Overview of the OWASP Top Ten Proactive Controls

Sean Scott

University of Missouri—St. Louis

SP21-INFSYS6868-001

Dr. Anshuman Singh

Abstract

The *OWASP Top Ten Proactive Controls* is a list of the ten most important things a development team can do to proactively prevent application security bugs. Studies have shown (Veracode, 2020) that SAST and DAST tools are effective in reducing the number of bugs that are released into production. SAST and DAST tools are automated tools that can find certain software security bugs. However, while SAST and DAST tools can help find bugs prior to releasing the product code into production, implementing the *OWASP proactive controls* document can reduce the bugs found by SAST and DAST tools and, therefore, decrease the amount of rework required.

Keywords: OWASP, Top Ten, Risks, Proactive Controls

Overview of the OWASP Top Ten Proactive Controls

The *OWASP Top Ten Proactive Controls* document lists the top ten techniques that can proactively prevent many application security bugs (OWASP, 2018, p. 5).

The purpose of the list is two-fold. First, to inform application developers on the techniques in the document and, second, to get application developers to begin thinking about application security. The document is a starting point for security and should not be considered as comprehensive documentation on preventing application security bugs (OWASP, 2018, p. 3).

History

The *OWASP Proactive Controls* list was first created by the OWASP Foundation in 2014 (OWASP, 2014 B). It was updated in 2016 (OWASP, 2016) and then the latest version was published in 2018 (OWASP, 2018, p. 1). In performing research for each publication, the OWASP team gathers data and then scores the data and selects the top ten items to include based upon the need at the of the time of the study (OWASP, 2018, p. 3). *Table 1 OWASP Proactive Control Version Comparison* lists the top ten proactive controls the OWASP organization recommended for each of the three versions of the preventative controls lists that have been published to date.

The OWASP organization also publishes another, more well-known list, the *OWASP Top Ten Risks* list. The *Top Ten Risks* list represents the top ten application security risks the OWASP team has determined exist in the wild at the time of publication. The *OWASP Proactive Control* list describes ten activities or actions development teams can take to proactively prevent introducing application security bugs represented in the risk list into their code (OWASP, 2018, p. 6). The risks and controls lists are published at different times, by different teams, so there may not be a direct correlation between the two but, in practice, adhering to the suggestions in the latest *OWASP Proactive Controls* list tends to protect against the risks defined in the latest risk list.

The Proactive Controls

C1: Define Security Requirements

Security requirements help prevent all types of application security bugs (OWASP, 2018, p. 9). The *Application Security Verification Standard* (ASVS) is a categorized list of security requirements that can be helpful to teams as they develop their security requirements (OWASP, 2020, p. 7). The ASVS model is divided into three levels.

Level 1 is the lowest assurance level and "adequately defends against application security vulnerabilities that are easy to discover and included in the OWASP Top 10 and other similar checklists" (OWASP, 2020, p. 11). It represents an effective level of security for many public Websites that store little or no personal information. For example, a Website that only has a list of email addresses it uses for its login scheme.

Level 2 is more appropriate for Web applications that store more personal information than just an email address. As the ASVS puts it, it should be used for "applications that handle significant business-to-business transactions, including those that process healthcare information, implement business-critical or sensitive functions, or process other sensitive assets, or industries where integrity is a critical facet to protect their business, such as the game industry to thwart cheaters and game hacks" (OWASP, 2020, p. 11).

Level 3 requires an extreme amount of security architecture and coding. The ASVS states Level 3, "is typically reserved for applications that require significant levels of security verification, such as those that may be found within areas of military, health and safety, critical infrastructure, etc." (OWASP, 2020, p. 11).

Two of the more commonly used security maturity models also require security requirements to be documented. The BSIMM includes "*Standards and Requirements*" as one of the twelve practices it defines (Synopsis, 2020, p. 39). *Task SR1.3 Translate compliance constraints to requirements* requires development teams to turn "compliance constraints…into software requirements for individual projects" (Synopsis, 2020, p. 71).

In the OpenSAMM model, the *Security Requirements (SR)* practice falls under the *construction* business function. It requires teams to commit time to, "proactively specifying the expected behavior of software with respect to security" (OpenSAMM, 2020, p. 12).

