Getting Started with IAST and RASP using Contrast Security

Houston Java Users Group

May 29, 2019

Overview

This lab will provide a basic introduction to IAST and RASP using the Contrast Security Community Edition, showing how the platform works against known vulnerable locations in WebGoat. By observing how Contrast operates against known vulnerable locations with known payloads, users can gain the experience that will them use Contrast effectively against other applications.

This lab will walk through a couple of basic use cases:

- Contrast Assess – IAST which continuously monitors for vulnerabilities based on normal interactions with your application
- Contrast Protect – RASP which continuously monitors for attacks against your applications, and can block them

Note: This lab uses the Contrast Security Community Edition, which provides one free license for use against one Java application.

Setting Up Contrast with WebGoat

Prerequisites

Please have a 64 bit Windows or Linux environment with Java JDK 1.8 and Internet access. The environment should be suitable sized for running a web application.

A web browser such as Chrome or Firefox will also be used to access the WebGoat and Contrast UI's.

Optional: Docker and git

Procedure Using Docker

Visit https://www.contrastsecurity.com/contrast-community-edition and set up a free Community Edition instance of our product. This should be a straightforward task but please feel free to reach out if you have questions. **Note: you will need to use a work email address for your account, or you can use a temporary email – you can generate one at sites such as** https://temp-mail.org

1. Log in to your Community Edition account, accept the EULA, and visit your **Your Account** page. Examine the keys in the **Organization Keys** area.

2. In your work environment, create a working directory called WebGoatHJUG and cd into it:

   % mkdir ./WebGoatHJUG

   % cd ./WebGoatHJUG

3. Download the Dockerfile from https://github.com/rstatsinger/hjugmay2019 by either cloning the repository using **git** or simply download the raw file from the repository

4. Create a backup copy of the Dockerfile, then edit the original and set the **OrgID, Auth, and APIKey** variables from the corresponding keys in your **Your Account** page

5. Build the docker image: docker build `pwd` -t hjugdockerwebgoat

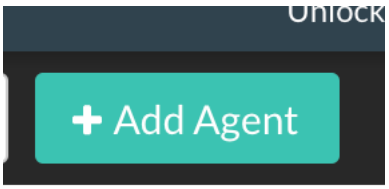
6. Run your docker container: docker run -p 8080:8080 -t hjugdockerwebgoat

7. Wait for startup to complete, then open another tab in your browser (side by side with your Contrast Security CE UI) and surf to http://localhost:8080/WebGoat

Procedure Using Agent Download from the Contrast UI

Visit https://www.contrastsecurity.com/contrast-community-edition and set up a free Community Edition instance of our product. This should be a straightforward task but please feel free to reach out if you have questions. **Note: you will need to use a work email address for your account, or you can use a temporary email – you can generate one at sites such as** https://temp-mail.org

1. Log in to your Community Edition account, accept the EULA, and download the Java agent (contrast.jar) by clicking the green 'Add Agent' button on the top right of the page



2. Create a directory or folder inside your environment and call it WebGoat
    1. Inside the WebGoat folder, place the following
        - contrast.jar that you just downloaded from TeamServer
        - webgoat-container-7.1-exec.jar: you can download it from https://github.com/WebGoat/WebGoat/releases/tag/7.1

    2. cd to the WebGoat folder and run the following command:

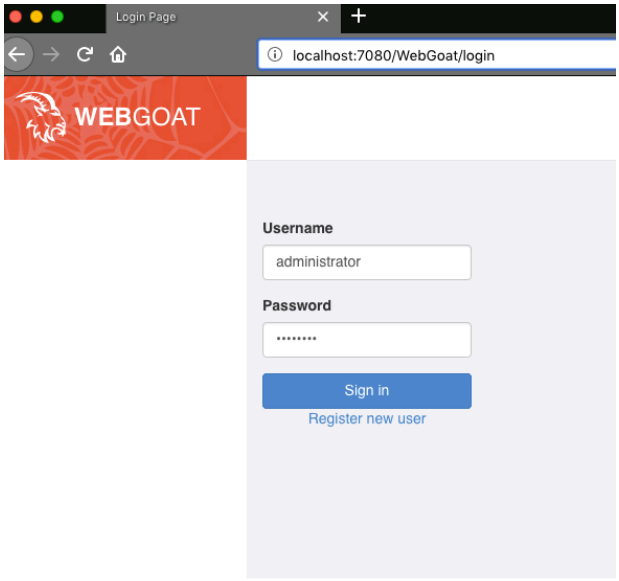
```
java -Dcontrast.app.activity.period=5000 -Dcontrast.server.activity.period=5000
-Dcontrast.standalone.appname=WebGoatHJUG -Dcontrast.server=HJUGLabServer
-javaagent:contrast.jar -jar webgoat-container-7.1-exec.jar --server.port=8080
```

