## Contents

### 1 User Documentation
- 1.1 About DefectDojo ........................................... 3
- 1.2 Getting Started ............................................... 4
- 1.3 Integrations .................................................. 5
- 1.4 Models .......................................................... 17
- 1.5 Usage Examples .............................................. 19
- 1.6 Workflows ...................................................... 28
- 1.7 Upgrading ....................................................... 29
- 1.8 Running in Production ........................................ 38

### 2 Feature Documentation
- 2.1 DefectDojo Features .......................................... 45
- 2.2 Setting up Social Authentication via OAuth2 Providers .... 78

### 3 API and settings Documentation
- 3.1 DefectDojo API Documentation ............................... 87
- 3.2 DefectDojo API v2 Documentation ............................ 92
- 3.3 DefectDojo settings.py variables ............................. 96

### 4 Plugins
- 4.1 Defect Dojo Burp-Plugin ..................................... 101

### 5 Dev Documentation
- 5.1 How to write a DefectDojo parser .......................... 103
About DefectDojo

What is DefectDojo?

DefectDojo is a security tool that automates application security vulnerability management. DefectDojo streamlines the application security testing process by offering features such as importing third party security findings, merging and de-duping, integration with Jira, templating, report generation and security metrics.

What does DefectDojo do?

While traceability and metrics are the ultimate end goal, DefectDojo is a bug tracker at its core. Taking advantage of DefectDojo’s Product:Engagement model, enables traceability among multiple projects and test cycles, and allows for fine-grained reporting.

How does DefectDojo work?

DefectDojo is based on a model that allows the ultimate flexibility in your test tracking needs.

- Working in DefectDojo starts with a Product Type.
- Each Product Type can have one or more Products.
- Each Product can have one or more Engagements.
- Each Engagement can have one or more Tests.
- Each Test can have one or more Findings.
The code is open source, and available on github and a running example is available on the demo server using the credentials admin / defectdojo@demo#appsec. Note: The demo server is refreshed regularly and provisioned some sample data.

Our documentation is organized in the following sections:

- User Documentation
- Feature Documentation
- API and settings Documentation
- Plugins
- Dev Documentation
1.1 About DefectDojo

1.1.1 DefectDojo Basics

Terms

There are several terms that will be helpful to understand as you work with DefectDojo.

Products

This is the name of any project, program, team, or company that you are currently testing.

Examples

- Wordpress
- Internal wiki
- Slack

Product types

These can be business unit divisions, different offices or locations, or any other logical way of distinguishing “types” of products.

Examples

- Internal / 3rd party
- Main company / Acquisition
- San Francisco / New York offices
Engagements

Engagements are moments in time when testing is taking place. They are associated with a name for easy reference, a
time line, a lead (the user account of the main person conducting the testing), a test strategy, and a status.

Examples

- Beta
- Quarterly PCI Scan
- Release Version X

Test Types

These can be any sort of distinguishing characteristic about the type of testing that was done during an Engagement.

Examples

- Functional
- Security
- Nessus Scan
- API test

Environments

These describe the environment that was tested during a particular Engagement.

Examples

- Production
- Staging
- Stable

1.2 Getting Started

1.2.1 Docker Compose Install (recommended)

- Go to https://github.com/DefectDojo/django-DefectDojo
- Select the appropriate branch you’re working on
- Instructions in the [DOCKER.md](https://github.com/DefectDojo/django-DefectDojo/blob/master/DOCKER.md) file at the root of the repository.

1.2.2 Kubernetes

- Go to https://github.com/DefectDojo/django-DefectDojo
- Select the appropriate branch you’re working on
- Instructions in the [KUBERNETES.md](https://github.com/DefectDojo/django-DefectDojo/blob/master/KUBERNETES.md) file at the root of the repository.
1.2.3 Setup.bash Install (no longer maintained)

**Warning:** This installation method will is EOL and will be removed on 2020-12-31

- Go to https://github.com/DefectDojo/django-DefectDojo
- Select the appropriate branch you’re working on
- Under “Installation Options” click “Setup.bash”
- Follow the instructions

1.2.4 Customizing settings

See [Settings](settings-docs.rst)

1.3 Integrations

DefectDojo has the ability to import reports from other security tools.

1.3.1 Acunetix Scanner

XML format.

1.3.2 Anchore-Engine

JSON vulnerability report generated by anchore-cli tool, using a command like `anchore-cli --json image vuln <image:tag> all`

1.3.3 Aqua

JSON report format.

1.3.4 Arachni Scanner

Arachni JSON report format.

1.3.5 AppSpider (Rapid7)

Use the VulnerabilitiesSummary.xml file found in the zipped report download.

1.3.6 AWS Security Hub

The JSON output from AWS Security Hub exported with the [aws securityhub get-findings](https://docs.aws.amazon.com/cli/latest/reference/securityhub/get-findings.html) command.
1.3.7 AWS Scout2 Scanner
JS file in scout2-report/inc-awsconfig/aws_config.js.

1.3.8 AWS Prowler Scanner
Prowler file can be imported as a CSV file (-M csv).

1.3.9 Bandit
JSON report format

1.3.10 Blackduck Hub
2 options: * Import the zip file as can be created by Blackduck export. The zip file must contain the security.csv and files.csv in order to produce findings that bear file locations information. * Import a single security.csv file. Findings will not have any file location information.

1.3.11 Brakeman Scan
Import Brakeman Scanner findings in JSON format.

1.3.12 Bugcrowd
Import Bugcrowd results in CSV format.

1.3.13 Bundler-Audit
Import the text output generated with bundle-audit check

1.3.14 Burp XML
When the Burp report is generated, the recommended option is Base64 encoding both the request and response fields - e.g. check the box that says “Base64-encode requests and responses”. These fields will be processed and made available in the ‘Finding View’ page.

1.3.15 Burp Enterprise Scan
Import HTML reports from Burp Enterprise Edition

1.3.16 CCVS Report
Import JSON reports from [CCVS API](https://github.com/William-Hill-Online/CCVS-API)
1.3.17 Checkov Report

Import JSON reports of Infrastructure as Code vulnerabilities.

1.3.18 Clair Scan

Import JSON reports of Docker image vulnerabilities.

1.3.19 Clair Klar Scan

Import JSON reports of Docker image vulnerabilities from clair klar client.

1.3.20 Cobalt.io Scan

CSV Report

1.3.21 Crashtest Security

Import JSON Report Import XML Report in JUnit Format

1.3.22 Contrast Scanner

CSV Report

1.3.23 Checkmarx

Detailed XML Report

1.3.24 Choctaw Hog parser

From: https://github.com/newrelic/rusty-hog Import the JSON output.

1.3.25 DawnScanner

Import report in JSON generated with -j option

1.3.26 Dependency Check

OWASP Dependency Check output can be imported in Xml format.

1.3.27 Dependency Track

The Finding Packaging Format (FPF) from OWASP Dependency Track can be imported in JSON format.
See here for more info on this JSON format: https://docs.dependencytrack.org/integrations/file-formats/
1.3.28 DrHeader
Import of JSON report from https://github.com/Santandersecurityresearch/DrHeader

1.3.29 ESLint
ESLint Json report format (-f json)

1.3.30 Fortify
Import Findings from XML file format.

1.3.31 Generic Findings Import
Import Generic findings in CSV format.

1.3.32 Hadolint
Hadolint Dockerfile scan in json format.

1.3.33 Harbor Vulnerability
Import findings from Harbor registry container scan: https://github.com/goharbor/harbor

1.3.34 JFrogXRay
Import the JSON format for the “Security Export” file.

1.3.35 Gosec Scanner
Import Gosec Scanner findings in JSON format.

1.3.36 Gitleaks
Import Gitleaks findings in JSON format.

1.3.37 GitLab SAST Report
Import SAST Report vulnerabilities in JSON format: https://docs.gitlab.com/ee/user/application_security/sast/#reports-json-format

1.3.38 GitLab Dependency Scanning Report
Import Dependency Scanning Report vulnerabilities in JSON format: https://docs.gitlab.com/ee/user/application_security/dependency_scanning/#reports-json-format
1.3.39 Github Vulnerability


Github v4 graphql query to fetch data:

```graphql
query getVulnerabilitiesByOwner($owner: String!) {
  search(query: $owner, type: REPOSITORY, first: 100) {
    nodes {
      ... on Repository {
        name
        vulnerabilityAlerts(last: 100) {
          nodes {
            id
            securityVulnerability {
              severity
              package {
                name
              }
              advisory {
                description
                summary
                identifiers {
                  type
                  value
                }
                references {
                  url
                }
              }
            }
          }
        }
      }
    }
  }
}
```

1.3.40 HuskyCI Report

Import JSON reports from [HuskyCI](https://github.com/globocom/huskyCI)

1.3.41 IBM AppScan DAST

XML file from IBM App Scanner.

1.3.42 Immuniweb Scan

XML Scan Result File from Immuniweb Scan.

1.3.43 Kiuwan Scanner

Import Kiuwan Scan in CSV format. Export as CSV Results on Kiuwan.
1.3.44 kube-bench Scanner
Import JSON reports of Kubernetes CIS benchmark scans.

1.3.45 Microfocus Webinspect Scanner
Import XML report

1.3.46 MobSF Scanner
Export a JSON file using the API, api/v1/report_json.

1.3.47 Mozilla Observatory Scanner
Import JSON report.

1.3.48 Nessus (Tenable)
Reports can be imported in the CSV, and .nessus (XML) report formats.

1.3.49 Netsparker
Vulnerabilities List - JSON report

1.3.50 Nexpose XML 2.0 (Rapid7)
Use the full XML export template from Nexpose.

1.3.51 Nikto
XML output

1.3.52 Nmap
XML output (use -oX)

1.3.53 Node JS Scan
Node JS Scan output file can be imported in JSON format.

1.3.54 Node Security Platform
Node Security Platform (NSP) output file can be imported in JSON format.
1.3.55 NPM Audit

Node Package Manager (NPM) Audit plugin output file can be imported in JSON format. Only imports the ‘advisories’ subtree.

1.3.56 Openscap Vulnerability Scan

Import Openscap Vulnerability Scan in XML formats.

1.3.57 OpenVAS CSV

Import OpenVAS Scan in CSV format. Export as CSV Results on OpenVAS.

