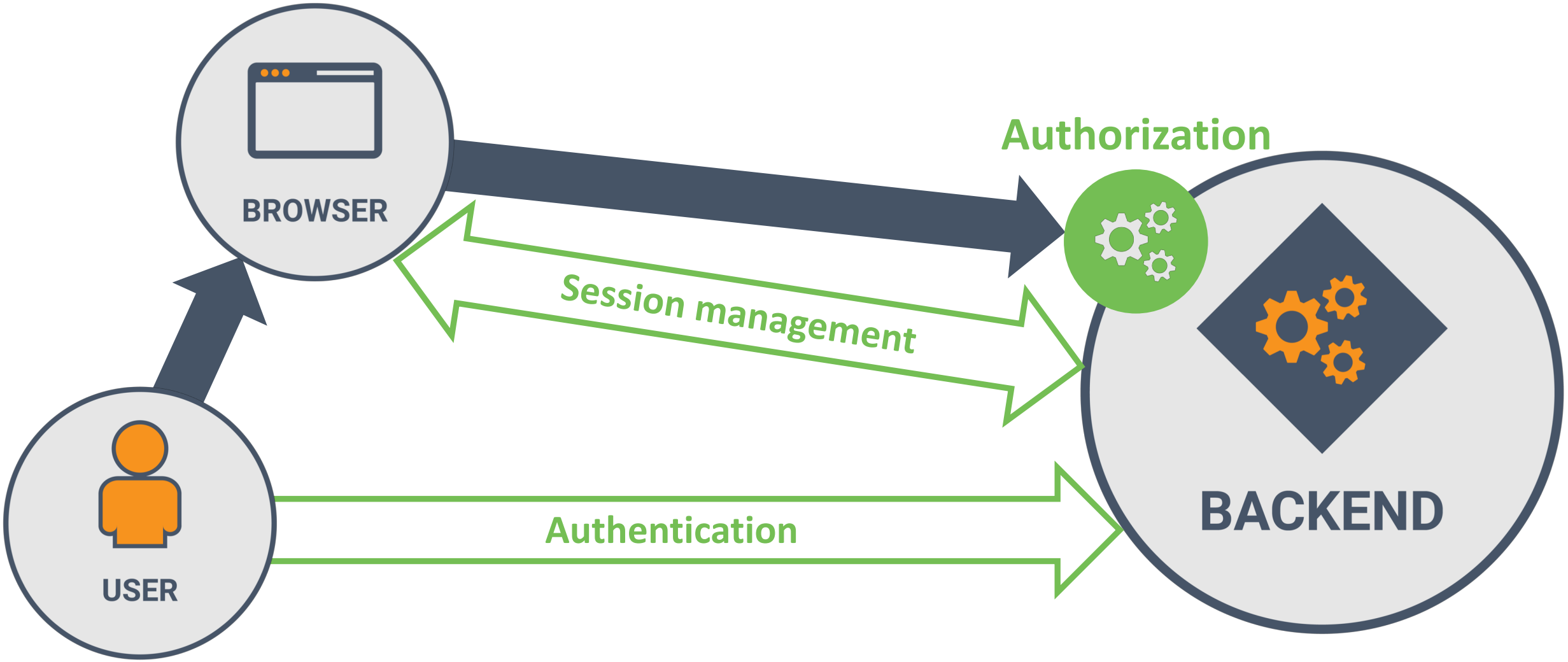
Broken Authentication

8



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.		The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.	


Login
Email address
philippe@pragmaticwebsecurity.com
Password
••••
LOGIN


BROWSER
Authorization
Session management
BACKEND
USER
Authentication

A2 :2017

Broken Authentication

Is the Application Vulnerable?

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords (see **A3:2017-Sensitive Data Exposure**).
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

How to Prevent

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

OWASP TOP 10



Awareness on the most critical issues in web applications

Brief overview of do's and don'ts in web applications

Advice is independent of application, user impact or required skills

OWASP TOP 10 PROACTIVE CONTROLS



Ten critical security areas that developers must be aware of

The list is ordered by importance with list item number 1 being the most important:

C1: Define Security Requirements

C2: Leverage Security Frameworks and Libraries

C3: Secure Database Access

C4: Encode and Escape Data

C5: Validate All Inputs

C6: Implement Digital Identity

C7: Enforce Access Controls

C8: Protect Data Everywhere

C9: Implement Security Logging and Monitoring

C10: Handle All Errors and Exceptions

C1: Define Security Requirements

Description

A security requirement is a statement of needed security functionality that ensures one of many different security properties of software is being satisfied. Security requirements are derived from industry standards, applicable laws, and a history of past vulnerabilities. Security requirements define new features or additions to existing features to solve a specific security problem or eliminate a potential vulnerability.

