**High -24**

***A6*** *- Cleartext Password over HTTP -2*

Vega detected a form with a password input field that submits to an insecure (HTTP) target. Password values should never be sent in the clear across insecure channels. This vulnerability could result in unauthorized disclosure of passwords to passive network attackers.

The full perils of unsafe cryptography, SSL usage, and data protection are well beyond the scope of the Top 10. That said, for all sensitive data, do all of the following, at a minimum:

***A3*** *- Cross Site Scripting – 11*

Cross-site scripting (XSS) is a class of vulnerabilities affecting web applications that can result in security controls implemented in browsers being circumvented. When a browser visits a page on a website, script code originating in the website domain can access and manipulate the DOM (document object model), a representation of the page and its properties in the browser. Script code from another website can not. This is known as the "same origin policy", a critical control in the browser security model. Cross-site scripting vulnerabilities occur when a lack of input validation permits users to inject script code into the target website such that it runs in the browser of another user who is visiting the same website. This would circumvent the browser same-origin policy because the browser has no way to distinguish authentic script code from inauthentic, apart from its origin.

***A1*** *- SQL Injection – 7*

Vega has detected a possible SQL injection vulnerability. These vulnerabilities are present when externally-supplied input is used to construct a SQL query. If precautions are not taken, the externally-supplied input (usually a GET or POST parameter) can modify the query string such that it performs unintented actions. These actions include gaining unauthorized read or write access to the data stored in the database, as well as modifying the logic of the application.

***A3*** *- Remote File Include – 1*

Vega has determined that content from a client-specified location is being retrieved by the server and output. In some circumstances, code included in this content will be executed by the server. If this is possible, an attacker may be able to gain unauthorized access to the server. Minimally, the inclusion of content originating on a third-party server introduces the possibility of phishing or social engineering attacks.

***A1*** *-MySQL Error Detected – Possible SQL Injection – 3*

Vega has detected a SQL error string known to be output by MySQL. This can indicate a possible SQL injection vulnerability. These vulnerabilities are present when externally-supplied input is used to construct a SQL query. If precautions are not taken, the externally-supplied input (usually a GET or POST parameter) can modify the query string such that it performs unintented actions. These actions include gaining unauthorized read or write access to the data stored in the database, as well as modifying the logic of the application.

**Medium – 6**

***A1*** *- Local Filesystem Paths Found – 6*

Vega has detected a possible SQL injection vulnerability. These vulnerabilities are present when externally-supplied input is used to construct a SQL query. If precautions are not taken, the externally-supplied input (usually a GET or POST parameter) can modify the query string such that it performs unintented actions. These actions include gaining unauthorized read or write access to the data stored in the database, as well as modifying the logic of the application.

**Low – 2**

***A6*** *- Form Password Field with Autocomplete Enabled – 2*

Vega detected a form that included a password input field. The autocomplete attribute was not set to off. This may result in some browsers storing values input by users locally, where they may be retrieved by third parties.

*Blank Body Detected – 1*

Vega has detected that requesting this URI returned a blank response body.

**Recommendations:**

***A1***

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.

2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP’s ESAPI provides many of these escaping routines.

3. Positive or “white list” input validation is also recommended, but is not a complete defense as many applications require special characters in their input. If special characters are required, only approaches 1. and 2. above will make their use safe. OWASP’s ESAPI has an extensible library of white list input validation routines.

***A3***

Preventing XSS requires separation of untrusted data from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into.

2. Positive or “whitelist” input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications require special characters in their input. Such validation should, as much as possible, validate the length, characters, format, and business rules on that data before accepting the input.

3. For rich content, consider auto-sanitization libraries like OWASP’s AntiSamy or the Java HTML Sanitizer Project.

4. Consider Content Security Policy (CSP) to defend against XSS across your entire site.

***A6***

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all sensitive data at rest and in transit in a manner that defends against these threats.

2. Don’t store sensitive data unnecessarily. Discard it as soon as possible. Data you don’t have can’t be stolen.

3. Ensure strong standard algorithms and strong keys are used, and proper key management is in place. Consider using FIPS 140 validated cryptographic modules.

4. Ensure passwords are stored with an algorithm specifically designed for password protection, such as bcrypt, PBKDF2, or scrypt.

5. Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.