1.3.58 OssIndex Devaudit

Import JSON formatted output from [OSSIndex Devaudit](https://github.com/sonatype-nexus-community/DevAudit).

1.3.59 Oss Review Toolkit


1.3.60 PHP Security Audit v2

Import PHP Security Audit v2 Scan in JSON format.

1.3.61 PHP Symfony Security Checker

Import results from the PHP Symfony Security Checker.

1.3.62 Probely

Synchronize Probely Plus findings with DefectDojo.

To setup this integration set the DefectDojo URL and API key on the Integrations page on Probely. Then, select which Product, Engagement, and, optionally, the Test you want to synchronize to. The API key needs to belong to a staff user.

Works with DefectDojo 1.5.x and 1.6.x. Probely also supports non-public DefectDojo instances.


1.3.63 Qualys Scan

Qualys output files can be imported in API XML format. Qualys output files can be imported in WebGUI XML format.
1.3.64 Qualys Webapp Scan

Qualys WebScan output files can be imported in XML format.

1.3.65 Retire.js

Retire.js JavaScript scan (–js) output file can be imported in JSON format.

1.3.66 Risk Recon API Importer

Import findings from Risk Recon via the API. Configure your own JSON report as follows

```json
{
    "url_endpoint": "https://api.riskrecon.com/v1",
    "api_key": "you-api-key",
    "companies": [
    {
      "name": "Company 1",
      "filters": {
        "domain_name": [],
        "ip_address": ["127.0.0.1"],
        "host_name": ["localhost"],
        "asset_value": [],
        "severity": ["critical", "high"],
        "priority": [],
        "hosting_provider": [],
        "country_name": []
      }
    },
    {
      "name": "Company 2",
      "filters": {
        "ip_address": ["0.0.0.0"]
      }
    }
    ],
    "filters": {
      "domain_name": [],
      "ip_address": [],
      "host_name": [],
      "asset_value": [],
      "severity": ["critical"],
      "priority": [],
      "hosting_provider": [],
      "country_name": []
    }
}
```

- More than one company finding list can be queried with it’s own set of filters. Company 1 shows all available filters, while Company 2 shows that empty filters need not be present.
- To query all companies in your Risk Recon instance, simple remove the “companies” field entirely.
- If the “companies” field is not present, and filtering is still requested, the “filters” field can be used to filter all findings across all companies. It carries the same behavior as the company filters. The “filters” field is disregarded in the presence of the “companies” field.
• Removing both fields will allow retrieval of all findings in the Risk Recon instance.

1.3.67 Safety Scan

Safety scan (--json) output file can be imported in JSON format.

1.3.68 SARIF

OASIS Static Analysis Results Interchange Format (SARIF). SARIF is supported by many tools. More details about the format here: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sarif

1.3.69 ScoutSuite

Multi-Cloud security auditing tool. It uses APIs exposed by cloud providers. Scan results are located at scan-reports/scoutsuite-results/scoutsuite_*.json files. Multiple scans will create multiple files if they are running against different Cloud projects. See https://github.com/nccgroup/ScoutSuite

1.3.70 SKF Scan

Output of SKF Sprint summary export.

1.3.71 Snyk

Snyk output file (snyk test --json > snyk.json) can be imported in JSON format.

1.3.72 SonarQube Scan (Aggregates findings per cwe, title, description, file_path.)

SonarQube output file can be imported in HTML format.
To generate the report, see https://github.com/soprasteria/sonar-report
Version: >= 1.1.0

1.3.73 SonarQube Scan Detailed (Import all findings from SonarQube html report.)

SonarQube output file can be imported in HTML format.
To generate the report, see https://github.com/soprasteria/sonar-report
Version: >= 1.1.0

1.3.74 SonarQube API Import

SonarQube API will be accessed to gather the report. No report file required.
Follow below steps to setup API Import:

1. Configure the SonarQube Authentication details by navigating to Configuration->Tool Configuration. Note the url should be in the formation of http://<sonarqube_hostname>/api. Select the tool type to SonarQube.

1.3. Integrations 13
2. In the Product settings fill the details for the SonarQube Project Key (Key name can be found by navigating to a specific project and selecting the value from the url http://<sonarqube_host>/dashboard?id=<key_name>

3. Once all of the above setting are made , the API Import should be able to auto import all vulnerability information from the sonarqube instance.

1.3.75 SpotBugs

XML report of textui cli.

1.3.76 Sonatype

JSON output.

1.3.77 SSL Labs

JSON Output of sslabs-scan cli.

1.3.78 Ssslcan

Import XML output of sslscan report.

1.3.79 SSlyze Scan

XML report of SSlyze version 2 scan

1.3.80 SSlyze 3 Scan (JSON)

JSON report of SSlyze version 3 scan

1.3.81 Testssl Scan

Import CSV output of testssl scan report.

1.3.82 Trivy

JSON report of trivy scanner.

1.3.83 Trufflehog

JSON Output of Trufflehog.

1.3.84 Trustwave

CSV output of Trustwave vulnerability scan.
1.3.85 Twistlock

JSON output of the `twistcli` tool. Example:

```bash
./twistcli images scan <REGISTRY/REPO:TAG> --address https://<SECURE_URL_OF_TWISTLOCK_CONSOLE> --user <USER> --details --output-file=<PATH_TO_SAVE_JSON_FILE>
```

The CSV output from the UI is now also accepted.

1.3.86 Visual Code Grepper (VCG)

VCG output can be imported in CSV or XML formats.

1.3.87 Veracode

Detailed XML Report

1.3.88 Wapiti Scan

Import XML report.

1.3.89 Whitesource Scan

Import JSON report

1.3.90 Wpscan Scanner

Import JSON report.

1.3.91 Xanitizer

Import XML findings list report, preferably with parameter ‘generateDetailsInFindingsListReport=true’.

1.3.92 Zed Attack Proxy

ZAP XML report format.

The importers analyze each report and create new Findings for each item reported. DefectDojo collapses duplicate Findings by capturing the individual hosts vulnerable.
Additionally, DefectDojo allows for re-imports of previously uploaded reports. DefectDojo will attempt to capture the deltas between the original and new import and automatically add or mitigate findings as appropriate.

Bulk import of findings can be done using a CSV file with the following column headers:

**Date::** Date of the finding in mm/dd/yyyy format.

**Title::** Title of the finding

**CweId::** Cwe identifier, must be an integer value.

**Url::** Url associated with the finding.

**Severity::** Severity of the finding. Must be one of Info, Low, Medium, High, or Critical.
Description: :: Description of the finding. Can be multiple lines if enclosed in double quotes.
Mitigation: :: Possible Mitigations for the finding. Can be multiple lines if enclosed in double quotes.
Impact: :: Detailed impact of the finding. Can be multiple lines if enclosed in double quotes.
References: :: References associated with the finding. Can be multiple lines if enclosed in double quotes.
Active: :: Indicator if the finding is active. Must be empty, True or False
Verified: :: Indicator if the finding has been verified. Must be empty, True, or False
FalsePositive: :: Indicator if the finding is a false positive. Must be True, or False.
Duplicate: :: Indicator if the finding is a duplicate. Must be True, or False.

1.4 Models

DefectDojo attempts to simplify how users interact with the system by minimizing the number of objects it defines. The definition for each as well as sample usages is below.

1.4.1 Product Types

Product types represent the top level model, these can be business unit divisions, different offices or locations, development teams, or any other logical way of distinguishing “types” of products.

- **Examples**:
  - IAM Team
  - Internal / 3rd Party
  - Main company / Acquisition
  - San Francisco / New York offices

1.4.2 Products

This is the name of any project, program, or product that you are currently testing.

- **Examples**:
  - Wordpress
  - Internal wiki
  - Slack

1.4.3 Environments

These describe the environment that was tested in a particular Test.

- **Examples**
  - Production
  - Staging
  - Stable
1.4.4 Engagements

Engagements are moments in time when testing is taking place. They are associated with a name for easy reference, a time line, a lead (the user account of the main person conducting the testing), a test strategy, and a status. Engagement consists of two types: Interactive and CI/CD. An interactive engagement is typically an engagement conducted by an engineer, where findings are usually uploaded by the engineer. A CI/CD engagement, as it’s name suggests, is for automated integration with a CI/CD pipeline.

- **Examples**
  - Beta
  - Quarterly PCI Scan
  - Release Version X

1.4.5 Test Types

These can be any sort of distinguishing characteristic about the type of testing that was done in an Engagement.

- **Examples**
  - Functional
  - Security
  - Nessus Scan
  - API test
  - Static Analysis

1.4.6 Test

Tests are a grouping of activities conducted by engineers to attempt to discover flaws in a product. Tests represent an instance of a Test Type - a moment in time when the product is being analyzed. Tests are bundled within engagements, have a start and end date and are defined by a test type.

- **Examples**
  - Burp Scan from Oct. 29, 2015 to Oct. 29, 2015
  - Nessus Scan from Oct. 31, 2015 to Oct. 31, 2015

1.4.7 Finding

A finding represents a flaw discovered while testing. It can be categorized with severities of Critical, High, Medium, Low, and Informational (Info).

- **Examples**
  - OpenSSL ‘ChangeCipherSpec’ MiTM Potential Vulnerability
  - Web Application Potentially Vulnerable to Clickjacking
  - Web Browser XSS Protection Not Enabled
1.5 Usage Examples

DefectDojo is designed to make tracking testing engagements simple and intuitive. The Models page will help you understand the terminology we use below, so we recommend taking a look at that first.

1.5.1 Create a new Product Type

The first step to using DefectDojo is to create a Product Type. Some examples might be “Mobile Apps” or “New York Office.” The idea is to make it easy to divide your Products into logical categories, based on your organizational structure, or just to divide internal and external applications.

Select “View Product Types” from the “Products” dropdown in the main menu.

Click the “New Product Type” button at the top.

Enter a name for your new Product Type.
1.5.2 Create a new Test Type

Test Types will help you differentiate the scope of your work. For instance, you might have a Performance Test Type, or a specific type of security testing that you regularly perform.

Select “Test Types” from the “Engagements” dropdown in the main menu.

Click the “New Test Type” button at the top.

Enter a name for your new Test Type.

