**Appian Vulnerability Submission Worksheet**

This worksheet was designed by Appian to help you submit vulnerabilities in alignment with our security team guidelines. Please review the guidelines [here](https://community.appian.com/support/w/kb/762/kb-1447-vulnerability-testing) and submit a completed version of this worksheet through a [Technical Support Case](https://forum.appian.com/suite/sites/myappian/page/support/).

Important Note

Appian requires **verifiable evidence such as screenshots, payloads, or any other associated proof-of-concept material as well as manual reproduction steps in order to properly validate any reported vulnerability findings.**

**Our platform is unique and lends itself to a very high volume of false positives** **from automated scans**. This is because:

* Appian endpoints are dynamically created as a user navigates through the solution, which means the sitemaps generated by automated scanners are often incorrect.
* Due to our expression language - SAIL - solutions within our platform do not wait for user submission of state changing requests. For the sake of security and increased speed, data is posted back in memory as the user is interacting with the UI and validated either during that interaction or during submission. After entry of data into a given field, the application only returns object IDs and no actual values. This workflow is not conventional, and automated scanners cannot differentiate between vulnerable and non-vulnerable components.

We have yet to find a Dynamic Application Security Testing (DAST) tool that can adequately identify vulnerabilities within our platform and the solutions built on top of it. Therefore, we require evidence and manual reproduction steps for each reported vulnerability. We appreciate your help with ensuring a streamlined submission process!

Requirements

Please copy and fill the table below for each vulnerability. All **DARK BLUE** fields are REQUIRED.

**Vulnerability 1**

| **ID Number** | *Example1* |
| --- | --- |
| **Name** | ***Reflected Cross Site Scripting on \*/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/6*** |
| **Severity Level** | ***Medium*** |
| **Description** | ***The /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/6 endpoint is vulnerable to reflected Cross Site Scripting when creating a new order*** |
| **Evidence of Validation** | ***Payload used for testing:***  ***4128 3214 0002 1999</p><script>alert("XSS POC")</script>***  ***Screenshot of successful payload injections:***   1. ***Screenshot of payload embedded in the DOM of /6:*** |
| **Steps to Reproduce** | 1. ***Log in to the Webgoat application as any existing user or register a new user*** 2. ***Click on the “A3. Injection” menu option and select “Cross Site Scripting”*** 3. ***Click on the 7 at the top of the screen*** 4. ***In the “Enter your credit card number:” field, enter “4128 3214 0002 1999</p><script>alert("XSS POC")</script>”*** 5. ***Click “Purchase”*** 6. ***Observe the javascript alert pop up with the text “XSS POC” displayed*** |

**Vulnerability 2**

| **ID Number** | *Example2* |
| --- | --- |
| **Name** | ***SQL Injection found on \*/WebGoat/start.mvc#lesson/SqlInjectionAdvanced.lesson/2*** |
| **Severity Level** | ***High*** |
| **Description** | ***It was found that the /WebGoat/start.mvc#lesson/SqlInjectionAdvanced.lesson/2 in the Web Goat application is vulnerable to SQL Injection. This vulnerability leads to complete disclosure of the application database including: usernames, passwords, credit information, and other PII.*** |
| **Evidence of Validation** | ***POC Payload Used: a' OR 1=1 UNION select userid as ui, user\_name, password, cookie, NULL AS "test", NULL AS "tess", NULL AS "monkey" from user\_system\_data --***   1. ***BurpSuite being used to capture the vulnerable request:***      1. ***BurpSuite being used to exploit the vulnerability leaking all data in the application:*** |
| **Steps to Reproduce** | 1. ***Log in to the Webgoat application as any existing user or register a new user*** 2. ***Click on the “3. Injection” menu option and select “SQL Injection (Advanced)”*** 3. ***Click on the 3 at the top of the screen*** 4. ***Prepare BurpSuite or any other similar traffic proxy to capture traffic*** 5. ***Put any data into the “Name” field and click “Get Account Info”*** 6. ***Locate the POST request to “/WebGoat/SqlInjectionAdvanced/attack6a” that was just sent*** 7. ***Observe the query in the response as seen in image “1” above*** 8. ***Send the request to BurpSuite repeater*** 9. ***Replace the value of the parameter “userid\_6a” with the PoC payload and send the request*** 10. ***Observe the complete contents of the table as displayed in image “2”*** |