The list is ordered by importance with list item number 1 being the most important:

C1: Define Security Requirements

C2: Leverage Security Frameworks and Libraries

C3: Secure Database Access

C5: Encode and Escape Data

C6: Implement Digital Identity

C7: Enforce Access Controls

C8: Protect Data Everywhere

C9: Implement Security Logging and Monitoring

C10: Handle All Errors and Exceptions

AWARENESS



C6: Implement Digital Identity

Description

Digital Identity is the unique representation of a user (or other subject) as they engage in an online transaction. Authentication is the process of verifying that an individual or entity is who they claim to be. Session management is a process by which a server maintains the state of the users authentication so that the user may continue to use the system without re-authenticating. The NIST Special Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management provides solid guidance on implementing digital identity, authentication and session management controls.

Below are some recommendations for secure implementation.

Level 1 : Passwords

Passwords are really really important. We need policy, we need to store them securely, we need to sometimes allow users to reset them.

Level 2 : Multi-Factor Authentication

NIST 800-63b AAL level 2 is reserved for higher-risk applications that contain "self-asserted PII or other personal information made available online." At AAL level 2 multi-factor authentication is required including OTP or other forms of multi-factor implementation.

Level 3 : Cryptographic Based Authentication

NIST 800-63b Authentication Assurance Level 3 (AAL3) is required when the impact of compromised systems could lead to personal harm, significant financial loss, harm the public interest or involve civil or criminal violations. AAL3 requires authentication that is "based on proof of possession of a key through a cryptographic protocol." This type of authentication is used to achieve the strongest level of authentication assurance. This is typically done though hardware cryptographic modules.

Implement Secure Password Storage

In order to provide strong authentication controls, an application must securely store user credentials. Furthermore, cryptographic controls should be in place such that if a credential (e.g., a password) is compromised, the attacker does not immediately have access to this information.