1.5.3 Create a new Development Environment

Development Environments are for tracking distinct deployments of a particular Product. You might have one called “Local” if you deploy the Product on your own computer for testing, or “Staging” or “Production” for official deploy-
Select “Development Environments” from the “Engagements” dropdown in the main menu.

Click the “New Development Environment” button at the top.

Enter a name for your new Development Environment.

### 1.5.4 Create a new Engagement

Engagements are useful for tracking the time spent testing a Product. They are associated with a Product, a Testing Lead, and are comprised of one or more Tests that may have Findings associated with them. Engagements also show up on your calendar.
Select “Engagements” from the “Engagements” dropdown in the main menu.

Click the “New Engagement” button on the right.

Enter the details of your Engagement.

The Deduplication Level specifies whether to perform deduplication only for tests in the engagement or to perform deduplication on all tests in the product which have an engagement also on Deduplication Level product. Enabled deduplication is mandatory.

1.5.5 Adding Tests to an Engagement

From the Engagement creation page, you can add a new Test to the Engagement. You can also add a Test to the Engagement later from that Engagement’s main page. Tests are associated with a particular Test Type, a time, and an Environment.
Enter the details of your Test.

### 1.5.6 Adding Findings to a Test

Findings are the defects or interesting things that you want to keep track of when testing a Product during a Test/Engagement. Here, you can lay out the details of what went wrong, where you found it, what the impact is, and your proposed steps for mitigation. You can also reference CWEs, or add links to your own references.

Templating findings allows you to create a version of a finding that you can then re-use over and over again, on any Engagement.

Enter the details of your Finding, or click the “Add Finding from Template” button to use a templated Finding.
From the “Add Finding Template” popup, you can select finding templates from the list, or use the search bar. Templates can be used across all Engagements.

Define what kind of Finding this is. Is it a false positive? A duplicate? If you want to save this finding as a template, check the “Is template” box.

1.5.7 Accepting a Finding Risk

Findings cannot always be remediated or addressed for various reasons. A finding status can change to accepted by doing the following. Findings are accepted in the engagement view. To locate the engagement from the finding click the link to engagement as shown below.
Then, in the engagement view click the plus icon in the ‘Risk Acceptance’ box and fill in the details to support the risk acceptance.

The engagement view is now updated with the risk.

The finding status changes to ‘Accepted’ with a link to the risk acceptance.

1.5.8 Viewing an Engagement

Most of the work of an Engagement can be done from that Engagement’s main page. You can view the Test Strategy or Threat Model, modify the Engagement dates, view Tests and Findings, add Risk Acceptance, complete the security Check List, or close the Engagement.
This page lets you do most of the common tasks that are associated with an Engagement.

### 1.5.9 Tracking your Engagements in the calendar

The calendar can help you keep track of what Engagements your team is currently working on, or determine the timeline for past Engagements.

Select “Calendar” in the main menu.

Here you can view the current engagements for the month, or go back in time.
1.5.10 Tracking metrics for your Products

Tracking metrics for your Products can help you identify Products that may need additional help, or highlight a particularly effective member of your team.

You can also see the Dashboard view, a page that scrolls automatically, showing off the results of your testing. This can be useful if you want to display your team’s work in public without showing specific details.

Select “All” or a Product Type from the “Metrics” drop-down in the main menu.

Here you can see graphs of various metrics, with the ability to filter your results by time, Product Type, and severity.
At the bottom of the Metrics page, you can see granular data about your work, such as a breakdown of the most severe bugs by Product, lists of open, accepted, and closed Findings, and trends for each week, as well as the age of all current open Findings.

### 1.6 Workflows

#### 1.6.1 Example 1 - Bill the security engineer

Bill wants a place to keep track of what he’s worked on, so that he can show his boss exactly what issues he reports, and statistics about how long it takes to close them.

When he is asked to audit an application, Bill registers a new Product in DefectDojo, and creates a new Engagement. Here he sets some basic information, like how long he expects the Engagement will take, who will be leading the
testing (himself), what Product he will be working on, and what tests he will be doing.

Next, he can add a Test to the Engagement, or upload a Nessus scan and start picking out the real vulnerabilities from the false positives (Nessus scan Findings are imported as inactive by default).

Within the Test section, Bill can add Findings for any issues that he has uncovered during his audit. He can assign a severity to the Findings, describe replication steps, mitigation strategies, and impact on the system. This will come in handy when he wants to generate a report to send to the development team responsible for this Product, or his manager.

Once Bill has completed his Engagement, he can close the Engagement on the main Engagement page. He can then view the results of his Tests, and generate a report to send to the development team.

If Bill hears back from the development team that they won’t be able to fix the issue for a while, he can make a note of this on the Engagement page. Bill will also receive Alerts for any bugs that persist longer than they are supposed to based on their severity.

### 1.6.2 Example 2 - John the QE manager

John wants to keep tabs on what his team members are up to, and find issues that are taking a long time to get fixed. He creates his own DefectDojo account with superuser privileges so that he can view other team members’ metrics.

To get a better idea of what his team members are currently working on, he can start by checking the Calendar. This will show him any active Engagements that his team is involved in, based on the dates assigned to those Engagements.

He can view metrics for a Product Type, such as “Third Party Apps” to track his team’s activity and follow up with Product teams who have long-lived bugs. He can also look at all the Findings for which there is a Risk Acceptance associated, and ensure that the proper documentation or timeline has been provided for the Findings in question.

If he wants to check on a particular team member’s progress, he can look at the Engineer Metrics dashboard under “Additional Metrics” for that user.

### 1.7 Upgrading

#### 1.7.1 Docker-compose

When you deploy a vanilla docker-compose, it will create a persistent volume for your MySQL database. As long as your volume is there, you should not lose any data.

**Using docker images provided in DockerHub**

**Note:** If you’re using `latest`, then you need to pre pull the `latest` from DockerHub to update.

The generic upgrade method for docker-compose follows these steps:

- **Pull the latest version**

  ```bash
docker pull defectdojo/defectdojo-django:latest
docker pull defectdojo/defectdojo-nginx:latest
```

- If you would like to use something older (so not the latest version), specify the version (tag) you want to upgrade to:
docker pull defectdojo/defectdojo-django:1.10.2
docker pull defectdojo/defectdojo-nginx:1.10.2

- Go to the directory where your docker-compose.yml file lives
- Stop DefectDojo: docker-compose stop
- Re-start DefectDojo, allowing for container recreation: docker-compose up -d
- Run the database migrations to bring your database schema up to speed with the latest code
- If you have the initializer disabled (or if you want to be on the safe side), run the migration command: docker-compose exec uwsgi /bin/bash -c 'python manage.py migrate

Building your local images

If you build your images locally and do not use the ones from DockerHub, the instructions are much the same, except that you’d build your images first. (Of course, if you’re doing this, then you know you have to update the source code first)

Replace the first step above with this one: -docker-compose build

1.7.2 Setup.bash

Warning: This installation method will is EOL and will be removed on 2020-12-31

The easiest way to upgrade to a new version of DefectDojo is to pull from Github. Assuming the source code lives in a directory named defect-dojo you can complete the following steps to upgrade to the latest DefectDojo release:

cd defect-dojo
git checkout master
git pull
pip freeze > pip_frozen.txt
pip install -r pip_frozen.txt --upgrade
./manage.py makemigrations dojo
./manage.py makemigrations
./manage.py migrate

Because yarn assets change from time to time, it is always a good idea to re-install them and collect the static resources.

cd defect-dojo
cd components
yarn
cd ..

At this point yarn may ask you to select from different versions of packages, choose the latest on each.

Next you can run:

./manage.py collectstatic --noinput

If you are in your production system, you will need to restart gunicorn and celery to make sure the latest code is being used by both.
1.7.3 FAQ

Celery Error:
If you have an issue starting Django with the error: TypeError: config_from_object() got an unexpected keyword argument ‘namespace’
Upgrade Celery to the latest version:

```
    pip install --upgrade celery
```

1.7.4 Upgrading to DefectDojo Version 1.13.x

- See release notes: https://github.com/DefectDojo/django-DefectDojo/releases/tag/1.13.0
- Hashcode settings affecting deduplication have changed, to update existing findings run:

```
/manage.py dedupe
```
If you’re using docker:

```
docker-compose exec uwsgi /manage.py dedupe
```
This can take a while depending on your instance size. It might possible that new duplicates are detected among existing findings, so make a backup before running!

1.7.5 Upgrading to DefectDojo Version 1.12.x

- See release notes: https://github.com/DefectDojo/django-DefectDojo/releases/tag/1.12.0
- 1.12.1 is a security release https://github.com/DefectDojo/django-DefectDojo/releases/tag/1.12.1

1.7.6 Upgrading to DefectDojo Version 1.11.x

- See release notes: https://github.com/DefectDojo/django-DefectDojo/releases/tag/1.11.0
- 1.11.1 is a security release https://github.com/DefectDojo/django-DefectDojo/releases/tag/1.11.1

1.7.7 Upgrading to DefectDojo Version 1.10.x

1.10.4 is a security release
- See the security advisory: https://github.com/DefectDojo/django-DefectDojo/security/advisories/GHSA-96vq-gqr9-vf2c
- See release notes: https://github.com/DefectDojo/django-DefectDojo/releases/tag/1.10.4
- Version 1.10.4 replaces 1.10.3 as the latter contained an incomplete fix

What’s New:
- See release notes: https://github.com/DefectDojo/django-DefectDojo/releases
- A quickfix is to rename your own / customized settings.py or settings.dist.py to local_settings.py. Details of that PR: https://github.com/DefectDojo/django-DefectDojo/pull/3136
• Major JIRA integration refactoring, for which you should at least use 1.10.1 and not 1.10.0 for many bug fixes.

Breaking changes

Kubernetes/Helm users: we have moved away from the “stable” repository to “bitnami” in this release. The bitnami postgresql chart required us to add a new key to the postgresql secret, which will give you the error `postgresql-postgres-password is missing if you have createPostgresqlSecret: false`. In 1.10.1, a fix was also included to allow your existing `postgresqlPassword` to be reused properly.