One common problem among application security documents, like the *OWASP Proactive Controls* list, BSIMM, and OpenSAMM, is they blur the distinction between a requirement and a constraint. By definition, a requirement is something that must be present[1], while a constraint is a limitation[2] on how a requirement is delivered. By this definition, a rule that states that all queries must be parameterized queries is not technically a requirement that can be fulfilled, because it must be done for every functional requirement for which database access is needed. This makes it a constraint on how functional requirements are built. Conversely, a requirement stating all passwords must be at least 8 characters in length and include at least one number and one special character, is a requirement that can be coded and then marked as complete.

While this may sound like semantics, it should be remembered that tools such as Jira, a common tool used to manage requirements, is designed to have a requirement (in the form of a task or story) opened, assigned, completed, and then closed. This process does not, for example, support a "requirement" to use parameterized queries everywhere database interaction is required. This type of need is better defined as a constraint and documented as part of the description of the functional requirement or, better yet, as part of the acceptance criteria.

C2: Leverage Security Frameworks and Libraries

Secure frameworks and libraries can help to prevent a wide range of Web application vulnerabilities (OWASP, 2018, p. 10). The National Vulnerability Database is a repository of known software vulnerabilities. For example, as of the writing of this document, the database shows 79 open

[1] See https://www.merriam-webster.com/dictionary/requirement
[2] See https://www.merriam-webster.com/dictionary/constraint

vulnerabilities that can affect the ReactJS library (NIST, 2021). Several security frameworks use this data to inform development teams about vulnerabilities in frameworks they are using, including OWASP Dependency Check, RetireJS, and JFROG Xray. These security frameworks, when installed and configured properly, work to help development teams keep the code they import into their solution as free from vulnerabilities as possible (OWASP, 2018, p. 10).

Security frameworks can also be used to implement functionality directly into the application. For example, OAuth 2.0 is an open-source secure authorization framework that has the benefit of being written and reviewed by security experts. It can be argued that authentication, authorization, login management, and session management form the foundation of application security. If these are not secure, then there is no way to assure only authorized users have access to the system data.

C3: Secure Database Access

The second most dangerous access an attacker can gain, after root access, is the ability to manipulate the database. Gaining access to a system's database would allow an attacker to perform attacks from defacing the application to destroying or stealing data.

According to the *OWASP Proactive Controls* document (OWASP, 2018, pp. 11-12) there are four main areas a development team should consider regarding database access: 1) securing their queries, 2) securing the database configuration, 3) securing authentication, and 4) securing communication.

Securing all four of these areas can help prevent Web-based injection attacks (OWASP, 2018, p. 12) and mobile attacks described in *M1: Weak Server Side Controls* of the 2014 version of the *OWASP Mobile Top Ten* (OWASP, 2014 A).

Secure Configuration

Unnecessary features and services

Most modern database management systems come with various components other than the database engine. For example, SQL Server 2019 includes Analysis Services, Reporting Services, Integration Services, and Machine Learning Services among others (Microsoft, 2021 A). Turning off unused

components should be part of the system hardening process performed prior to using the DBMS. Every running service is a potential entry point into the application or server.

Service Packs

Service packs fix bugs as they are found. Failure to install services packs on a regular basis introduces the organization to increased risk of an attack. Microsoft SQL Server is updated several times per year (Microsoft, 2021 B).

Unused protocols

Microsoft SQL Server supports three network protocol configurations, Shared Memory, TCP/IP, and Named Pipes (Microsoft, 2019 A). Turning off unused protocols gives attackers fewer avenues of attack.

Change ports

The port for the Default Instance of SQL Server is 1433 (Anis, 2012). Other services have their own default ports set. Changing the ports used by each service makes it more difficult for attackers because they will not know what port goes to each of the services.

SQL Server browser service

"The SQL ServerBrowser [sic] program runs as a Windows service. SQL Server Browser listens for incoming requests for Microsoft SQL Server resources and provides information about SQL Server instances installed on the computer" (Microsoft, 2017 A). Information provided by the Server Browser service is invaluable to attackers. It should be turned off when it is not needed.