(This script is available in the git repository mentioned above – feel free to clone it even if you don't use Docker)

3. Wait a minute while WebGoat starts. The console will stop scrolling - the last messages should look like this:

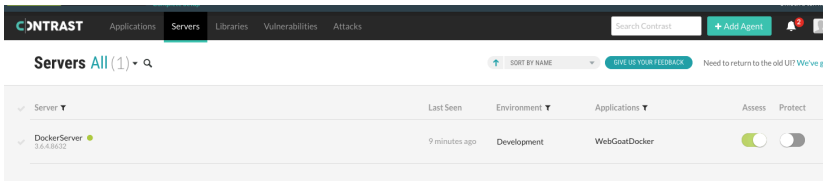
```
2019-01-09 12:26:10.120  INFO 19833 --- [          main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2019-01-09 12:26:10.138  INFO 19833 --- [          main] org.owasp.webgoat.StartWebGoat          :
Started StartWebGoat in 53.441 seconds (JVM running for 89.941)
```

4. Wait for startup to complete, then open another tab in your browser (side by side with your Contrast Security CE UI) and surf to http://localhost:8080/WebGoat



Don't log in just yet.

Within Contrast TeamServer, you should see your **DockerServer** or **HJUGLabServer** in the Servers area. Please ask me for assistance if you do not.



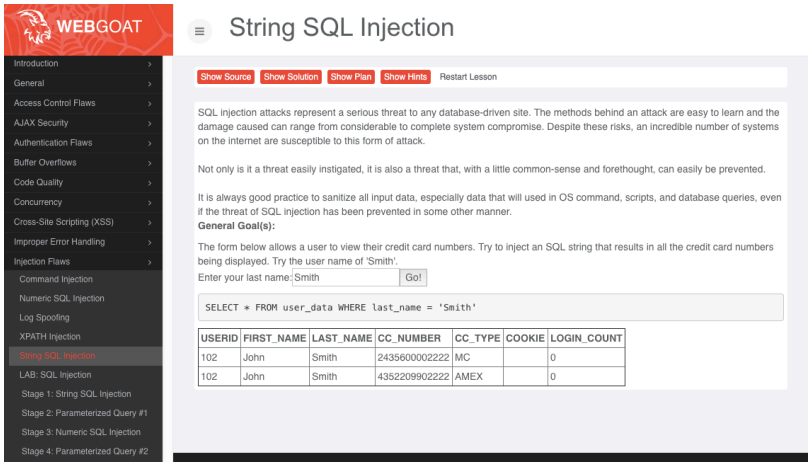
If **Protect** is enabled, slide its slider so that it looks like the above.

Lab 1: IAST Use Case: Discovering SQL Injection Vulnerabilities

This exercise will show you how IAST detects vulnerabilities in applications through simple manual interactions.

Register a new WebGoat user with any combination, such as administrator/admin123, or log in the guest or admin accounts show in the UI. Now look in the **Vulnerabilities** area of the Contrast UI. What do you see?