Including in 1.10.1 were a couple fixes related to a rabbitMQ upgrade. The path to access password, `erlangCookie` and `existingPasswordSecret` changed from `rabbitmq` to `auth`. Furthermore, as rabbitMQ is deployed as a StatefulSet, an in-place upgrade is not possible and an error will likely be thrown such as `Forbidden: updates to statefulset spec for fields other than 'replicas', 'template', and 'updateStrategy' are forbidden`. After ensuring your rabbitMQ celery queue is empty, you will then want to delete your rabbitMQ StatefulSet and PVC to allow them to get re-created, or fully delete and recreate defectdojo.

1.7.8 Upgrading to DefectDojo Version 1.9.3

This is a security release

• See the security advisory
• See release notes

What’s New:

• See release notes: https://github.com/DefectDojo/django-DefectDojo/releases

NOTE:

When upgrading from before 1.9.2, a corrective script may need to be ran

`.manage.py create_endpoint_status`

If you’re using docker:

docker-compose exec uwsgi `.manage.py create_endpoint_status`

This can take a while depending on your hardware and the number of findings in your instance.

• Search index tweaking index rebuild after upgrade:

This requires a (one-time) rebuild of the Django-Watson search index. Execute the django command from the defect dojo installation directory:

`.manage.py buildwatson`

If you’re using docker:

docker-compose exec uwsgi `.manage.py buildwatson`

This can take a while depending on your hardware and the number of findings in your instance.

1.7.9 Upgrading to DefectDojo Version 1.8.0

What’s New:

• See release notes: https://github.com/DefectDojo/django-DefectDojo/releases
• Improved search, which requires an index rebuild (https://github.com/DefectDojo/django-DefectDojo/pull/2861)
This requires a (one-time) rebuild of the Django-Watson search index. Execute the django command from the defect dojo installation directory:

```
./manage.py buildwatson
```

If you’re using docker:

```
docker-compose exec uwsgi ./manage.py buildwatson
```

This can take a while depending on your hardware and the number of findings in your instance.

- **NOTE:**

As a result of a breaking bug revolving around Endpoint_status objects, a corrective script will need to be ran after every dynamic scan imported through either API version.

The script can be found [here](#).

```
./manage.py create_endpoint_status
```

If you’re using docker:

```
docker-compose exec uwsgi ./manage.py create_endpoint_status
```

This can take a while depending on your hardware and the number of findings in your instance.

### 1.7.10 Upgrading to DefectDojo Version 1.7.0

**What’s New:**

- Updated search, you can now search for CVE-XXXX-YYYY

This requires a (one-time) rebuild of the Django-Watson search index. Execute the django command from the defect dojo installation directory:

```
./manage.py buildwatson dojo.Finding
```

If you’re using docker:

```
docker-compose exec uwsgi ./manage.py buildwatson dojo.Finding
```

### 1.7.11 Upgrading to DefectDojo Version 1.5.0

**What’s New:**

- Updated UI with a new DefectDojo logo, default colors and CSS.
- Updated Product views with tabs for Product Overview, Metrics, Engagements, Endpoints, Benchmarks (ASVS), and Settings to make it easier to navigate and manage your products.
- Languages pie chart on product overview, only supported through the API and Django admin, integrates with clocl analyzer
- New Engagement type of CI/CD to support continual testing
- Engagement shortcuts and ability to import findings and auto-create an engagement
• Engagement labels for overdue, no tests and findings
• New Contextual menus throughout DefectDojo and shortcuts to new findings and critical findings
• Ability to merge a finding into a parent finding and either inactivate or delete the merged findings.
• Report improvements and styling adjustment with the default option of HTML reports
• SLA for remediation of severities based on finding criticality, for example critical findings remediated within 7 days. Configurable in System Settings.
• Engagement Auto-Close Days in System Settings. Automatically close an engagement if open past the end date.
• Ability to apply remediation advice based on CWE. For example XSS can be configured as a template so that it’s consistent across all findings. Enabled in system settings.
• Finding confidence field supported from scanners. First implementation in the Burp importer.
• Goast importer for static analysis of Golang products
• Celery status check on System Settings
• Beta rules framework release for modifying findings on the fly
• DefectDojo 2.0 API with Swagger support
• Created and Modified fields on all major tables
• Various bug fixes reported on Github

Upgrading to 1.5.0 requirements:

1. Back up your database first, ideally take the backup from production and test the upgrade on a staging server.
2. Edit the settings.py file which can be found in django-DefectDojo/dojo/settings/settings.py.
   Copy in the rest framework configuration after the CSRF_COOKIE_SECURE = True:

   ```python
   REST_FRAMEWORK = {
       'DEFAULT_AUTHENTICATION_CLASSES': (  
           'rest_framework.authentication.TokenAuthentication',
           'rest_framework.authentication.BasicAuthentication',
       ),
       'DEFAULT_PERMISSION_CLASSES': (  
           'rest_framework.permissions.DjangoModelPermissions',
       ),
       'DEFAULT_RENDERER_CLASSES': (  
           'rest_framework.renderers.JSONRenderer',
       ),
       'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
       'PAGE_SIZE': 25
   }
   ``

   Navigate to: LOGIN_EXEMPT_URLS and add the following after r'^%sfinding/image/(?P<token>[^/]++)$' % URL_PREFIX:
   ```python
   r'^%sfinding/image/(?P<token>[^/]++)$' % URL_PREFIX,
   r'^%sapi/v2/' % URL_PREFIX,
   ```

   Navigate to: INSTALLED_APPS and add the following after: ‘multiselectfield’:

   ```python
   'multiselectfield',
   'rest_framework',
   'rest_framework.authtoken',
   ```
Navigate to: CELERY_TASK_IGNORE_RESULT = True and add the following after CELERY_TASK_IGNORE_RESULT line:

```
CELERY_RESULT_BACKEND = 'db+sqlite:///dojo.celeryresults.sqlite'
```

Save your modified settings file. For reference the modified file should look like the new 1.5.0 [settings](https://github.com/DefectDojo/django-DefectDojo/blob/master/dojo/settings/settings.dist.py) file, minus the environmental configurations. As an alternative this file can be used and the enviromental configurations from you environment can be copied into this file.

3. Activate your virtual environment and then upgrade the requirements:

```
pip install -r requirements.txt --upgrade
```

4. Upgrade the database:

```
./manage.py makemigrations
./manage.py migrate
```

5. Collect the static files (Javascript, Images, CSS):

```
./manage.py collectstatic --noinput
```

6. Complete

### 1.7.12 Upgrading to DefectDojo Version 1.3.1

**What's New:**

- New importers for Contrast, Nikto and TruffleHog (finding secrets in git repos).
- Improved merging of findings for dynamic and static importers
- Markdown support for findings
- HTML report improvements including support of Markdown.
- System settings Celery status page to assist in debugging if Celery is functional.

**Upgrading to 1.3.1 requires:**

1. pip install markdown pip install pandas
2. ./manage.py makemigrations ./manage.py migrate
3. ./manage.py collectstatic --noinput
4. Complete

### 1.7.13 Upgrading to DefectDojo Version 1.2.9

**What's New:** New feature: Benchmarks (OWASP ASVS)

**Upgrading to 1.2.9 requires:**

1.7. Upgrading
1. ./manage.py makemigrations ./manage.py migrate ./manage.py loaddata dojo/fixtures/benchmark_type.json
   ./manage.py loaddata dojo/fixtures/benchmark_category.json ./manage.py loaddata
dojo/fixtures/benchmark_requirement.json
2. ./manage.py collectstatic –noinput
3. Complete

1.7.14 Upgrading to DefectDojo Version 1.2.8

New feature: Product Grading (Overall Product Health) Upgrading to 1.2.8 requires:
1. ./manage.py makemigrations ./manage.py migrate ./manage.py system_settings
2. ./manage.py collectstatic –noinput
3. pip install asteval
4. pip install –upgrade celery
5. Complete

1.7.15 Upgrading to DefectDojo Version 1.2.4

Upgrading to 1.2.4 requires:
1. ./manage.py makemigrations ./manage.py migrate ./manage.py loaddata dojo/fixtures/objects_review.json

1.7.16 Upgrading to DefectDojo Version 1.2.3

Upgrading to 1.2.3 requires:
1. ./manage.py makemigrations ./manage.py migrate ./manage.py loaddata dojo/fixtures/language_type.json
2. Currently languages and technologies can be updated via the API or in the admin section of Django.

1.7.17 July 6th 2017 - New location for system settings

Pull request #313 moves a number of system settings previously located in the application’s settings.py to a model that can be used and changed within the web application under “Configuration -> System Settings”.

If you’re using a custom URL_PREFIX you will need to set this in the model after upgrading by editing dojo/fixtures/system_settings.json and setting your URL prefix in the url_prefix value there. Then issue the command ./manage.py loaddata system_settings.json to load your settings into the database.

If you’re not using a custom URL_PREFIX, after upgrading simply go to the System Settings page and review which values you want to set for each setting, as they’re not automatically migrated from settings.py.

If you like you can then remove the following settings from settings.py to avoid confusion:

- ENABLE_DEDUPLICATION
- ENABLE_JIRA
- S_FINDING_SEVERITY_NAMING
- URL_PREFIX
- TIME_ZONE
1.7.18 Upgrading to DefectDojo Version 1.2.2

Upgrading to 1.2.2 requires:

1. Copying settings.py to the settings/ folder.
2. If you have supervisor scripts change DJANGO_SETTINGS_MODULE=dojo.settings.settings

1.7.19 Upgrading to Django 1.1.5

If you are upgrading an existing version of DefectDojo, you will need to run the following commands manually:

2. The following must be removed/commented out from settings.py:

   ```python
   'djangobower.finders.BowerFinder',
   From the line that contains:
   # where should bower install components
   ...
   To the end of the bower declarations
   'justgage'
   }
   
   3. The following needs to be updated in settings.py:

   ```python
   STATICFILES_DIRS = ( 
   # Put strings here, like "/home/html/static" or "C:/www/django/static". 
   # Always use forward slashes, even on Windows. 
   # Don't forget to use absolute paths, not relative paths. 
   os.path.dirname(DOJO_ROOT) + "\components\yarn_components",
   )
   ```

1.7.20 Upgrading to Django 1.11

Pull request #300 makes DefectDojo Django 1.11 ready. A fresh install of DefectDojo can be done with the setup.bash script included - no special steps are required.