Disable shell

The *xp_cmdshell* transact SQL command is available in Microsoft SQL Server. It "spawns a Windows command shell and passes in a string for execution. Any output is returned as rows of text" (Microsoft, 2019 C). Running Windows O/S commands from within the database is a dangerous procedure. Disabling the shell will prevent attackers from running a Windows O/S command through the SQL Server.

Secure Authentication

SQL Server has two authentication modes: Windows and Mixed Mode (Microsoft, 2017 B). Mixed mode allows Windows and SQL Server authentication. There is no SQL Server authentication only mode. Because allowing SQL Server authentication requires Windows mode to also be enabled, even if not being used, using Windows mode limits the footprint of attack. It also allows Kerberos security protocol to be used and "offers additional password policies that are not available for SQL Server logins" (Microsoft, 2017 B).

Secure Communication

TLS should be used to encrypt data in transit between the client and server. "TLS allows the client and the server to authenticate the identity of each other. After the participants are authenticated, TLS provides encrypted connections between them for secure message transmission" (Microsoft, 2021 C).

C4: Encode and Escape Data

While not failproof, using parameterized queries, rather than concatenating strings received from user input to build database queries, is recommended. Escaping or encoding data can also be used to help keep queries secure. For example, adding a backslash in front of all single quotes will cause most SQL database engines to treat the single quote as text and not recognize it as a string terminator.

Another option is to use quoting syntax. For example, the string *' or 1 = 1* -- can be used to bypass query logic in certain unsecure queries. However, wrapping the dangerous string with quote syntax, *q'[' or 1 = 1 --]'* will instruct the database engine to view the entire string provided by the user interface as string text and will not be interpreted by the query engine.

Encoding converts characters into their equivalent HTML character code. For example, encoding a less-than character "<" to the string *<* will tell the HTML rendering engine to display the less-than character, rather than interpreting it as the start of an HTML tag. The result of proper encoding will cause any injected code to be printed to the screen, rather than being interpreted by the browser.

Proper encoding and escaping can help prevent *Injection and Cross-site Scripting* attacks (OWASP, 2018, p. 15).

C5: Validate All Inputs

Input validation is an important feature of user experience design. However, client-side user input validation should never be considered a form of data security. Attackers can bypass client-side validation by turning off the script or communicating directly with the server using a browser or a tool like Burp Suite.

Therefore, it is important for all data coming into the system be validated by the back-end code. According to the *OWASP Proactive Controls* guide, there are two types of validity: Syntax and Semantic (OWASP, 2018, p. 16). If data is syntactically valid, it has the correct form. For example, if the system expects a social security number, it might check to make sure there are nine digits and no other characters, or eleven digits with dashes in the fourth and seventh position. If it expects a date, then it should assure the data looks like a valid date was sent. Regular expressions are one method for determining if a value is syntactically valid. However, they can be difficult to construct correctly (OWASP, 2018, p. 16). Regular expressions can also cause serious problems if not designed carefully. For example, input strings can be crafted that can cause some regular expressions to get lost in "backtracking loops" (Sebastian, 2021) and create a denial-of-service interruption, known as a ReDoS (Regular Expression Denial of Service.)

Semantic validation means the data is in the correct range. For example, if a piece of data represents an invoice date, then we might assume the date should be on or after the date for the earliest invoice in the system, and not in the future.

Another tool available to us to validate input is the use of whitelists (OWASP, 2018, p. 16). A whitelist is a list of acceptable values. For example, a list of valid countries or states. We might also use whitelists to validate a request is originating from an IP address which exists on a pre-defined list. However, caution must be taken for this kind of whitelist validation because attackers can spoof their IP address. Blacklisting is the opposite of whitelisting, i.e., it is a list of "bad" things that should not be accepted. In general, blacklists are ineffective and should be avoided in most instances. It is impossible to maintain a list of every iteration of every bad value. However, with that said, sometimes blacklists are the best choice. For example, anti-virus software essentially uses a list of known virus fingerprints to

determine if a file is infected. Likewise, most modern browsers will read a list of known dangerous sites and warn the user if they attempt to browse to a URL on the list. These are both examples of effective use of blacklists. One common attribute each of these have is a dedicated group of people or an organization that works to keep the list updated. Without constant updating, blacklists become stale and ineffective. And, worse than that, they continue to give the illusion of security.