In the WebGoat nav area, click "String SQL Injection". In the **Enter Your Last Name** field, try the following input: Smith. (Notice that this is *not* an attack or custom script – it's a simple manual functional test in the WebGoat UI):

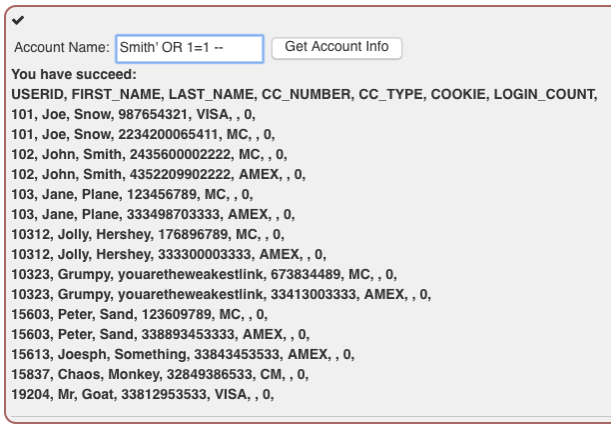


Now look in the **Vulnerabilities** area of the Contrast UI again. What do you see? Do you see more than one vulnerability? If so, why?

Class discussion: IAST discovers **vulnerable routes**, 24x7, in the background, as you write and test software.

LAB 2: Performing a SQL Injection Attack, and then Blocking it with RASP!

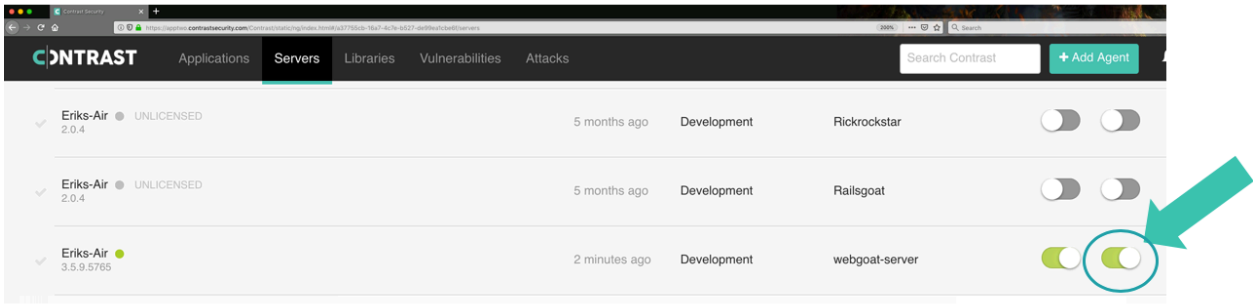
In the same field in WebGoat, try the following input: Smith' or '1'='1 This will show all users:



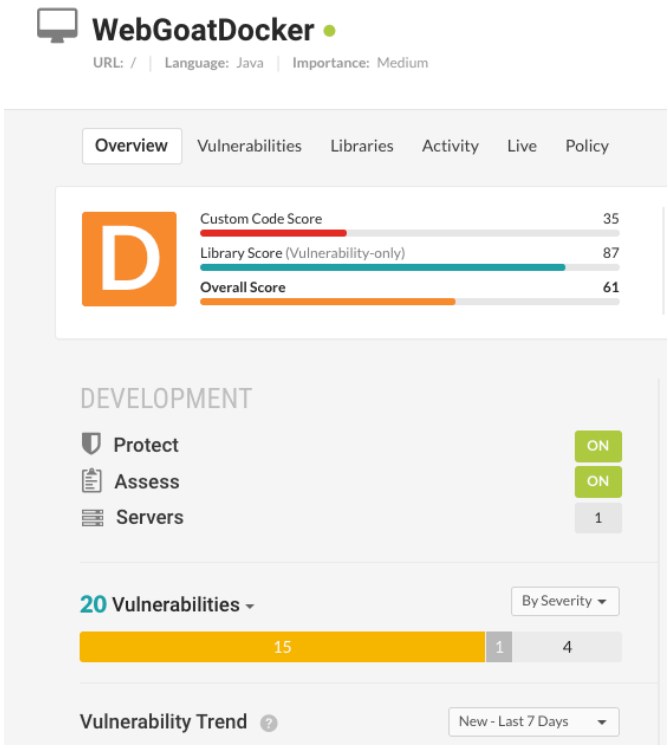
Now we'll show how Contrast Protect's RASP can detect and block such an attack.