If you are upgrading an existing installation of DefectDojo, you will need to run the following commands manually:

```
pip install django-tastypie --upgrade
pip install django-tastypie-swagger --upgrade
pip install django-filter --upgrade
pip install django-watson --upgrade
pip install django-polymorphic --upgrade
pip install django --upgrade
pip install pillow --upgrade
./manage.py makemigrations
./manage.py migrate
```

The following must be removed/commented out from settings.py:
The following needs to be added to settings.py:

```python
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Once all these steps are completed your installation of DefectDojo will be running under Django 1.11

## 1.8 Running in Production

### 1.8.1 Improving your docker-compose performance

**Database**

Run your database elsewhere. Tweak your docker-compose configuration to that effect. If you don’t, you cannot pretend to be running in production.

**Instance size**

**Note:** Please read the paragraphs below about key processes tweaks.

Having taken the database to run elsewhere, the minimum recommendation is:

- 2 vCPUs
- 8 GB of RAM
- 2 GB of disk space (remember, your database is not here – so basically, what you have for your O/S should do). You could allocate a different disk than your O/S’s for potential performance improvements.

**Key processes**

Per https://github.com/DefectDojo/django-DefectDojo/pull/2813, it is now easy to somewhat improve the uWSGI and celery worker performance.
uWSGI

By default (except in ptvsd mode for debug purposes), uWSGI will handle 4 concurrent connections. Based on your resource settings, you can tweak:

- `DD_UWSGI_NUM_OF_PROCESSES` for the number of spawned processes. (default 2)
- `DD_UWSGI_NUM_OF_THREADS` for the number of threads in these processes. (default 2)

For example, you may have 4 processes with 6 threads each, yielding 24 concurrent connections.

Celery worker

By default, a single mono-process celery worker is spawned. This is fine until you start having many findings, and when async operations like deduplication start to kick in. Eventually, it will starve your resources and crawl to a halt, while operations continue to queue up.

The following variables will help a lot, while keeping a single celery worker container.

- `DD_CELERY_WORKER_POOL_TYPE` will let you switch to `prefork`. (default `solo`)

As you’ve enabled `prefork`, the following variables have to be used. The default are working fairly well, see the Dockerfile.django for in-file references.

- `DD_CELERY_WORKER_AUTOSCALE_MIN` defaults to 2.
- `DD_CELERY_WORKER_AUTOSCALE_MAX` defaults to 8.
- `DD_CELERY_WORKER_CONCURRENCY` defaults to 8.
- `DD_CELERY_WORKER_PREFETCH_MULTIPLIER` defaults to 128.

You can execute the following command to see the configuration:

docker-compose exec celerybeat bash -c "celery -A dojo inspect stats" and see what is in effect.

1.8.2 Production with setup.bash

Warning: From this point down, this page is slated to get a revamp

This guide will walk you through how to setup DefectDojo for running in production using Ubuntu 16.04, nginx, and uwsgi.

Install, Setup, and Activate Virtualenv

Assumes running as root or using sudo command for the below.

```
pip install virtualenv
cd /opt
virtualenv dojo
cd /opt/dojo
```

(continues on next page)
Install Dojo

**Warning:** The setup.bash installation method will be EOL on 2020-12-31

```bash
cd django-DefectDojo/setup
./setup.bash
```

Install Uwsgi

```bash
pip install uwsgi
```

Install WKHTML

From inside the django-DefectDojo/ directory execute:

```bash
./reports.sh
```

Disable Debugging

Using the text-editor of your choice, change `DEBUG` in django-DefectDojo/dojo/settings/settings.py to:

```python`
'DEBUG = False'
```

Configure external database

If you host your DefectDojo into AWS and you decide to use their managed database service (AWS RDS), you will have to do the following configuration updates:

1) Download the root certificate to encrypt traffic between DefectDojo and the database
2) Update your Dockerfile to add the SSL certificate to the container

Listing 1: Dockerfile.django

```bash
```

3) Update Django settings to use encrypted connection to the database (Changes highlighted below)

Listing 2: dojo/settings/settings.dist.py

```python
DATABASES = {
    'default': env.db('DD_DATABASE_URL')
}

DATABASES['default']['OPTIONS'] = {
    'ssl': {'ca': '/etc/ssl/certs/rds-ca-2019-root.pem'}
}
```

(continues on next page)
4) Update the environment variables for the database connection: `DD_DATABASE_URL` or `DD_DATABASE_HOST`, `DD_DATABASE_PORT`, `DD_DATABASE_NAME`, `DD_DATABASE_USER` and `DD_DATABASE_PASSWORD`.

Note: This configuration can be adapted to other cloud providers.

Start Celery and Beats

From inside the django-DefectDojo/ directory execute:

```bash
celery -A dojo worker -l info --concurrency 3
celery beat -A dojo -l info
```

It is recommended that you daemonized both these processes with the sample configurations found [here](#) and [here](#).

However, for a quick setup you can use the following to run both in the background:

```bash
celery -A dojo worker -l info --concurrency 3 &
celery beat -A dojo -l info &
```

Start Uwsgi

From inside the django-DefectDojo/ directory execute:

```bash
uwsgi --socket :8001 --wsgi-file wsgi.py --workers 7
```

It is recommended that you use an Upstart job or a @restart cron job to launch uwsgi on reboot. However, if you’re in a hurry you can use the following to run it in the background:

```bash
uwsgi --socket :8001 --wsgi-file wsgi.py --workers 7 &
```

Making Defect Dojo start on boot

Below we configure service files for systemd. The commands follow, the config files are below the Nginx in the next section.

```bash
$ cd /etc/systemd/system/
$ sudo vi dojo.service
[contents below]
$ sudo systemctl enable dojo
$ sudo systemctl start dojo
$ sudo systemctl status dojo
[ensure it launched OK]
$ sudo vi celery-worker.service
[contents below]
$ sudo systemctl enable celery-worker
$ sudo systemctl start celery-worker
$ sudo systemctl status celery-worker
[ensure it launched OK]
```

(continues on next page)
DefectDojo Documentation, Release 1.14.0

NGINX Configuration

Everyone feels a little differently about nginx settings, so here are the barebones to add your to your nginx configuration to proxy uwsgi. Make sure to modify the filesystem paths if needed:

```plaintext
upstream django {
    server 127.0.0.1:8001;
}

server {
    listen 80;
    return 301 https://$host$request_uri;
}

server {
    listen 443;
    server_name <YOUR_SERVER_NAME>;
    client_max_body_size 500m; # To accommodate large scan files
    ssl_certificate <PATH_TO_CRT>;
    ssl_certificate_key <PATH_TO_KEY>;
    ssl on;
    <YOUR_SSL_SETTINGS> # ciphers, options, logging, etc

    location /static/ {
        alias <PATH_TO_DOJO>/django-DefectDojo/static/;
    }