According to the OWASP Proactive Controls guide, effectively validating input can:

- reduce the attack surface of applications and can help make attacks against an application more difficult, and
- help prevent *Cross-site Scripting* and *SQL Injection* attacks, although validating input is not sufficient by itself (OWASP, 2018, p. 20).

C6: Implement Digital Identity

"Digital Identity is the unique representation of a user (or other subject) as they engage in an online transaction" (OWASP, 2018, p. 21). In other words, a digital identity is an electronic data string, such as a cookie, token, or key, that can be used to identify a single entity. A digital identity is created by first creating the identity object and then verifying the entity is who the system expects them to be. Once the entity has verified their identity, the digital identity is then trusted to represent the entity. This process of assuring the entity is who they are expected to be is known as *authentication*. Authentication is the first step in enforcing access control. Once we know the entity is who we expect them to be, the next step is to verify the entity is allowed to perform each activity they attempt to perform. This step is called *authorization*. The most secure authentication methods are of no value if proper authorization is not performed prior to each action. Authorization will be discussed in the next section, *Enforce Digital Identity*.

A login ID and password is a common form of authentication. However, multi-factor authentication use is on the rise by many online services (Microsoft Support, n.d.). Authentication normally requires some kind of identity, e.g., a login ID or email address, and then something else that helps prove the

person is the actual person represented by the identity string. This proving data often falls into one of three categories:

- Something you know, e.g., a PIN or password,
- Something you have, e.g., a smartphone or other physical device,
- Something you are, e.g., fingerprint, or other biometric.

Multi-factor authentication combines two or more pieces of proving data. For example, a password and a one-time pin sent to your smartphone. Common examples of one-time pin providers are RSA tokens and the Microsoft Authenticator app. RSA soft tokens, which are smartphone apps capable of generating pins, and the Microsoft Authenticator app are potentially even more secure than hard (physical) tokens. A hard token continuously displays a PIN valid for a short period of time. However, most soft-token software requires the person to not only have the physical item, for example, the smartphone, but have the ability to log into it. Soft tokens can often be setup to accept a fingerprint for logging in. The effect of this is, in order to gain access to the account (short of hacking the system) the individual needs to know their login ID and their password, have the smart phone in their possession, and have the correct fingerprint to enable the app to generate an ID.

The final step in this process is to manage the session. The session is typically where the authenticated identity is held. Making sure the session is secure and destroyed when the entity logs off is imperative. If sessions are not handled securely and destroyed in a timely manner, attackers will have a greater opportunity to hijack the session and gain access to the application as if they were the owner of the session.

C7: Enforce Access Controls

Knowing the digital identity represents a valid entity is just the first step in securing an application. As mentioned in the previous section, the next step is to make sure the entity only performs those actions they are authorized to perform. Authorization should be done prior to each event, especially those that read from or write to a database. A proper authorization scheme should assume the entity is NOT able to perform an action until it determines otherwise. This is known as the *deny by default* (OWASP, 2018, p.

28) principle. Another principle that should be implemented is that of *least privilege*. Least privilege means that entities should be given the least amount of access necessary to perform the job (OWASP, 2018, p. 28). In other words, an admin-level account should never be used by a system to login to a database. A database administrator should create a login for the database which only has the privileges it needs to run properly.

The *OWASP Top Ten Proactive Controls* guide (p. 27) describes the four most common types of access control:

- **Discretionary Access Control** (DAC)—which limits access based upon identity and need-to-know.
- **Mandatory Access Control** (MAC)—which limits access based upon the sensitivity of the data.
- **Role Based Access Control** (RBAC)—which limits access based upon a set of defined roles and those roles to which the entity has been assigned.
- **Attribute Based Access Control** (ABAC)—which limits access based upon an arbitrary set of attributes assigned to the object and the entity.

Another common type of access control is **Rules Based Access Control** (RuBAC)—which limits access based upon a set of business rules. For example, a rule-based access control could allow a user to only have "access to files during certain hours of the day" (Gentry, 2021).

