Getting Started with IAST and RASP using Contrast Security

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 few basic use cases:

- Contrast Assess – IAST which continuously discovers vulnerabilities as you write and test your applications
- Contrast Protect – RASP which continuously monitors for attacks against your applications, and can block them
- Contrast OSS – which continuously monitors your usage of Open Source Software, and assesses your risks due to OSS usage

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

Note: you should have already completed this section, but it's included here for your reference.

Prerequisites: Docker and a web browser. Optional: git

Sign up for your Free Contrast Security Community Edition Account

Visit https://www.contrastsecurity.com/contrast-community-edition and set up a free Community Edition instance of our product. **Note: you will need to use a work email address for your account.**

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.

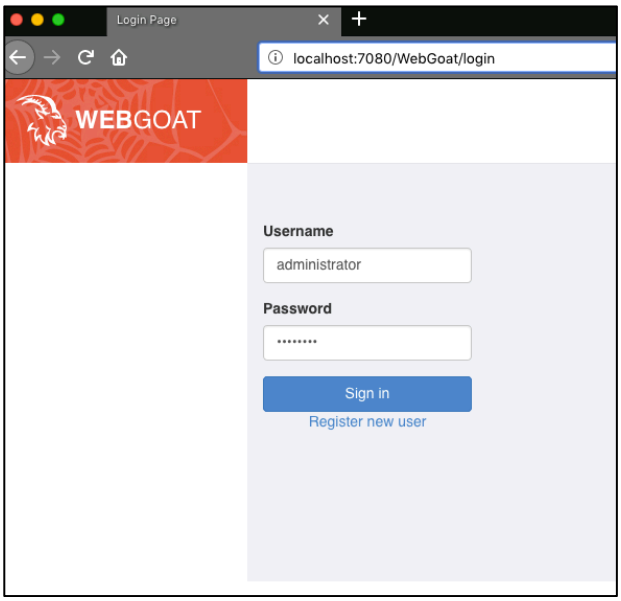
2. Download the Dockerfile from https://github.com/rstatsinger/iastraspworkshop by cloning the repository using **git,** or simply download the file from the repository.

3. 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.

4. Build the docker image: docker build `pwd` -t dockerwebgoat

5. Run your docker container: docker run -p 8080:8080 -t dockerwebgoat

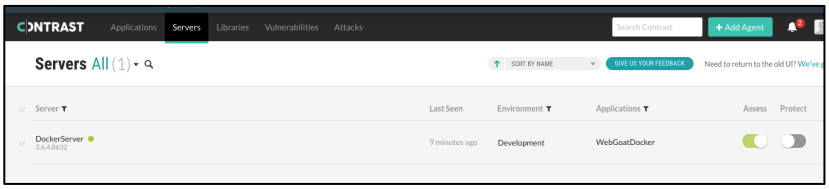
6. Wait for startup to complete, then open another tab in your browser (side by side with your Contrast Security CE UI) and surf to http://localhost:8080/WebGoat



Don't log in just yet.

Congratulations! Let's Get Started…..

Within Contrast TeamServer, you should see your **DockerServer** in the Servers area. Please raise your hand or type into the webinar chat window if you don't.

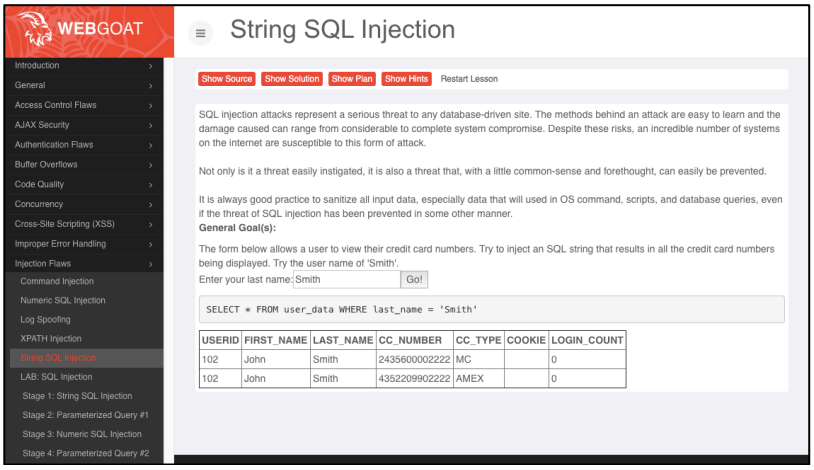


If **Protect** is enabled, slide its slider so that it looks like the above.