    location /media/ {
        alias <PATH_TO_DOJO>/django-DefectDojo/media/;
    }

    location / {
        uwsgi_pass django;
        include <PATH_TO_DOJO>/django-DefectDojo/wsgi_params;
    }
}
```

Systemd Configuration Files

dojo.service

[Unit]
Description=uWSGI instance to serve DefectDojo
Requires=nginx.service mysql.service
Before=nginx.service

(continues on next page)
After=mysql.service

[Service]
ExecStart=/bin/bash -c 'su - dojo -c "cd /opt/dojo/django-DefectDojo && source ../bin/activate && uwsgi --socket :8001 --wsgi-file wsgi.py --workers 7"
Restart=always
RestartSec=3
#StandardOutput=syslog
#StandardError=syslog
SyslogIdentifier=dojo

[Install]
WantedBy=multi-user.target

celery-worker.service

[Unit]
Description=celery workers for DefectDojo
Requires=dojo.service
After=dojo.service

[Service]
ExecStart=/bin/bash -c 'su - dojo -c "cd /opt/dojo/django-DefectDojo && source ../bin/activate && celery -A dojo worker -l info --concurrency 3"
Restart=always
RestartSec=3
#StandardOutput=syslog
#StandardError=syslog
SyslogIdentifier=celeryworker

[Install]
WantedBy=multi-user.target

celery-beat.service

[Unit]
Description=celery beat for DefectDojo
Requires=dojo.service
After=dojo.service

[Service]
ExecStart=/bin/bash -c 'su - dojo -c "cd /opt/dojo/django-DefectDojo && source ../bin/activate && celery beat -A dojo -l info --concurrency 3"
Restart=always
RestartSec=3
#StandardOutput=syslog
#StandardError=syslog
SyslogIdentifier=celerybeat

[Install]
WantedBy=multi-user.target

That’s it!

Monitoring

To expose Django statistics for Prometheus, using the text-editor of your choice, change DJANGO_METRICS_ENABLED to True in django-DefectDojo/dojo/settings/settings.py to:

1.8. Running in Production
DJANGO_METRICS_ENABLED = True

Or export `DD_DJANGO_METRICS_ENABLED` with the same value.

Prometheus endpoint than is available under the path: http://dd_server/django_metrics/metrics
2.1 DefectDojo Features

Below are the main sections within DefectDojo. Each is designed to allow for ease of use and simple organization of Products and their Tests. The *Models* page will help you understand the terminology we use below, so we recommend taking a look at that first.

### 2.1.1 Products

The following attributes describe a Product:

**Name**  A short name for the product, used for easy identification. This field can hold up to 300 characters.

**Description**  Used to fully describe the product. This field can hold up to 2000 characters.

**Product Manager**  Provides the ability to store who manages the product lifecycle. Useful for contacting team members. This field can hold up to 200 characters.

**Technical Contact**  Provides the ability to store who should be contacted in case of technical questions and/or difficulties. This field can hold up to 200 characters.

**Manager**  Provides the ability to store who manages the technical resources for the product. This field can hold up to 200 characters.

**Date Created**  Stores when the Product was first added to DefectDojo.

**Date Updated**  Stores when the Product was updated.

**Business Criticality**  Criticality of the product.

**Platform**  Type of product: web, API, mobile etc.

**Lifecycle**  Stage of product development

**Product Type**  Used to group products together.

**Authorized Users**  List of users who are allowed to view and interact with the product.
Products are listed on the /product page and can be filtered by their attributes as well as sorted by their name and product type.

Visual representation of a product:

Product with metrics:

2.1.2 Engagements

The following attributes describe an Engagement:
Name  Helps distinguish one Engagement from another on the same product. This field can hold up to 300 characters.

Target Start Date  The projected start date for this engagement.

Target End Date  The projected end date for this engagement.

Lead  The DefectDojo user who is considered the lead for this group of tests.

Product  The Product being tested as part of this group of tests.

Active  Denotes if the Engagement is currently active or not.

Test Strategy  The URL of the testing strategy defined for this Engagement.

Threat Model  The document generated by a threat modeling session discussing the risks associated with this product at this moment in time.

Hash Code  A hash over a configurable set of fields that is used for findings deduplication.

Payload  Payload used to attack the service / application and trigger the bug / problem.

Status  Describes the current state of the Engagement. Values include In Progress, On Hold and Completed.

Engagements are listed in the /engagement page and can be filtered by their attributes as well as sorted by the product or product type.

Visual representation of an engagement:

### 2.1.3 Endpoints

Endpoints represent testable systems defined by IP address or Fully Qualified Domain Name.

The following attributes describe an Endpoint:
Protocol  The communication protocol such as ‘http’, ‘https’, ‘ftp’, etc.

Host  The host name or IP address, you can also include the port number. For example ‘127.0.0.1’, ‘127.0.0.1:8080’, ‘localhost’, ‘yourdomain.com’.

Path  The location of the resource, it should start with a ‘/’. For example ‘/endpoint/420/edit’

Query  The query string, the question mark should be omitted. ‘For example ‘group=4&team=8’

Fragment  The fragment identifier which follows the hash mark. The hash mark should be omitted. For example ‘section-13’, ‘paragraph-2’.

Product  The Product that this endpoint should be associated with.

Endpoints are listed in the /endpoints page and can be filtered by their attributes as well as sorted by the product or host.

Visual representation of an endpoint:

Visual representation of an endpoint with metrics displayed:
2.1.4 Findings

Findings represent a flaw within the product being tested. The following attributes help define a Finding:

**Title** A short description of the flaw (Up to 511 characters).

**Description** Longer more descriptive information about the flaw.

**Date** The date the flaw was discovered.

**CVE** The Common Vulnerabilities and Exposures (CVE) associated with this flaw.

**CVSSv3** Common Vulnerability Scoring System version 3 (CVSSv3) score associated with this flaw.

**CWE** The CWE number associated with this flaw.

**URL** External reference that provides more information about this flaw.

**Severity** The severity level of this flaw (Critical, High, Medium, Low, Informational)

**Numerical Severity** The numerical representation of the severity (S0, S1, S2, S3, S4)

**Mitigation** Text describing how to best fix the flaw.

**Impact** Text describing the impact this flaw has on systems, products, enterprise, etc.

**Steps to Reproduce** Text describing the steps that must be followed in order to reproduce the flaw / bug.

**Severity Justification** Text describing why a certain severity was associated with this flaw.

**Endpoints** The hosts within the product that are susceptible to this flaw.

**Endpoint Status** The status of the endpoint associated with this flaw (Vulnerable, Mitigated, . . .).

**References** The external documentation available for this flaw.

**Thread ID** Thread ID

**Hash Code** A hash over a configurable set of fields that is used for findings deduplication.

**Test** The test that is associated with this flaw.
Is Template  Denotes if this finding is a template and can be reused.
Active  Denotes if this flaw is active or not.
Verified  Denotes if this flaw has been manually verified by the tester.
False Positive  Denotes if this flaw has been deemed a false positive by the tester.
Duplicate  Denotes if this flaw is a duplicate of other flaws reported.
Duplicate Finding  Link to the original finding if this finding is a duplicate.
Out Of Scope  Denotes if this flaw falls outside the scope of the test and/or engagement.
Under Review  Denotes is this flaw is currently being reviewed.
Mitigated  Denotes if this flaw has been fixed, by storing the date it was fixed.
Is Mitigated  Denotes if this flaw has been fixed.
Mitigated By  Documents who has deemed this flaw as fixed.
Reporter  Documents who reported the flaw.
Reviewers  Document who reviewed the flaw.
Last Reviewed  Provides the date the flaw was last “touched” by a tester.
Last Reviewed By  Provides the person who last reviewed the flaw.
Component Name  Name of the affected component (library name, part of a system, . . ).
Component Version  Version of the affected component.
Found By  The name of the scanner that identified the flaw.
SonarQube Issue  The SonarQube issue associated with this finding.
Unique ID from tool  Vulnerability technical id from the source tool. Allows to track unique vulnerabilities.
Defect Review Requested By  Document who requested a defect review for this flaw.
Under Defect Review  Denotes if this finding is under defect review.
Review Requested By  Document who requested a review for this finding.
Static Finding  Flaw has been detected from a Static Application Security Testing tool (SAST).
Dynamic Finding  Flaw has been detected from a Dynamic Application Security Testing tool (DAST).
Jira Creation  The date a Jira issue was created from this finding.
Jira Change  The date the linked Jira issue was last modified.
SLA Days Remaining  The number of day remaining to stay within the SLA.
Finding Meta  Custom metadata (K/V) that can be set on top of findings.
Tags  Add custom tags on top of findings (helpful for searching).
Created  The date the finding was created inside DefectDojo.
Param  Parameter used to trigger the issue (DAST).
Payload  Payload used to attack the service / application and trigger the bug / problem.
Age  The number of days since the finding was created.
Scanner confidence  Confidence level of vulnerability which is supplied by the scanner.
Number of Occurrences  Number of occurrences in the source tool when several vulnerabilities were found and aggregated by the scanner.

Source File  Name of the source code file in which the flaw is located.

Source File Path  Filepath of the source code file in which the flaw is located.

Notes  Stores information pertinent to the flaw or the mitigation. Initially there isn’t a way to categorize notes added for Findings. Admin can introduce a new attribute to notes as ‘note-type’ which can categorize notes. To enable note-types go to System Settings, select Note Types and add new note-types to Dojo.

Note-type  A note-type has 4 attributes.

- Name
- Description
- is_active - This has to be true to assign the note-type to a note.
- is_single - If true, only one note of that note-type can exist for a Finding.
- is_mandatory - If true, a Finding has to have at least one note from the note-type in order to close it.

If note-types are enabled, User has to first select the note-type from the “Note Type” drop down and then add the contents of the note.

Images  Image(s) / Screenshot(s) related to the flaw.

SAST specific

For SAST, when source (start of the attack vector) and sink (end of the attack vector) information are available.

Line  Source line number of the attack vector.

Line Number  Deprecated will be removed, use line.

File Path  Identified file(s) containing the flaw.

SAST Source Object  Source object (variable, function... ) of the attack vector.

SAST Sink Object  Sink object (variable, function... ) of the attack vector.

SAST Source line  Source line number of the attack vector.

SAST Source File Path  Source file path of the attack vector.

Images

Images  Finding images can now be uploaded to help with documentation and proof of vulnerability.

If you are upgrading from an older version of DefectDojo, you will have to complete the following and make sure MEDIA_ROOT and MEDIA_URL are properly configured:

Add imagekit to INSTALLED_APPS:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'polymorphic',  # provides admin templates
)
```

(continues on next page)
Add `r'^media/'` to `LOGIN_EXEMPT_URLS`:

```python
LOGIN_EXEMPT_URLS = {
    r'^static/',
    r'^api/v1/',
    r'^ajax/v1/',
    r'^reports/cover$',
    r'^finding/image/(.+)$'
}
```

Then run the following commands (make sure your virtual environment is activated):

```
pip install django-imagekit
pip install pillow --upgrade
./manage.py makemigrations dojo
./manage.py makemigrations
./manage.py migrate
```

New installations will already have finding images configured.

Findings are listed on the /finding/open, /finding/closed, /finding/accepted and /finding/all pages. They can be filtered by their attributes as well as sorted by their Name, Date, Reviewed Date, Severity and Product.
2.1. DefectDojo Features
Visual representation of a Finding:

Category: PCI DSS v3.1, PCI DSS (3.1) - 6.5.7 - Cross-site scripting (XSS), OWASP Top 10:2013 A3 – Cross-Site Scripting (XSS)
Language: Java
Group: Java High Risk
Status: New

Category: PCI DSS v3.1, PCI DSS (3.1) - 6.5.7 - Cross-site scripting (XSS), OWASP Top 10:2013 A3 – Cross-Site Scripting (XSS)
Language: Java
Group: Java High Risk
Status: New

Mitigation
Mitigation for this finding.

Impact
Impact

References
OWASP

Notes
<table>
<thead>
<tr>
<th>User</th>
<th>Date</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>July 21, 2018, 11 p.m.</td>
<td>Please follow up with more information</td>
</tr>
</tbody>
</table>

Add Note
Deduplication / Similar findings

Automatically Flag Duplicate Findings  ‘De-duplication’ is a feature that when enabled will compare findings to automatically identify duplicates. To enable de-duplication go to System Settings and check Deduplicate findings. Dojo deduplicates findings by comparing endpoints, CWE fields, and titles. If two findings share a URL and have the same CWE or title, Dojo marks the less recent finding as a duplicate. When deduplication is enabled, a list of deduplicated findings is added to the engagement view. The following image illustrates the option deduplication on engagement and deduplication on product level:

![Diagram of deduplication](image.png)

Similar Findings Visualization:

![Similar Findings Visualization](image2.png)
Similar Findings  While viewing a finding, similar findings within the same product are listed along with buttons to mark one finding a duplicate of the other. Clicking the “Use as original” button on a similar finding will mark that finding as the original while marking the viewed finding as a duplicate. Clicking the “Mark as duplicate” button on a similar finding will mark that finding as a duplicate of the viewed finding. If a similar finding is already marked as a duplicate, then a “Reset duplicate status” button is shown instead which will remove the duplicate status on that finding along with marking it active again.

2.1.5 Metrics

DefectDojo provides a number of metrics visualization in order to help with reporting, awareness and to be able to quickly communicate a products/product type’s security stance.

The following metric views are provided:

Product Type Metrics  This view provides graphs displaying Open Bug Count by Month, Accepted Bug Count by Month, Open Bug Count by Week, Accepted Bug Count by Week as well as tabular data on Top 10 Products by bug severity, Detail Breakdown of all reported findings, Opened Findings, Accepted Findings, Closed Findings, Trending Open Bug Count, Trending Accepted Bug Count, and Age of Issues.