PHP Example for Password Storage

Below is an example for secure password hashing in PHP using password_hash() function (available since 5.5.0) which defaults to using the bcrypt algorithm. The example uses a work factor of 15.

```
<?php
  $cost = 15;
  $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

OWASP TOP 10 PROACTIVE CONTROLS



Awareness on the most important security controls

Mainly focusing on the do's that matter for almost every application

Advice is independent of application, user impact or required skills

OWASP Application Security Verification Standard



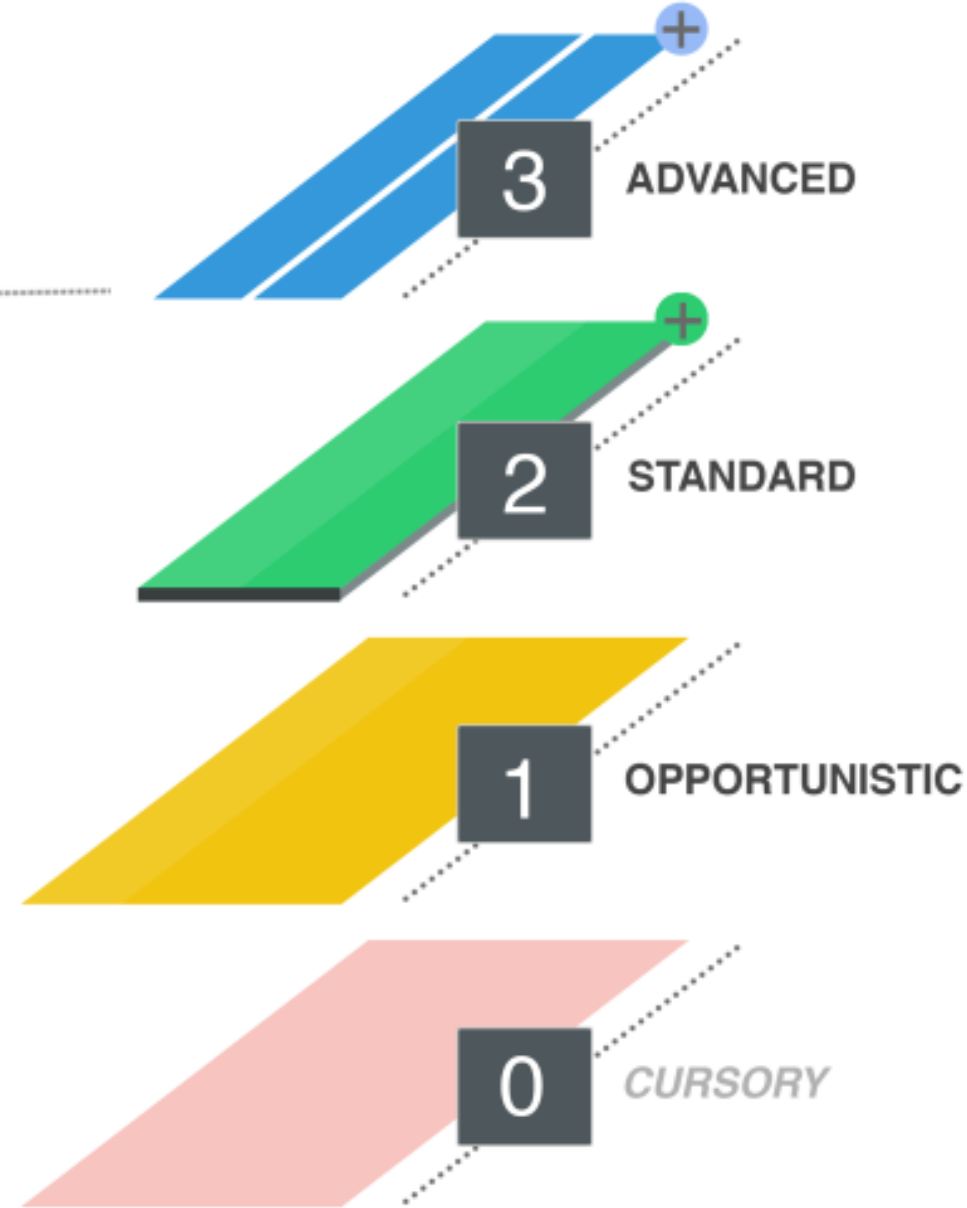
A list of security requirements or tests to determine how secure an application is

V1.	Architecture, design and threat modelling
V2.	Authentication
V3.	Session management
V4.	Access control
V5.	Malicious input handling
V7.	Cryptography at rest
V8.	Error handling and logging
V9.	Data protection
V10.	Communications
V11.	HTTP security configuration
V13.	Malicious controls
V15.	Business logic
V16.	File and resources
V17.	Mobile
V18.	Web services (NEW for 3.0)
V19.	Configuration (NEW for 3.0)

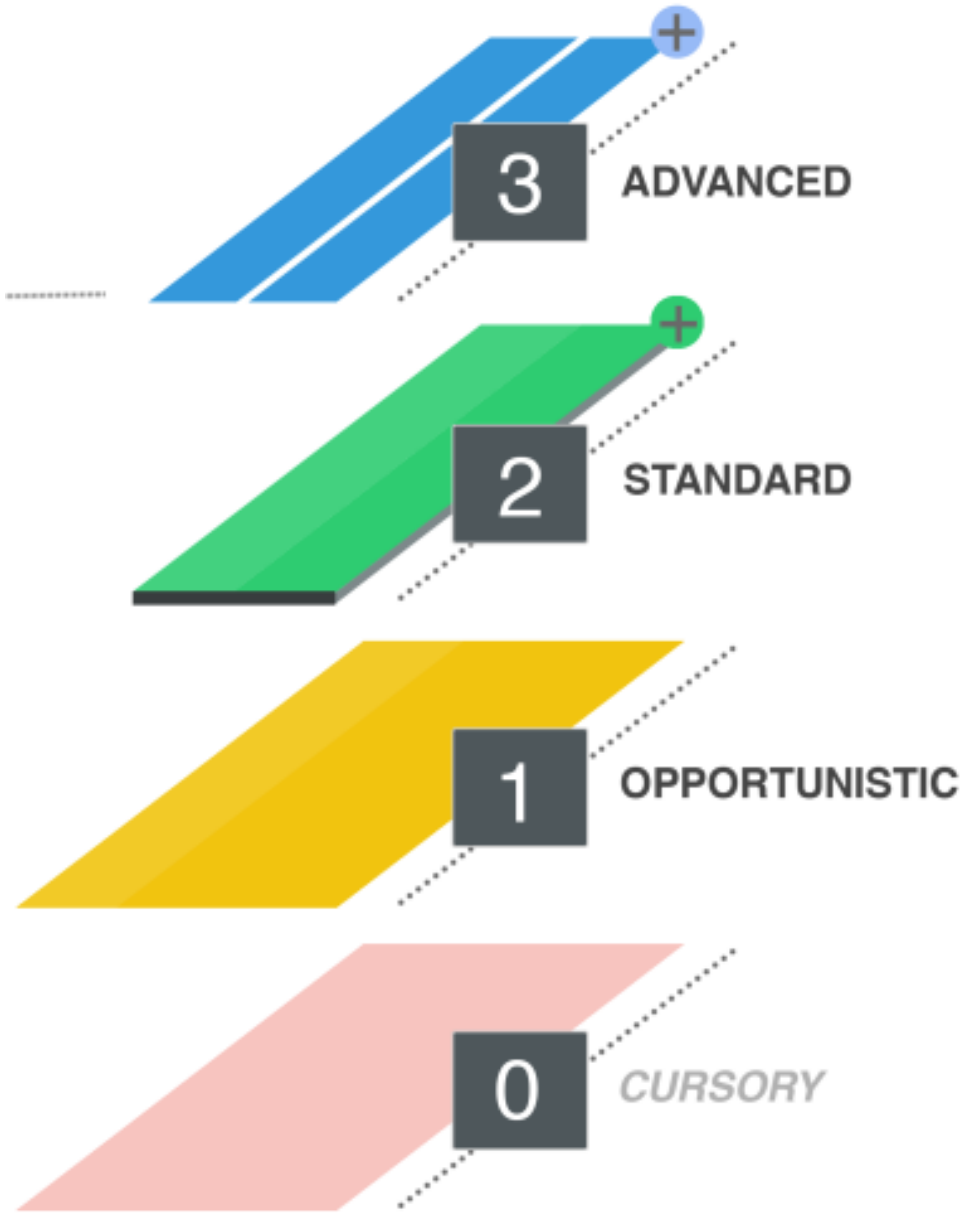
#	Description	1	2	3	Since
1.1	Verify that all application components are identified and are known to be needed.	✓	✓	✓	1.0
1.2	Verify that all components, such as libraries, modules, and external systems, that are not part of the application but that the application relies on to operate are identified.		✓	✓	1.0
1.3	Verify that a high-level architecture for the application has been defined.		✓	✓	1.0
1.4	Verify that all application components are defined in terms of the business functions and/or security functions they provide.			✓	1.0
1.5	Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the functions, and/or security functions, they provide.			✓	1.0