Lab 1: Discovering SQL Injection Vulnerability with IAST

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

Log in to Webgoat using the guest or admin accounts show in the Webgoat UI. In the WebGoat nav area, open **Injection Flaws**, then click **String SQL Injection**. In the **Enter Your Last Name** field, enter the word **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. What do you see? Do you see more than one vulnerability? If so, why?

Takeaway: IAST discovers vulnerable routes, 24x7, in the background, as you interact with software – that means manual UI usage, Selenium scripts, Postman, JMeter/Blazemeter, QA Regression, Neoload, Puppeteer, etc – IAST makes every interaction double as a security test.

LAB 2: Discovering an XML External Entity (XXE) Vulnerability with IAST

Let's exercise another vulnerable area in Webgoat. Open the **Parameter Tampering** section in WebGoat then click on **XML External Entity (XXE).** Type something into the **From:** box. Now check the **Vulnerabilities** page in the Contrast UI again. Drill into the **XML External Entity Injection (XXE) from Request Body** vulnerability and examine the **Overview**, **Details**, **HTTP Info**, **How to Fix**, **Notes**, and **Discussion** tabs.

Note: Vulnerabilities are persisted objects in Contrast, but they can be safely deleted (or marked as Not a Problem, Remediated, Fixed, etc). Don't worry if you delete a Vulnerability – you can re-

create it by restarting the Webgoat lesson and re-exercising the same route. Try it! And think about how this will help you test code fixes ☺

LAB 3: Performing a SQL Injection Attack, and then Blocking it with RASP!

Return to the **Injection Flaws** -> **String SQL Injection** lesson in Webgoat. This time, try the following input: **Smith' or '1'='1** This will show all users:

Enter your last name: Smith' or '1'='1 Go!

```
SELECT * FROM user_data WHERE last_name = 'Smith' or '1'='1'
```

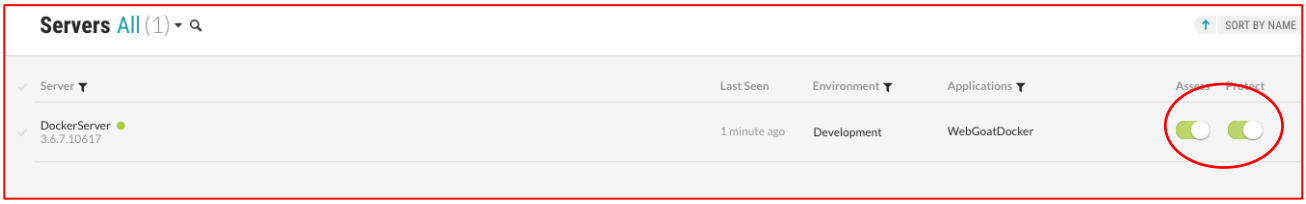
USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

Congratulations! You've just successfully performed a SQL Injection Attack, and ensured that Webgoat Inc. will make the front page of Securityboulevard.com tomorrow.