Product Type Counts  This view provides tabular data of Total Current Security Bug Count, Total Security Bugs Opened In Period, Total Security Bugs Closed In Period, Trending Total Bug Count By Month, Top 10 By Bug Severity, and Open Findings. This view works great for communication with stakeholders as it is a snapshot in time of the product.

**IPA [View Details]**

<table>
<thead>
<tr>
<th>Total Current Security Bug Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Security Bugs Opened In Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Security Bugs Closed In Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trending Total Bug Count By Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
</tr>
<tr>
<td>Oct-2014</td>
</tr>
<tr>
<td>Nov-2014</td>
</tr>
</tbody>
</table>

**Simple Metrics** Provides tabular data for all Product Types. The data displayed in this view is the total number of S0, S1, S2, S3, S4, Opened This Month, and Closed This Month.

**Product Type Demo [View Details]**

<table>
<thead>
<tr>
<th>Total</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>Opened This Month</th>
<th>Closed This Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>159</td>
<td>0</td>
<td>4</td>
<td>22</td>
<td>19</td>
<td>114</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Third Party Apps [View Details]**

<table>
<thead>
<tr>
<th>Total</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>Opened This Month</th>
<th>Closed This Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Engineer Metrics** Provides graphs displaying information about a tester's activity.
Metrics Dashboard  Provides a full screen, auto scroll view with many metrics in graph format. This view is great for large displays or “Dashboards.”


2.1.6 Users

DefectDojo users inherit from django.contrib.auth.models.User.

A username, first name, last name, and email address can be associated with each. Additionally the following describe the type of use they are:
Active  Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Staff status  Designates whether the user can log into this site.

Superuser status  Designates that this user has all permissions without explicitly assigning them.

2.1.7 Calendar

The calendar view provides a look at all the engagements occurring during the month displayed. Each entry is a direct link to the Engagement view page.

2.1.8 Port Scans

DefectDojo has the ability to run a port scan using nmap. Scan can be configured for TCP or UDP ports as well as for a Weekly, Monthly or Quarterly frequency.

In order for the scans to kick off the `dojo.management.commands.run_scan.py` must run. It is easy to set up a cron job in order to kick these off at the appropriate frequency. Below is an example cron entry:

```
0 0 * * 0 /root/.virtualenvs/dojo/bin/python /root/defect-dojo/manage.py run_scan
→Weekly
0 0 1 * * /root/.virtualenvs/dojo/bin/python /root/defect-dojo/manage.py run_scan
→Monthly
0 0 1 3,6,9,12 * /root/.virtualenvs/dojo/bin/python /root/defect-dojo/manage.py run_scan
→Quarterly
```

The scan process will email the configured recipients with the results.
These scans can also be kicked off on demand by selecting the Launch Scan Now option in the view scan screen.

2.1.9 Notifications

DefectDojo can inform you of different events in a variety of ways. You can be notified about things like an upcoming engagement, when someone mentions you in a comment, a scheduled report has finished generating, and more.

The following notification methods currently exist: - Email - Slack - Microsoft Teams - Alerts within DefectDojo

You can set these notifications on a global scope (if you have administrator rights) or on a personal scope. For instance, an administrator might want notifications of all upcoming engagements sent to a certain Slack channel, whereas an individual user wants email notifications to be sent to the user’s specified email address when a report has finished generating.

In order to identify and notify you about things like upcoming engagements, DefectDojo runs scheduled tasks for this purpose. These tasks are scheduled and run using Celery beat, so this needs to run for those notifications to work. Instructions on how to run Celery beat are available in the Reports section.
DefectDojo utilizes the OWASP ASVS Benchmarks to benchmark a product to ensure the product meets your application technical security controls. Benchmarks can be defined per the organizations policy for secure development and multiple benchmarks can be applied to a product.

Benchmarks are available from the Product view. To view the configured benchmarks select the dropdown menu from the right hand drop down menu. You will find the selection near the bottom of the menu entitled: ‘OWASP ASVS v.3.1’.
In the Benchmarks view for each product, the default level is ASVS Level 1. On the top right hand side the drop down can be changed to the desired ASVS level (Level 1, Level 2 or Level 3). The publish checkbox will display the ASVS score on the product page and in the future this will be applied to reporting.
On the left hand side the ASVS score is displayed with the desired score, the % of benchmarks passed to achieve the score and the total enabled benchmarks for that AVSV level.

Additional benchmarks can be added/updated in the Django admin site. In a future release this will be brought out to the UI.

2.1.11 Reports

DefectDojo’s reports can be generated in AsciiDoc and PDF. AsciiDoc is recommended for reports with a large number of findings.

The PDF report is generated using wkhtmltopdf via Celery and sane defaults are included in the settings.py file. This allows report generation to be asynchronous and improves the user experience.

If you are updating from an older version of DefectDojo, you will need to install wkhtmltopdf on your own. Please follow the directions for your specific OS in the wkhtmltopdf documentation.

Some operating systems are capable of installing wkhtmltopdf from their package managers:

Note: To get report email notifications, make sure you have a working email configuration in the system settings, and enable notifications for generated reports in the notification settings.

Mac:

```
brew install Caskroom/cask/wkhtmltopdf
```

Debian/Ubuntu:

```
sudo apt-get install wkhtmltopdf
```

Fedora/Centos:
sudo yum install wkhtmltopdf

**Warning:** Version in debian/ubuntu repos has reduced functionality (because it’s compiled without the wkhtmltopdf QT patches), such as adding outlines, headers, footers, TOC etc. To use these options you should install a static binary from wkhtmltopdf site.

Additionally, DefectDojo takes advantage of python-PDFKit to interact with the wkhtmltopdf commandline interface. It is easily installed by running:

```
pip install pdfkit
```

It will also be necessary to add the path of wkhtmltopdf to your settings.py file. By default the following entry ships with DefectDojo:

```
WKHTMLTOPDF_PATH = '/usr/local/bin/wkhtmltopdf'
```

However, you may have to update that entry to suit your installation.

Celery is included with DefectDojo and needs to be kicked off in order for reports to generate/work. In development you can run the celery process like:

```
celery -A dojo worker -l info --concurrency 3
```

In production it is recommended that the celery process be daemonized. Supervisor is also included with DefectDojo and can be set up by following the Celery documentation. A sample celeryd.conf can be found at.

Celery beat should also be running, this will enable defectDojo to perform periodic checks of things like upcoming and stale engagements as well as allowing for celery to clean up after itself and keep your task database from getting too large. In development you can run the process like:

```
celery beat -A dojo -l info
```

In production it is recommended that the celery beat process also be daemonized. A sample celerybeatd.conf can be found here.

If you are upgrading from an older version of DefectDojo, you will have to install Celery on your own. To do this you can run:

```
pip install celery
```

If you are using virtual environments make sure your environment is activated. You can also follow the installation instructions from the Celery documentation.

Reports can be generated for:

1. Groups of Products
2. Individual Products
3. Endpoints
4. Product Types
5. Custom Reports

2.1. DefectDojo Features
Filtering is available on all Report Generation views to aid in focusing the report for the appropriate need. Custom reports allow you to select specific components to be added to the report. These include:

1. Cover Page
2. Table of Contents
3. WYSIWYG Content
4. Findings List
5. Endpoint List
6. Page Breaks

The custom report workflow takes advantage of the same asynchronous process described above.

### 2.1.12 Slack integration

**Scopes**

The following scopes have to be granted.
# Scopes

A Slack app's capabilities and permissions are governed by the **scopes** it requests.

## Bot Token Scopes

Scopes that govern what your app can access.

<table>
<thead>
<tr>
<th>OAuth Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chat:write</td>
<td>Send messages as @testdojo</td>
</tr>
<tr>
<td>chat:write.customize</td>
<td>Send messages as @testdojo with a customized username and avatar</td>
</tr>
<tr>
<td>chat:write.public</td>
<td>Send messages to channels @testdojo isn't a member of</td>
</tr>
<tr>
<td>incoming-webhook</td>
<td>Post messages to specific channels in Slack</td>
</tr>
<tr>
<td>users:read</td>
<td>View people in the workspace</td>
</tr>
<tr>
<td>users:read.email</td>
<td>View email addresses of people in the workspace</td>
</tr>
</tbody>
</table>

[Add an OAuth Scope]

## User Token Scopes

Scopes that access user data and act on behalf of users that authorize them.

<table>
<thead>
<tr>
<th>OAuth Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>You haven’t added any OAuth Scopes for your User token.</td>
<td></td>
</tr>
</tbody>
</table>

[Add an OAuth Scope]

## Token

The bot token has to be chosen and put in your System Settings
2.1.13 JIRA Integration

DefectDojo’s JIRA integration is bidirectional. You may push findings to JIRA and share comments. If an issue is closed in JIRA it will automatically be closed in Dojo.

NOTE: These steps will configure the necessary webhook in JIRA and add JIRA integration into DefectDojo. This isn’t sufficient on its own. You will need to configure products and findings to push to JIRA. On a product’s settings page you will need to define a:

- Project Key (and this project must exist in JIRA)
- JIRA Configuration (select the JIRA configuration that you create in the steps below)
- Component (can be left blank)

Then elect (via tickbox) whether you want to ‘Push all issues’, ‘Enable engagement epic mapping’ and/or ‘Push notes’. Then click on ‘Submit’.