#	Description	1	2	3	Since
5.1	Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.	✓	✓	✓	1.0
5.3	Verify that server side input validation failures result in request rejection and are logged.	✓	✓	✓	1.0
5.5	Verify that input validation routines are enforced on the server side.	✓	✓	✓	1.0
5.6	Verify that a single input validation control is used by the application for each type of data that is accepted.			✓	1.0
5.10	Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection	✓	✓	✓	2.0

ASVS DEFINES DETAILED VERIFICATION REQUIREMENTS FOR LEVELS 1 AND ABOVE; WHEREAS LEVEL 0 IS MEANT TO BE FLEXIBLE AND IS CUSTOMIZED BY EACH ORGANIZATION



OWASP ASVS LEVELS



Applications handling critical info

Applications handling sensitive info

All applications

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access control

V5. Malicious input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)

DRIVE SECURITY PROCESSES ALIGNMENT STANDARD

#	Description	1	2	3	Since
2.1	Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation).	✓	✓	✓	1.0
2.2	Verify that forms containing credentials are not filled in by the application. Pre-filling by the application implies that credentials are stored in plaintext or a reversible format, which is explicitly prohibited.	✓	✓	✓	3.0.1
2.4	Verify all authentication controls are enforced on the server side.	✓	✓	✓	1.0
2.6	Verify all authentication controls fail securely to ensure attackers cannot log in.	✓	✓	✓	1.0
2.7	Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent password managers, long passphrases or highly complex passwords being entered.	✓	✓	✓	3.0.1

2.18	Verify that information enumeration is not possible via login, password reset, or forgot account functionality.	✓	✓	✓	2.0
2.19	Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”).	✓	✓	✓	2.0