Adequate authentication, authorization, and session management can help prevent *broken authentication and session management*, which is A2 on the *2017 OWASP Top Ten Risk* list and *broken access control*, which is A5 on the *2017 OWASP Top Ten* risk list. It can also help prevent *poor authorization and authentication*, which is M5 on the *2014 Mobile Top Ten Risk* list.

C8: Protect Data Everywhere

Businesses often protect their data because it is in their best interest to do so. Trade secrets and other information that gives the business an edge against their competition is often kept locked away and under

tight controls. Publicity from past data breaches also put pressure on businesses to protect their client data, so they do not find themselves in the same position as the organizations they heard about in the news.

However, there are also government and industry regulations and rules that have been implemented to put even more pressure on businesses to protect their client data (Raynovich, 2020). A few of the more common include:

- **Federal Information Processing Standard Publication** (FIPS)— establishes a security standard for the design and implementation of cryptographic modules.
- **General Data Protection Regulations** (GDPR)—consolidates the information security laws in the European Union.
- The **Federal Information Security Management Act** (FISMA)— requires federal agencies or state agencies managing federal programs, such as Medicare, to protect sensitive data.
- **Health Insurance Portability and Accountability Act** (HIPAA)—protects the integrity, privacy, and availability of personal health information in the United States.
- **Payment Card Industry Data Security Standards** (PCI DSS)—is a set of guidelines for data security created by the payment card industry, e.g., credit card companies, and requires any entity that accepts credit cards to adhere to the principles and policies described in the document.
- **Cybersecurity Information Sharing Act** (CISA)— is U.S. federal law that allows technology and manufacturing companies to share Internet traffic information with the government around cyber threats (some fear it may weaken privacy protections).

Protecting data is expensive. Physical security requires an investment in locks and hardened enclosures. Electronic security is often accomplished through encryption. Implementing a proper encryption scheme takes time and a good understanding of how the selected encryption mechanism works. In addition to being expensive monetarily, encryption is also slow. Encrypting all data everywhere

would impact the responsiveness of the application without providing a significant amount of security over just encrypting sensitive data. Therefore, the first step in implementing an encryption scheme is to classify the data. There are many ways to do this, and the rules and regulations listed earlier can provide guidance; but the most important distinction to be made is to determine what data needs to be protected.

Once the data has been classified, the sensitive data needs to be protected, which usually is accomplished by encryption. The sensitive data needs to be encrypted in two places: in transit and at rest. (A third place, known as *data in use*, also exists but, since that describes data in RAM or cache memory, it is outside of the scope of *protect data everywhere* because the data in memory needs to be unencrypted so it can be analyzed and manipulated.)

In transit encryption means using a mechanism such as *transport layer security* (TLS) to encrypt the data as it travels from the server to the client and back. TLS requires a certificate to exist on the server, and for the client to use the HTTPS protocol.

Encryption at rest means using encryption when storing data. Microsoft SQL Server (Microsoft, 2019 B) and Oracle Database Server (Oracle, n.d.) both support transparent data encryption (TDE). TDE encrypts and decrypts the data automatically which, for the most part, makes the process invisible to the application.

Protecting data everywhere using encryption helps prevent *sensitive data exposure*, which is A3 on the *2017 OWASP Top Ten Risks* list; and *insecure data storage*, which is M2 on the *2014 OWASP Top Ten Mobile Risks* list.

C9: Implement Security Logging and Monitoring

Implementing security logging and monitoring is the ninth proactive control; and the first one that does not actually do anything to help prevent an attacker from gaining access. However, what it does do, when implemented correctly, is to help organizations limit the damage done by an attacker by helping them detect the attack as early as possible.

There are several logging frameworks that are worth mentioning, including Graylog[3], Logstash[4], Fluentd[5], Flume[6], and Logalyze[7]. All of these frameworks do the "heavy lifting" of logging, making them easier to implement than a custom solution. It is still up to the development team to decide what events should be logged; then add the call to the logging framework, passing it the appropriate information.

Logging is not useful if the data is never analyzed. There are several log aggregator and analyzer tools available, including Papertrail[8], Loggly[9], LogDNA[10], and Splunk[11]. These tools allow the organization to aggregate the massive amount of data created by a logger and analyze it to see patterns that might suggest an attacker is attempting to gain or has gained access to the system. Logs can also feed into an automated intrusion detection system that can automatically send an email or text message warning of a potential security event.