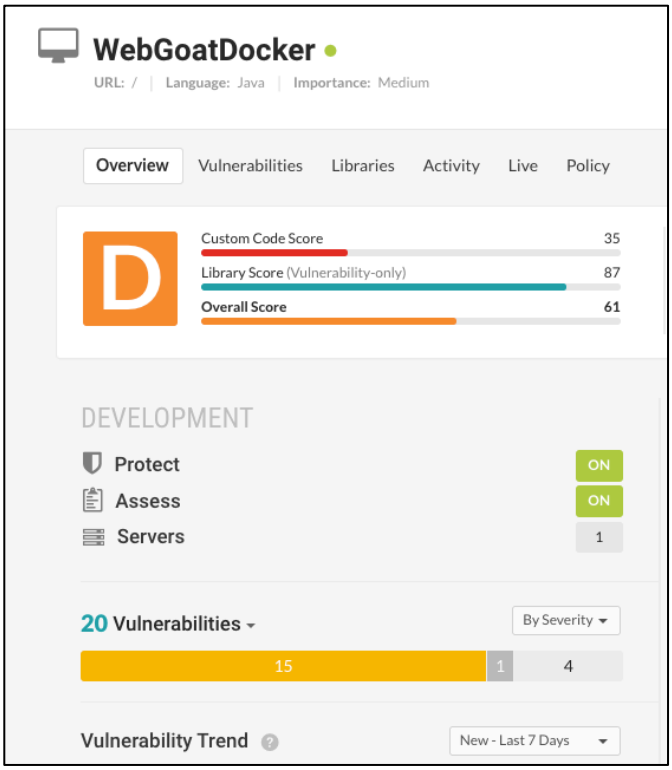
Now we'll show how Contrast Security's **Protect** RASP can detect and block such an attack.

Enabling Contrast Protect

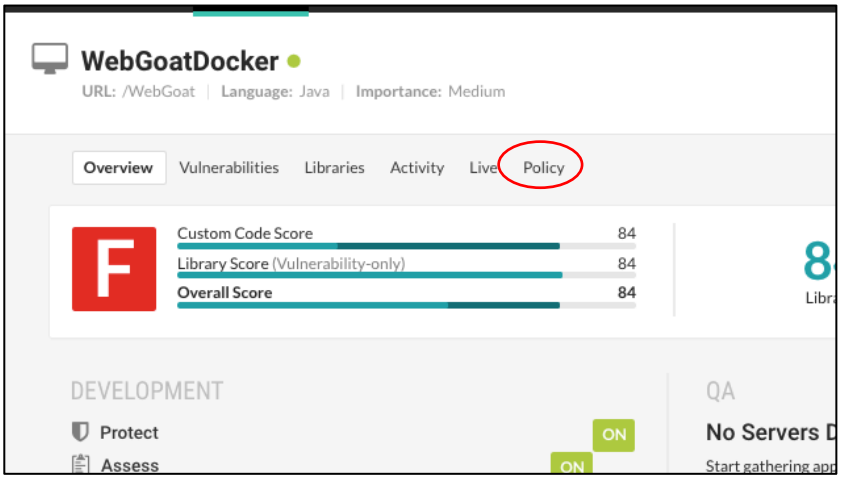
If **Protect** is off, you can easily enable it. On the **servers** page, find your server via its green icon and ensure both sliders are green:

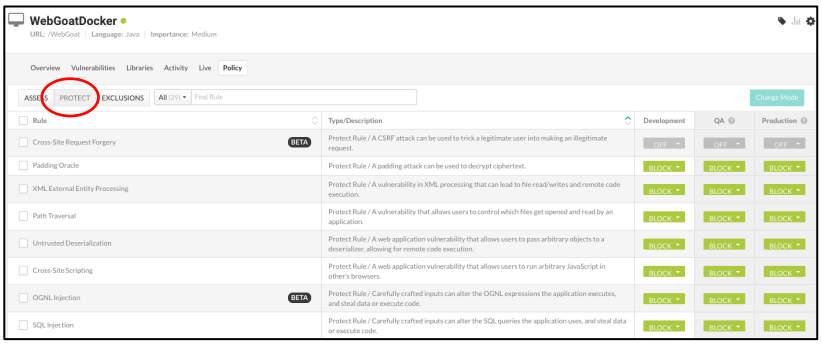


In the Applications area, make sure that **Protect** is turned on:

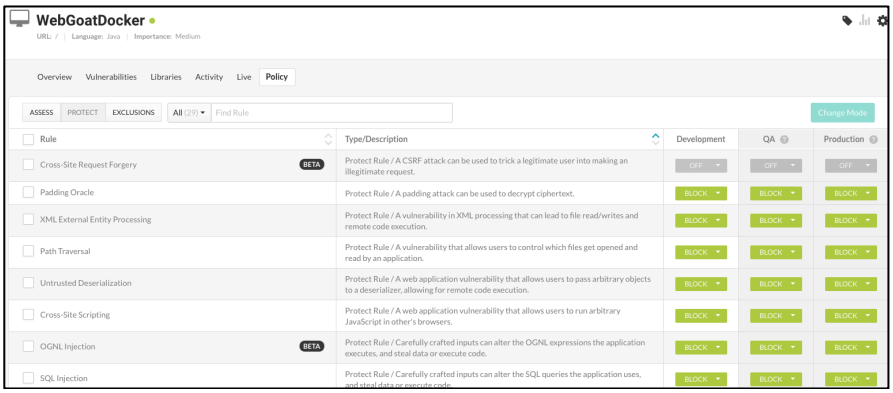


In the Contrast UI, find the **Policy** tab for your application, and click **Protect**:

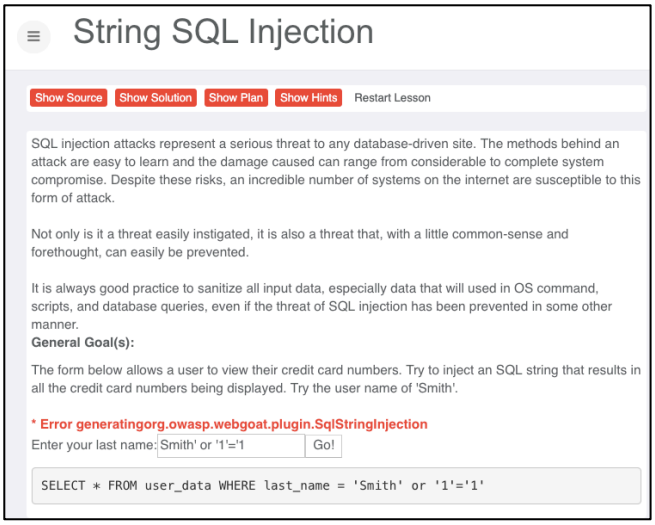




Find the SQL Injection Rule and set it to **Block** if it isn't already:



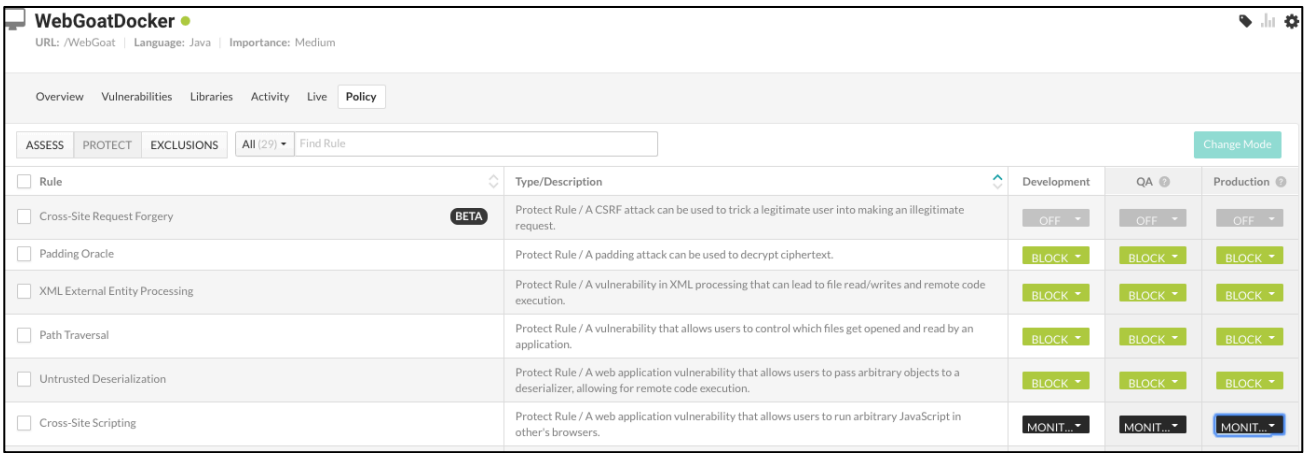
Back in Webgoat, try using just the name **Smith** again. Did this work as expected? Why? *The normal usage works because the SQL grammar did not change.* Now re-start the Webgoat lesson, and attempt the same SQL Injection attack as before. The attack no longer succeeds:



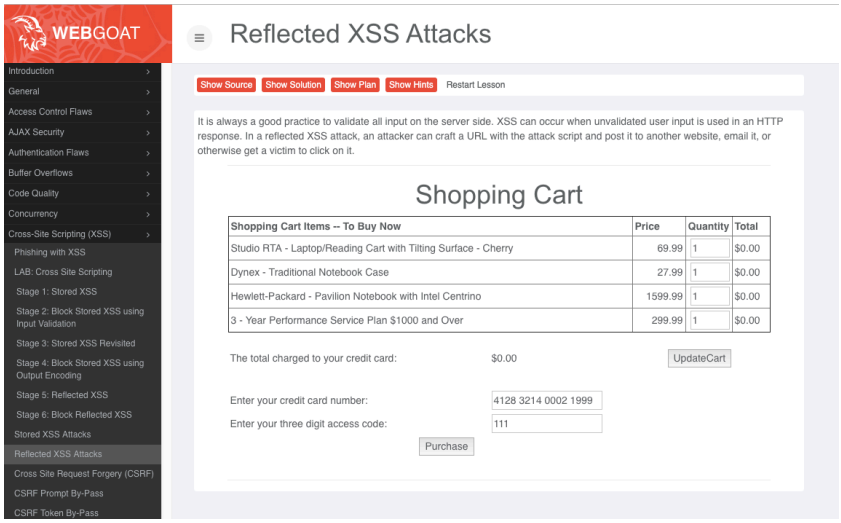
Takeaway: RASP forms a defense in depth layer inside the application which prevents exploits without interfering with non-malicious application usage, and which provides code level attack forensics.

Lab 4: 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, first set the **Protect** Policy for **Cross-Site Scripting** to **Monitor**:

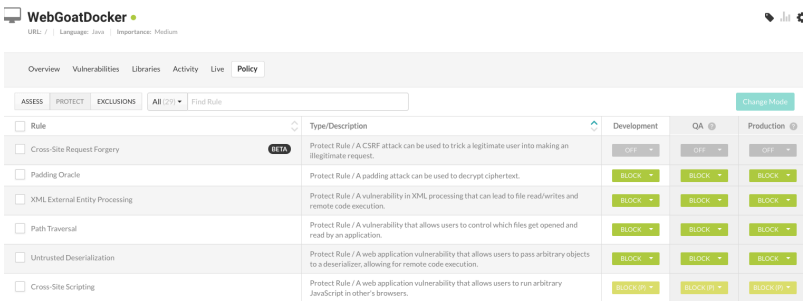


This will allow attacks to proceed but will of course alert you through your integrated notification methods (e.g. email, Slack, MS Teams, Jira, ServiceNow, etc). Now
open the **Cross-Site Scripting** section in WebGoat then click on **Reflected XSS Attacks**:



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 in a popup. Try running anything else between the script tags.

In the Contrast UI, set the **Protect** Policy for Cross-Site Scripting to **Block**.



Try running the attack payload again. What happens?

Contrast also provides a **Block at Perimeter** setting. 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 of this is saving the XSS attack to a persistent database).

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 grammatical 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.

Contrast Protect's RASP Modes

Protect provides different operation modes that enable you to handle attacks in different ways. Very often, users will start with **Protect** in **Monitor** mode until they become comfortable with its attack detection capabilities. The block operation modes for **Protect** are:

- Off – do nothing – do not monitor attempts or attempt to block them
- Monitor – watch only to notify, without interacting to take any action.
- Block – Block within the Contrast Agent, 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 5: Analyzing OSS Risk

Click on Libraries at the top of the Contrast UI. Click **Show Library Stats** at the top right. Which libraries would you update first?

Want More?

Integrations

In the dropdown under your name, click **Organization Settings**, then in the left nav area, click **Integrations.** Also check this URL: https://contrast-security-oss.github.io/

Product Documentation: https://docs.contrastsecurity.com

Blogs: https://www.contrastsecurity.com/security-influencers