2.20	Verify that anti-automation is in place to prevent breached credential testing, brute forcing, and account lockout attacks.	✓	✓	✓	3.0.1
2.21	Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location.		✓	✓	2.0
2.22	Verify that forgotten password and other recovery paths use a TOTP or other soft token, mobile push, or other offline recovery mechanism. Use of a random value in an e-mail or SMS should be a last resort and is known weak.	✓	✓	✓	3.0.1

2.28	Verify that all authentication challenges, whether successful or failed, should respond in the same average response time.			✓	3.0
2.29	Verify that secrets, API keys, and passwords are not included in the source code, or online source code repositories.			✓	3.0
2.31	Verify that if an application allows users to authenticate, they can authenticate using two-factor authentication or other strong authentication, or any similar scheme that provides protection against username + password disclosure.		✓	✓	3.0
2.32	Verify that administrative interfaces are not accessible to untrusted parties.	✓	✓	✓	3.0
2.33	Browser autocomplete, and integration with password managers are permitted unless prohibited by risk based policy.	✓	✓	✓	3.0.1

OWASP Application Security Verification Standard



Detailed overview of security do's and don'ts in web applications

Advice incorporates application type and development challenges

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access contr

V5. Malicious inp ng

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)



CHECKLIST

Pocket iNet ISP exposed 73GB of data including secret keys, plain text passwords

Updated: The Washington-based ISP's bucket exposed everything from passwords to internal corporate data.