Logs are also a valuable resource for forensic analysis and investigation. They provide non-refutable evidence of an event occurring. Non-refutation is an important part of an organizational security strategy because it holds individuals accountable for what happens through the use of their credentials.

As important as logs are, there are a couple caveats each team should keep in mind. First, logs take a lot of space. So, it is important to analyze the logging needs carefully, so they do not log too little or too much information. The second caveat is to be careful about what data gets stored in the logs. While it may be beneficial to have logs that can identify the line of code that caused an error, along with the data that was present at the time the error occurred, the development team needs to make sure that they are not storing personal data in the logs. Although, this is not always possible. In those cases, the least amount of

---

[3] https://www.graylog.org/
[4] https://www.elastic.co/logstash
[5] https://www.fluentd.org/
[6] https://flume.apache.org/
[7] http://www.logalyze.com/
[8] https://www.papertrail.com/
[9] https://www.loggly.com/
[10] https://www.logdna.com/
[11] https://www.splunk.com/

data possible should be stored for the shortest amount of time; and access to the log data should be restricted.

C10: Handle All Errors and Exceptions

Unhandled errors are like gold to an attacker; especially those that display verbose information. Development teams should limit the data shown in the event an unhandled error occurs, but they should also make every effort to handle all errors and exceptions using Try/Catch handlers, or whatever mechanism their coding language uses. Catching all errors and exceptions, logging them for review by the development team, and then offering a user-friendly message on what to do next is the best way to handle errors. Most honest users do not care about what error occurred, they just want to know what they should do to fix it or to move on.

Development teams creating Microsoft DotNet applications running on IIS can modify the Web.config file to turn the customErrors attribute on (Microsoft, 2020). This will cause a custom page to be displayed in the event an unhandled error occurs, preventing the actual error information from being displayed on the client. Because this is a simple configuration change, it allows the development team to have verbose errors display on the development environment, to assist in debugging, but hide the information on the production environment.

Conclusion

Implementing the *OWASP Top Ten Proactive Controls* will not prevent all attacks all of the time. However, implementing these controls will help a development team start building security into their applications early in the development process. The worst thing a development team can do is to not think about security until they get penetration test findings. By then, it could already be too late to prevent an attack. Developing quality code means developing secure code. It is important for every member of the development team to take application security seriously and do their part to protect their application from attacks.

References

Anis, Y. (2012, Aug 20). *SQL Server: Frequently Used Ports*. Retrieved from Microsoft TechNet: https://social.technet.microsoft.com/wiki/contents/articles/13106.sql-server-frequently-used-ports.aspx?

Batteau, A. W. (2011). Creating a Culture of Enterprise Cybersecurity. *International Journal of Business Anthropology, 2*, 36-47. Retrieved from https://www.researchgate.net/profile/Allen-Batteau/publication/266068991_Creating_a_Culture_of_Enterprise_Cybersecurity/links/56b2815308aed7ba3fede925/Creating-a-Culture-of-Enterprise-Cybersecurity.pdf

Boote, J. (2020, July 22). *Are you making software security a requirement?* Retrieved from Synopsis: https://www.synopsys.com/blogs/software-security/software-security-requirements/

Carielli, S. (2020). *The State Of Application Security, 2021*. Cambridge, MA: Forrester. Retrieved from https://www.forrester.com/report/The+State+Of+Application+Security+2021/-/E-RES164041

Dawson, H. (2019, June 27). *The Most Influential Security Frameworks of All Time*. Retrieved from Info Security Group: https://www.infosecurity-magazine.com/opinions/most-influential-frameworks-1-1-1/

Gentry, S. (2021, 03 02). *Access Control: Models and Methods [updated 2021]*. Retrieved from Infosec Institute: https://resources.infosecinstitute.com/certification/access-control-models-and-methods/

Granneman, J. (2019, May). *Top 7 IT security frameworks and standards explained*. Retrieved from TargetTech: https://searchsecurity.techtarget.com/tip/IT-security-frameworks-and-standards-Choosing-the-right-one

IBM. (2020). *Cost of a Data Breach Report 2020.* IBM.