Enabling Contrast Protect

If **Protect** is off, you can easily enable it for the server and application. On the **servers** page, find your server via its green icon and ensure both sliders are green:



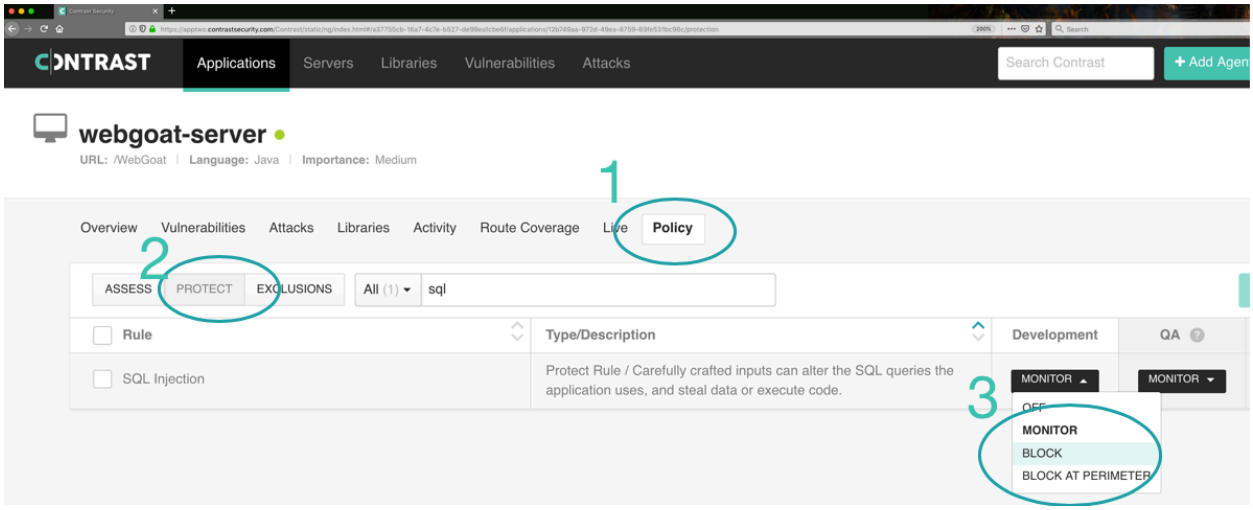
In the Applications area, make sure that Protect is turned on:



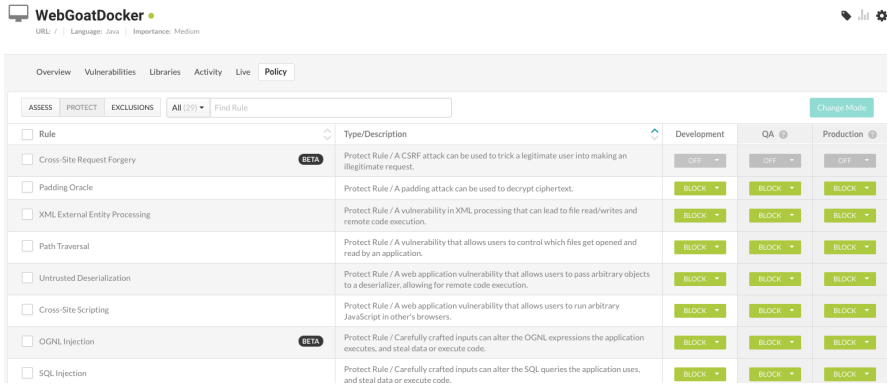
Note: one application restart may be required. This is the only time you'll ever need to restart the application – all subsequent config changes may be performed without application restart.

For Docker users, stop the container then rerun the docker run command you did earlier.