By Charlie Osborne for Zero Day | October 24, 2018 -- 10:17 GMT (11:17 BST) | Topic: Security

> **the data leak was caused by a misconfigured Amazon S3 storage bucket which permitted the access and download of information without the need for authorization**

2.19



Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”).

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access control

V5. Malicious input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)



AUTOMATED SECURITY TESTING

AdultFriendFinder hacked: 400 million accounts exposed

Huge breach reveals 15 million "deleted" accounts among compromised data...

TOM MENDELSOHN - 11/14/2016, 3:13 PM

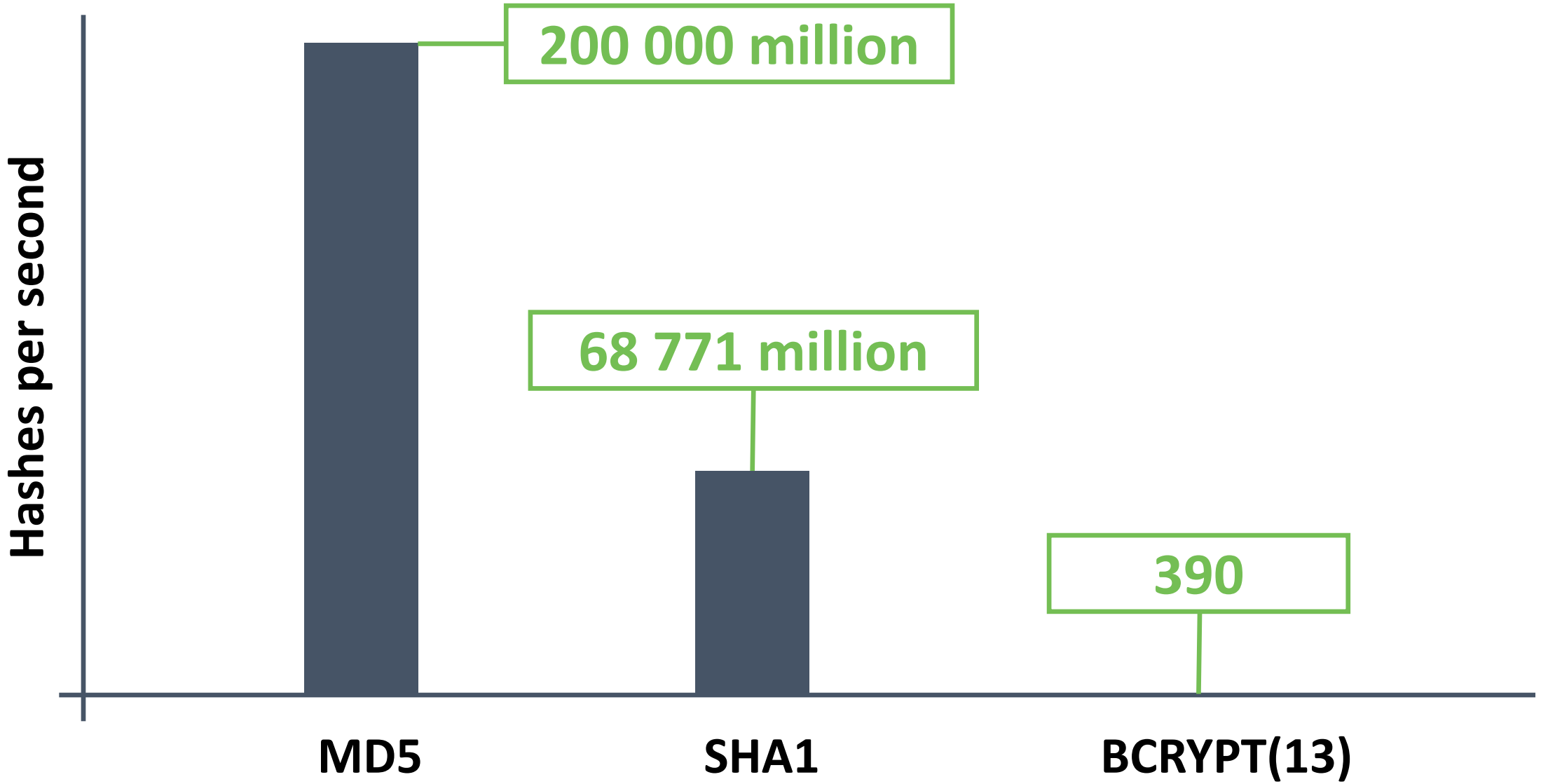
> The passwords were either kept in plain text format, or used the largely discredited SHA1 hashing algorithm.

MD5 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
md5_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	52 GB 64 GB	Perfect Non-perfect	Perfect Non-perfect
md5_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	460 GB 576 GB	Perfect Non-perfect	Perfect Non-perfect
md5_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090	99.9 %	127 GB 160 GB	Perfect Non-perfect	Perfect Non-perfect
md5_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	690 GB 864 GB	Perfect Non-perfect	Perfect Non-perfect


GEFORCE GTX
GEFORCE GTX
GEFORCE GTX
GEFORCE GTX
GEFORCE GTX
GEFORCE GTX
GEFORCE GTX

Is hash cracking really that fast?



2.13



Verify that account passwords are one way hashed with a salt, and there is sufficient work factor to defeat brute force and password hash recovery attacks.

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access control

V5. Malicious input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)



SECURE CODING GUIDELINES

USERNAME HARVESTING THROUGH TIMING ATTACKS