Microsoft. (2017 A, 03 14). *SQL Server Browser Service*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/tools/configuration-manager/sql-server-browser-service?view=sql-server-ver15

Microsoft. (2017 B, 03 14). *Choose an Authentication Mode*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/relational-databases/security/choose-an-authentication-mode?view=sql-server-ver15

Microsoft. (2019 A, 01 19). *Default SQL Server Network Protocol Configuration*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/default-sql-server-network-protocol-configuration

Microsoft. (2019 B, 05 09). *Transparent Data Encryption (TDE)*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver15

Microsoft. (2019 C, 121 01). *xp_cmdshell (Transact-SQL)*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-ver15

Microsoft. (2020, 07 28). *Create custom error reporting pages in ASP.NET by using Visual Basic .NET*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/troubleshoot/aspnet/custom-error-reporting-page

Microsoft. (2021 A, 03 26). *Editions and supported features of SQL Server 2019 (15.x)*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-version-15?view=sql-server-ver15

Microsoft. (2021 B). *KB4518398 - SQL Server 2019 build versions*. Retrieved from Microsoft Support: https://support.microsoft.com/en-us/topic/kb4518398-sql-server-2019-build-versions-782ed548-1cd8-b5c3-a566-8b4f9e20293a

Microsoft. (2021 C, 03 31). *Using encryption*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/sql/connect/jdbc/using-ssl-encryption?view=sql-server-ver15

Microsoft Support. (n.d.). *What is: Multifactor Authentication*. Retrieved from Microsoft Support: https://support.microsoft.com/en-us/topic/what-is-multifactor-authentication-e5e39437-121c-be60-d123-eda06bddf661

Mutune, G. (n.d.). *23 Top Cybersecurity Frameworks*. Retrieved from CyberExperts.com: https://cyberexperts.com/cybersecurity-frameworks/

NIST. (2021). *Search Results for React*. Retrieved from National Vulnerability Database: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=react&search_type=all

OpenSAMM. (2020). *OpenSAMM V2.0.* Retrieved from OpenSAMM.org: https://github.com/OWASP/samm/raw/master/Supporting%20Resources/v2.0/OWASP-SAMM-v2.0.pdf

Oracle. (n.d.). *2 Introduction to Transparent Data Encryption*. Retrieved from Oracle Help Center: https://docs.oracle.com/database/121/ASOAG/introduction-to-transparent-data-encryption.htm

OWASP. (2014 A). *OWASP*. Retrieved from OWASP Mobile Top 10: https://owasp.org/www-project-mobile-top-10/

OWASP. (2014 B). *OWASP Proactive Controls 2014.* Retrieved from Archive.org: https://web.archive.org/web/20150908052859/https://www.owasp.org/images/0/07/OWASP_Proactive_Controls_v1.pdf

OWASP. (2016). *OWASP Proactive Controls 2016.* Retrieved from OWASP: https://owasp.org/www-pdf-archive/OWASP_Proactive_Controls_2.pdf

OWASP. (2017). *OWASP Top 10 Risks.* Retrieved from https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP. (2018). *OWASP Proactive Controls for Developers.* Open Web Application Security Project. Retrieved from https://owasp.org/www-pdf-archive/OWASP_Top_10_Proactive_Controls_V3.pdf

OWASP. (2020, October). *ASVS.* Retrieved from OWASP: https://github.com/OWASP/ASVS/raw/v4.0.2/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.2-en.pdf

OWASP. (2021). *Database Security Cheat Sheet.* Retrieved from OWASP Cheat Sheet Series: https://cheatsheetseries.owasp.org/cheatsheets/Database_Security_Cheat_Sheet.html#database-configuration-and-hardening

Proserveit. (2020, June 10). *Information Security Requirements: Your Obligations & Considerations*. Retrieved from Proserveit: https://www.proserveit.com/blog/information-security-requirements

Raynovich, S. (2020, 09). *Data Security Regulations – an Overview*. Retrieved from SDX Central: https://www.sdxcentral.com/security/definitions/data-security-regulations/

Scott, S. (2021). Secure Coding in Large Enterprises: Does Application Security Coaching, Training, and Consulting Increase a Development Team's Ability to Deliver Secure Code. *Secure360.* Minneapolis, MN.