In the Contrast UI, find the **Policy** tab for your application, and click **Protect**:

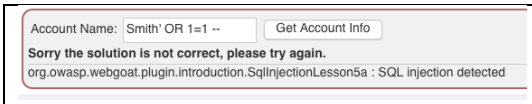
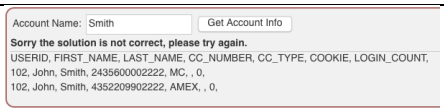


Find the SQL Injection Rule and set it to **Block:**

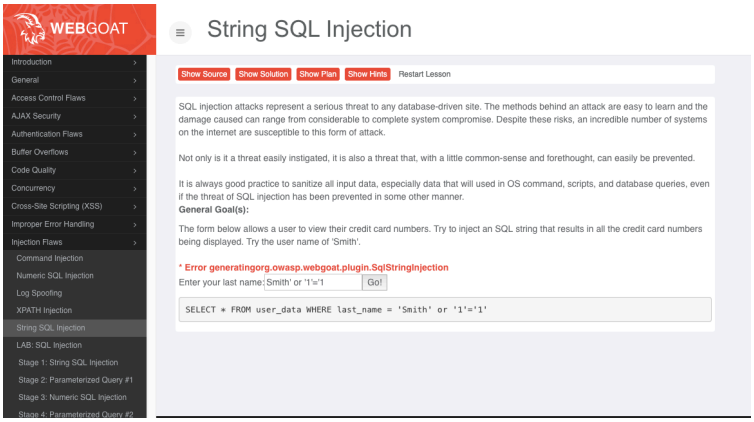


Attempt normal usage, using the name Smith again.

The normal usage works because the SQL grammar did not change:

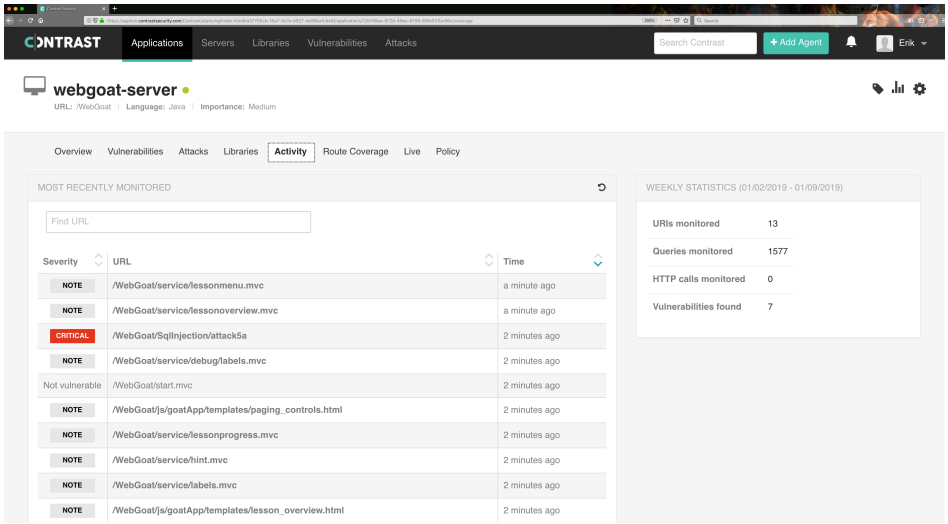
 The SQL Injection payload that previously worked will fail and be reported.	 The normal usage of last names will work fine.

Attempt the same SQL Injection payload as before. The attack no longer succeeds:



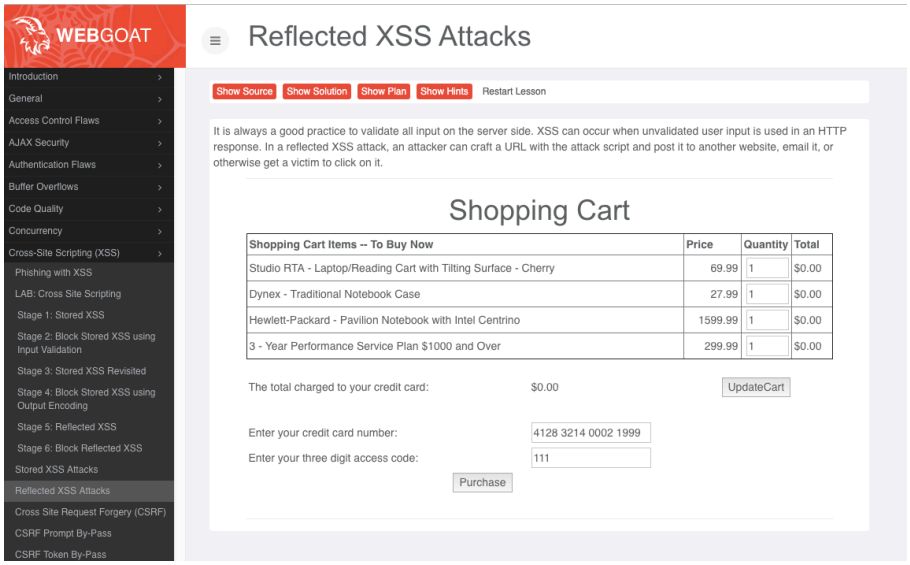
Understanding Application Usage

As you use the application for assessment, the Activity section will list URLs accessed within the application server. This helps demonstrate what Contrast is watching and can be compared against any URLs seen by the browser:



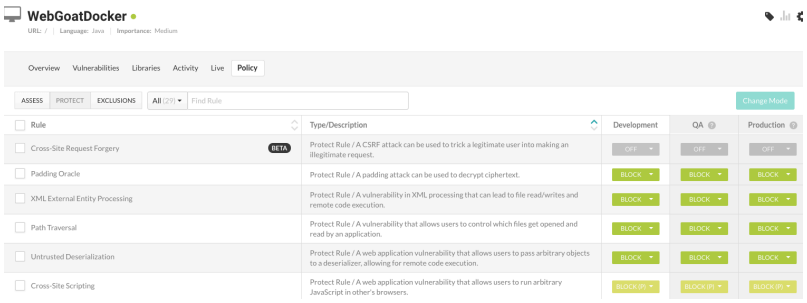
Lab 3: RASP Example: Blocking Cross Site Scripting

Cross Site Scripting enables attackers to add arbitrary JavaScript code to other users' browsing sessions, for example to steal cookie data or perform other XHR requests. To understand how Contrast protects against Cross Site Scripting, open the Cross-Site Scripting section in WebGoat then click on Reflected XSS Attacks:



In the **three digit access code** field, add an XSS payload such as: <script>alert("XSS Test")</script> Click **Purchase** and the browser will display the alert. Try running anything else between the script tags.

In the Contrast UI, set the **Protect** Policy for Cross-Site Scripting to **Block at Perimeter**. While **Block** prevents runtime exploits by generating exceptions in your application's code, **Block at Perimeter** will prevent XSS attacks before they enter the application. This will block reflected and persistent XSS, where the payload may pass to another area that technically is not exploitable but will exploit users later (an example is saving the XSS attack to a persistent database)



Try running the attack payload again.

Intelligent Reduction of False Positives

Contrast operates on something like a Hippocratic oath: first, do no harm. Developers may often store non-harmful HTML code in areas that do not perform attacks. Contrast does a grammatic analysis of content, letting format tags through while blocking script functions. For example, the following non-harmful attack will pass through the credit card field as normal:

Blah

Therefore it may be possible for attackers to mess up an HTML display but not in a harmful way.

Protect's RASP Modes

They key aspect of using Protect is ensuring that it is on and in block mode for relevant attack types. If Protect is not enabled or not set to block a certain attack type, then it will not block that attack. The block mechanisms for Protect are:

- Off – do nothing
- Monitor – watch only to notify, without interacting to take any action.
- Block – Block within the API, just before the insecure action (sink) would have taken place.
- Block at Perimeter – Block when a type of data is first detected coming in, regardless of whether it is headed for a vulnerable sink. This is most common for XSS type issues so that they do not pass through a sink that is not vulnerable to XSS and reflect back to the user.

Teams can monitor for vulnerabilities through the Vulnerabilities tab or watch attack attempts take place through the Attacks tab. Contrast TeamServer attempts to avoid noisy wolf-crying, alerting only when elements are worth looking at.

Lab 4: Analyzing OSS Risk

Click on Libraries at the top of the Contrast UI. Which libraries would you update first?