```
1  List<User> users = new UserDAO().findAllByEmailWithPassword(email);
2  if(users.size() == 1) {
3    User user = users.get(0);
4    if(AuthenticationUtils.verifyPassword(user, password)) {
5      Logger.info("Authentication successful.");
6      return redirectAfterLogin();
7    }
8    else {
9      Logger.warn("Invalid password. Authentication failed");
10     return handleLoginError();
11   }
12 }
13 else {
14   Logger.warn("No matching user account found. Authentication failed");
15   return handleLoginError();
16 }
```

100 – 200ms operation

Almost instant operation

2.28



Verify that all authentication challenges, whether successful or failed, should respond in the same average response time.

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. control

V5. ous input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)

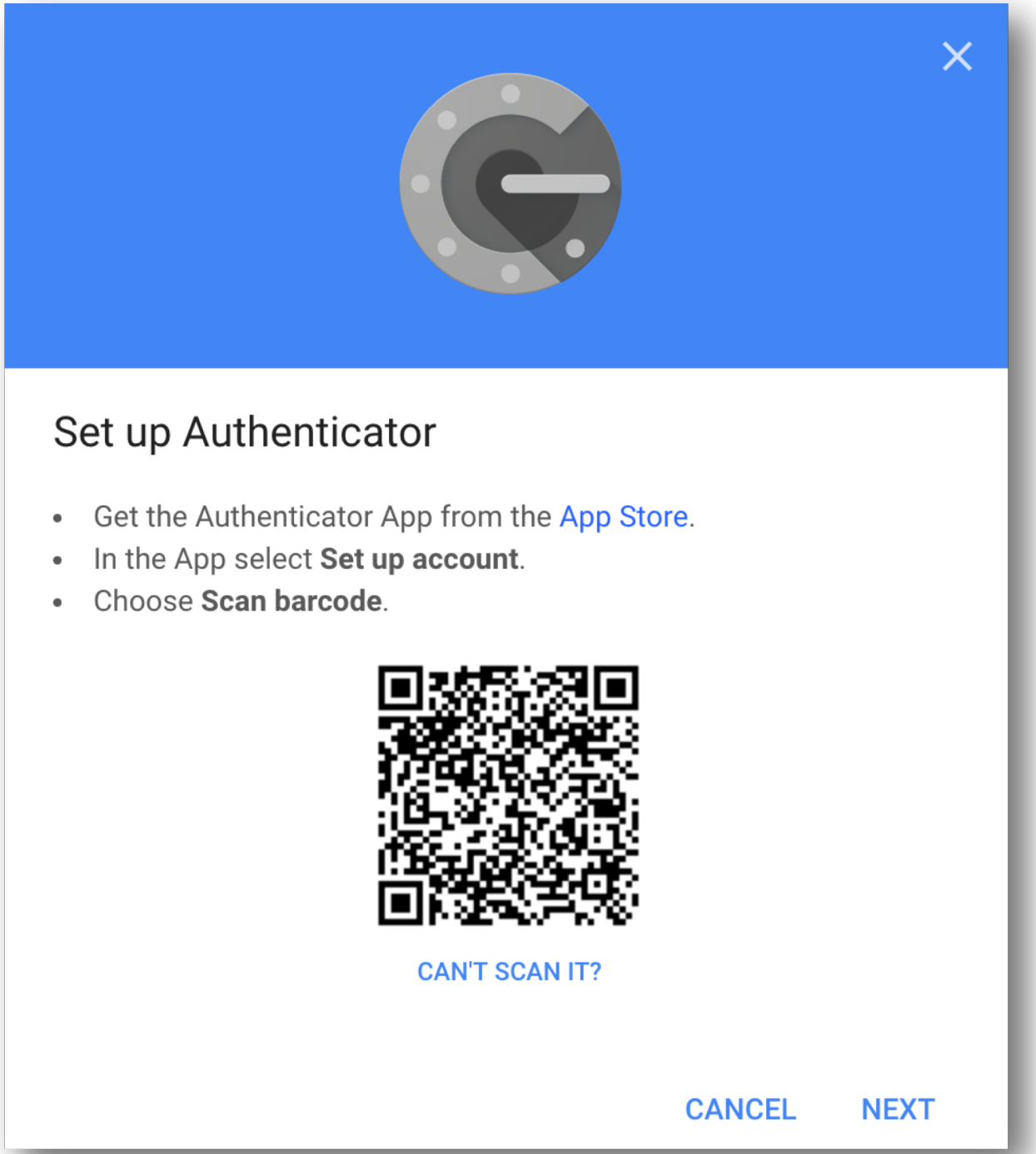


PRIORITIZE SECURITY

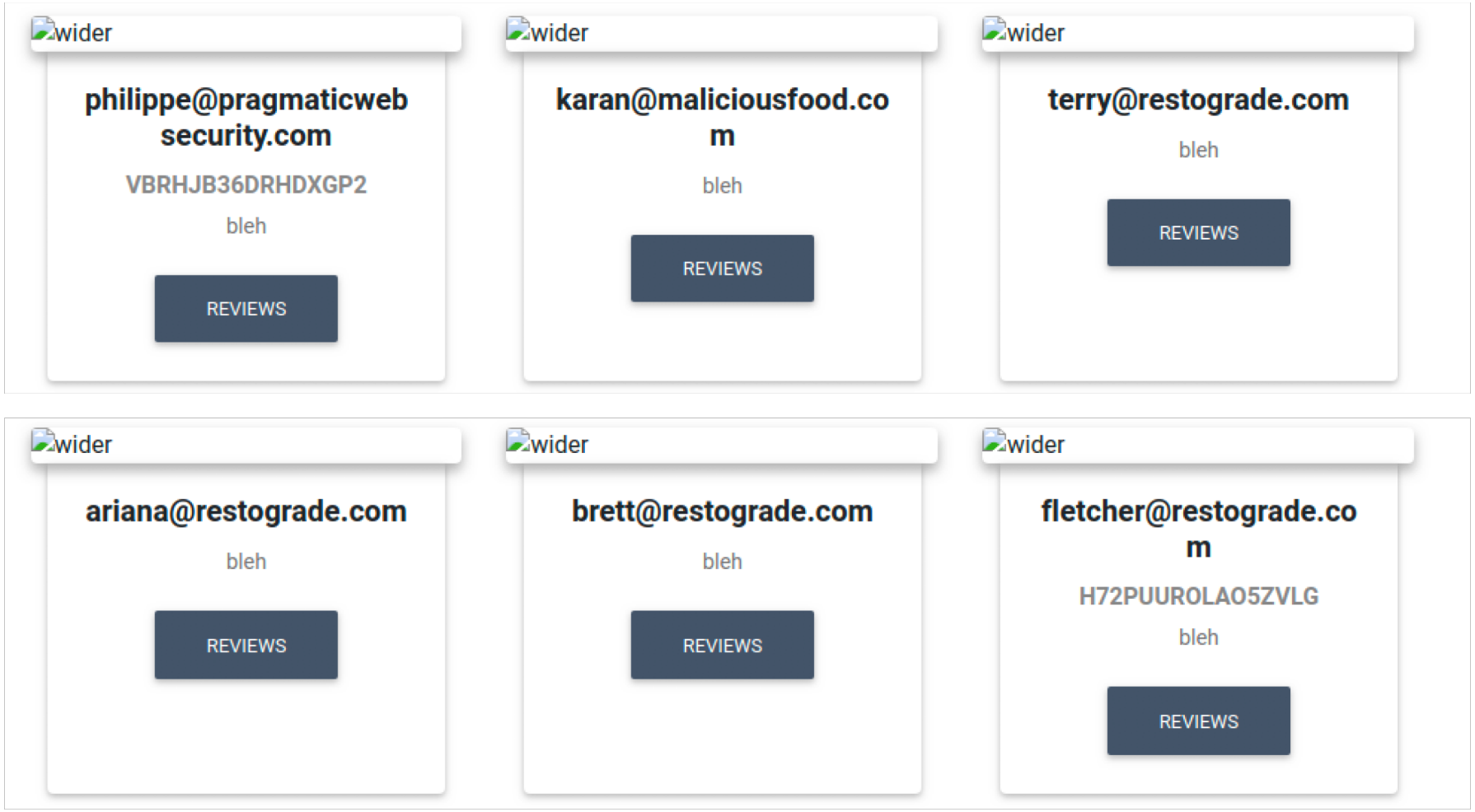
2.31



Verify that if an application allows users to authenticate, they can authenticate using two-factor authentication or other strong authentication, or any similar scheme that provides protection against username + password disclosure.


Set up Authenticator