Sebastian, S. (2021, 03 19). *Regexploit – Regular Expression Denial of Service*. Retrieved from ReconShell: https://reconshell.com/regexploit-regular-expression-denial-of-service/

Seiersen, R. (2021, Jan 4). *A Modern Shift-Left Security Approach*. Retrieved from Forbes: https://www.forbes.com/sites/forbestechcouncil/2021/01/04/a-modern-shift-left-security-approach/

Synopsis. (2020). *BUILDING SECURITY IN MATURITY MODEL (BSIMM) – VERSION 11.* Synopsis.

Veracode. (2020). *State of Software Security v11.* Burlington, MA: Veracode.

Wild, J. (2018, March 28). *Five Most Common Security Frameworks Explained*. Retrieved from Origin Security: https://originit.co.nz/the-strongroom/five-most-common-security-frameworks-explained/

Tables

OWASP Proactive Control Version Comparison

2018	2016	2014
C01: Define Security Requirements	C01: Verify for Security Early and Often	C01: Parameterize Queries
C02: Leverage Security Frameworks and Libraries	C02: Parameterize Queries	C02: Encode Data
C03: Secure Database Access	C03: Encode Data	C03: Validate All Inputs
C04: Encode and Escape Data	C04: Validate All Inputs	C04: Implement Appropriate Access Controls
C05: Validate All Inputs	C05: Implement Identity and Authentication Controls	C05: Establish Identity and Authentication Controls
C06: Implement Digital Identity	C06: Implement Appropriate Access Controls	C06: Protect Data and Privacy
C07: Enforce Access Controls	C07: Protect Data	C07: Implement Logging, Error Handling, and Intrusion Detection
C08: Protect Data Everywhere	C08: Implement Logging and Intrusion Detection	C08: Leverage Security Features of Frameworks and Security Libraries
C09: Implement Security Logging and Monitoring	C09: Leverage Security Frameworks and Libraries	C09: Include Security Specific Requirements
C10: Handle All Errors and Exceptions	C10: Error and Exception Handling	C10: Design and Architect Security In

Table 1 OWASP Proactive Control Version Comparison

Risk and Control Cross Reference

The following table maps the proactive controls with the risks they mitigate. Note that the OWASP Top Ten Proactive Controls guide is not always explicit about which risks each control helps protect against. The data in the following table has some level of subjectivity to it.

		Risks									
		A01	A02	A03	A04	A05	A06	A07	A08	A09	A10
Controls	C01	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	C02	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	C03	✓		✓			✓				
	C04	✓		✓				✓			
	C05	✓	✓								
	C06				✓			✓			
	C07						✓				
	C08	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	C09	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	C10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2 Risk and Control Cross Reference

Security Framework List

The following list of security frameworks was published by CyberExperts.com (Mutune, n.d.). It gives a brief description for each of the listed frameworks.

#	Framework Name	Brief Description
1	ISO IEC 27001/ISO 2700212	Identifies 114 controls in 14 categories
2	NIST Cybersecurity Framework	Describes 5 functions
3	IASME Governance	Similar to ISO27001
4	SOC 2	61 compliance requirements
5	CIS v7	Lists 20 security requirements
6	NIST 800-53 Framework	Designed for use by federal agencies
7	COBIT	Integrates IT security, governance, and management.
8	COSO	5 categories with 17 requirements
9	TC CYBER	telecommunication standards within European zones
10	HITRUST CSF	Created by alliance of health industry companies
11	CISQ	Based upon OWASP, SANS, and CWE
12	Ten Steps to Cybersecurity	Created by UK department for business
13	FedRAMP	Risk framework for federal agencies
14	HIPAA	U.S. regulation on use of personal health information
15	GDPR	EU regulation
16	FISMA	cybersecurity framework for federal agencies
17	NY DFS	New York Department of Financial Services
18	NERC CIP	Nine standards with 45 requirements
19	SCAP	Communication between security products and tools
20	ANSI	Used for industrial automation and control systems
21	NIST SP 800-12	Designed for governmental agencies
22	NIST SP 800-14	Lists commonly used security principles
23	NIST SP 800-26	Lists guidelines for managing IT security