Get the Authenticator App from the App Store.
In the App select Set up account.
Choose Scan barcode.
CAN'T SCAN IT?
CANCEL
NEXT

x' UNION SELECT id,email, totpsecret, 'bleh', 'bleh.png' FROM users WHERE 'x%' = 'x



5.10



Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection

X



Verify that all shared secrets are encrypted and stored in a protected location

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Acc ol

V5. Ma ut handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)



FORK AND CUSTOMIZE

ASVS v4.0

In progress

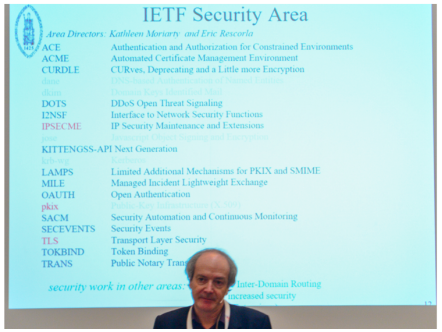
Available on Github

Participate!

SecAppDev 2019

February 18 - 22, Leuven, Belgium









1-day workshops

Building secure web & web service applications
Jim Manico

Whiteboard hacking (aka hands-on Threat Modeling)
Sebastien Deleersnyder

Securing Kubernetes the hard way
Jimmy Mesta

5-day dual-track program

Crypto, AppSec Processes, web security, access control, mobile security, ...

Pragmatic Web Security

Security training for developers



/in/PhilippeDeRyck

@PhilippeDeRyck

philippe@pragmaticwebsecurity.com

goto; copenhagen

GOTO Copenhagen 2018

Conference Nov. 19 - 21



Please

Remember to rate this session

Thank you!



Follow us @gotocph