Enabling the Webhook

1. Visit https://YOUR JIRA URL>/plugins/servlet/webhooks
2. Click ‘Create a Webhook’
3. For the field labeled ‘URL’ enter: https://YOUR DOJO DOMAIN>/webhook/

Configurations in Dojo

1. Navigate to the System Settings from the menu on the left side or by directly visiting <your url>/system_settings.
2. Enable ‘Enable JIRA integration’ and click submit.
Adding JIRA to Dojo

1. Click ‘JIRA’ from the left hand menu.
2. Select ‘Add Configuration’ from the drop-down.
3. If you use Jira Cloud, you will need to generate an API token for Jira to use as the password
4. To obtain the ‘open status key’ and ‘closed status key’ visit https://<YOUR JIRA URL>/rest/api/latest/issue/<ANY VALID ISSUE KEY>/transitions?expand=transitions.fields
5. The ‘id’ for ‘Todo’ should be filled in as the ‘open status key’
6. The ‘id’ for ‘Done’ should be filled in as the ‘closed status key’

To obtain ‘epic name id’: If you have admin access to JIRA:

1. visit: https://<YOUR JIRA URL>/secure/admin/ViewCustomFields.jspa
2. Click on the cog next to ‘Epic Name’ and select view.
3. The numeric value for ‘epic name id’ will be displayed in the URL
4. Note: dojojira uses the same celery functionality as reports. Make sure the celery runner is setup correctly as described: http://defectdojo.readthedocs.io/en/latest/features.html#reports

Or
1. login to JIRA
2. visit https://yourjiraurl/rest/api/2/field and use control+F or grep to search for ‘Epic Name’ it should look something like this:

   ```json
   {"id":"customfield_122","key":"customfield_122","name":"Epic Name","custom":true,"orderable":true,"navigable":true,"searchable":true,"clauseNames": ["cf[122]","Epic Name"],"schema": {"type": "string","custom": "com.pyxis.greenhopper.jira:gh-epic-label","customId":122}}
   ``

   **In the above example 122 is the number needed**

Engagement Epic Mapping

If creating an Engagement, ensure to tick ‘Enable engagement epic mapping’ if desired. This can also be done after engagement creation on the edit engagement page. This will create an ‘Epic’ type issue within Jira. All findings in the engagement pushed to Jira will have a link to this Epic issue. If Epic Mapping was enabled after associated findings have already been pushed to Jira, simply pushing them again will link the Jira issue to the Epic issue.

Pushing findings

Findings can be pushed to Jira in a number of ways:

1. When importing scanner reports, select ‘Push to JIRA’ to push every single finding in the report to Jira
2. When creating a new finding, select ‘Push to JIRA’ and submit. This will create the finding in DefectDojo and Jira simultaneously
3. If a finding already exist, visit the edit finding page and find the ‘Push to JIRA’ tick box at the bottom
4. When viewing a list of findings, select each relevant tick boxes to the left of the finding, and click the ‘Bulk Edit’ button at the top, find ‘Push to JIRA’ at the bottom of the menu


**Known Issues**

DefectDojo will try to keep the status in sync with the status in JIRA using the various status IDs configured for each JIRA instance. This will only work if your workflow in JIRA allows arbitrary transitions between the statuses JIRA issues can be in. The Risk Acceptance feature in DefectDojo will (for that reason) not (yet) try to sync statuses. A comment will be pushed to JIRA if a finding is risk accepted or unaccepted. Contributions are welcome to enhance the integration.

**Troubleshooting JIRA integration**

JIRA actions are typically performed in the celery background process. Errors are logged as alerts/notifications to be seen on the top right of the DefectDojo UI and in stdout of the celery workers.

### 2.1.14 Issue Consolidation

DefectDojo allows users to automatically consolidate issues from multiple scanners to remove duplicates.

To enable this feature, hover over the configuration tab on the left menu and click on system settings. In system settings, click ‘Deduplicate findings’. Click ‘Submit’ at the bottom of the page.

When deduplication is enabled, Dojo will compare CWE, title, and endpoint details for all findings in a given product. If an issue is added with either the CWE or title being the same while the endpoint is also the same, Dojo marks the old issue as a duplicate.

### 2.1.15 False Positive Removal

DefectDojo allows users to tune out false positives by enabling False Positive History. This will track what engineers have labeled as false positive for a specific product and for a specific scanner. While enabled, when a tool reports the same issue that has been flagged as a false positive previously, it will automatically mark the finding as a false positive, helping to tune overly verbose security tools.

### 2.1.16 Deduplication

Deduplication is a process that allows DefectDojo to find out that a finding has already been imported.

Upon saving a finding, defectDojo will look at the other findings in the product or the engagement (depending on the configuration) to find duplicates.

When a duplicate is found:

- The newly imported finding takes status: inactive, duplicate
- An “Original” link is displayed after the finding status, leading to the original finding

There are two ways to use the deduplication:

- **Deduplicate vulnerabilities in the same build/release. The vulnerabilities may be found by the same scanner (same scanner deduplication)**
  - this helps analysis and assessment of the technical debt, especially if using many different scanners; although detecting duplicates across scanners is not trivial as it requires a certain standardization.

- **Track unique vulnerabilities across builds/releases so that defectDojo knows when it finds a vulnerability whether it has seen it before**
  - this allows you keep information attached to a given finding in a unique place: all further duplicate findings will point to the original one.
Deduplication Configuration

Global configuration

The deduplication can be activated in “System Settings” by ticking “Deduplicate findings”.
An option to delete duplicates can be found in the same menu, and the maximum number of duplicates to keep for the same finding can be configured.

Engagement configuration

When creating an engagement or later by editing the engagement, the “Deduplication within engagement only” checkbox can be ticked.

- If activated: Findings are only deduplicated within the same engagement. Findings present in different engagements cannot be duplicates
- Else: Findings are deduplicated across the whole product

Note that deduplication can never occur across different products.

Deduplication algorithms

The behavior of the deduplication can be configured for each parser in settings.dist.py (or settings.py after install) by configuring the `DEDUPLICATION_ALGORITHM_PER_PARSER` variable.

The available algorithms are:

- `DEDUPE_ALGO_UNIQUE_ID_FROM_TOOL`
  - the deduplication occurs based on finding.unique_id_from_tool which is a unique technical id existing in the source tool. Few scanners populate this field currently. If you want to use this algorithm, you may need to update the scanner code beforehand
  - The tools that populate the unique_id_from_tool field are:
    * Checkmarx Scan detailed
    * SonarQube Scan detailed
  - Advantages:
    * If your source tool has a reliable means of tracking a unique vulnerability across scans, this configuration will allow defectDojo to use this ability
  - Drawbacks:
    * Using this algorithm will not allow cross-scanner deduplication as other tools will have a different technical id.
    * When the tool evolves, it may change the way the unique id is generated. In that case you won’t be able to recognise that findings found in previous scans are actually the same as the new findings.

- `DEDUPE_ALGO_HASH_CODE`
  - the deduplication occurs based on finding.hash_code. The hash_code itself is configurable for each scanner in parameter `HASHCODE_FIELDS_PER_SCANNER`

- `DEDUPE_ALGO_UNIQUE_ID_FROM_TOOL_OR_HASH_CODE`
- A finding is a duplicate with another if they have the same unique_id_from_tool OR the same hash_code

- **Allows to use both**
  - A technical deduplication (based on unique_id_from_tool) for a reliable same-parser deduplication
  - And a functional one (based on hash_code configured on CWE+severity+file_path for example) for cross-parser deduplication

**DEDUPE_ALGO_LEGACY**

- This is the algorithm that was in place before the configuration per parser was made possible, and also the default one for backward compatibility reasons.

- **Legacy algorithm basically deduplicates based on:**

- Note that there are some subtleties that may give unexpected results. Switch `dojo.specific_loggers.deduplication` to debug in settings.py to get more info in case of trouble.

**Hash_code computation configuration**

The hash_code computation can be configured for each parser using the parameter `HASHCODE_FIELDS_PER_SCANNER` in settings.dist.py.

The parameter `HASHCODE_ALLOWED_FIELDS` list the fields from finding table that were tested and are known to be working when used as a hash_code. Don’t hesitate to enrich this list when required (the code is generic and allows adding new fields by configuration only).

Note that `endpoints` isn’t a field from finding table but rather a meta value that will trigger a computation based on all the endpoints.

When populating `HASHCODE_FIELDS_PER_SCANNER`, please respect the order of declaration of the fields: use the same order as in `HASHCODE_ALLOWED_FIELDS` : that will allow cross-scanner deduplication to function because the hash_code is computed as a sha-256 of concatenated values of the configured fields.

**Tips:**

- It’s advised to use fields that are standardized for a reliable deduplication, especially if aiming at cross-scanner deduplication. For example `title` and `description` tend to change when the tools evolve and don’t allow cross-scanner deduplication

- **Good candidates are**
  - cwe or cve
  - Adding the severity will make sure the deduplication won’t be too aggressive (there are several families of XSS and sql injection for example, with various severities but the same cwe).
  - Adding the `file_path` or `endpoints` is advised too.

- The parameter `HASHCODE_ALLOWS_NULL_CWE` will allow switching to legacy algorithm when a null cwe is found for a given finding: this is to avoid getting many duplicates when the tool fails to give a cwe while we are expecting it.
Debugging deduplication

There is a specific logger that can be activated in order to have details about the deduplication process: switch `dojo.specific-loggers.deduplication` to debug in `settings.py`.

**Deduplication - APIv2 parameters**

- `skip_duplicates`: if true, duplicates are not inserted at all
- `close_old_findings`: if true, findings that are not duplicates and that were in the previous scan of the same type (example ZAP) for the same product (or engagement in case of “Deduplication on engagement”) and that are not present in the new scan are closed (Inactive, Verified, Mitigated)

### 2.1.17 Google Sheets Sync

With the Google Sheets sync feature, DefectDojo allow the users to export all the finding details of each test into a separate Google Spreadsheet. Users can review and edit finding details via Google Spreadsheets. Also, they can add new notes to findings and edit existing notes using the Google Spreadsheet. After reviewing and updating the finding details in the Google Spreadsheet, the user can import (sync) all the changes done via the Google Spreadsheet into DefectDojo database.

**Configuration**

**Creating a project and a Service Account**

1. Go to the Service Accounts page.
2. Create a new project for DefectDojo and select it.
3. Click +CREATE SERVICE ACCOUNT, enter a name and description for the service account. You can use the default service account ID, or choose a different, unique one. When done click Create.
4. The Service account permissions (optional) section that follows is not required. Click Continue.
5. On the Grant users access to this service account screen, scroll down to the Create key section. Click +Create key.
6. In the side panel that appears, select the format for your key as JSON
7. Click Create. Your new public/private key pair is generated and downloaded to your machine.

**Enabling the required APIs**

1. Go to the Google API Console.
2. From the projects list, select the project created for DefectDojo.
3. If the APIs & services page isn’t already open, open the console left side menu and select APIs & services, and then select Library.
4. Google Sheets API and Google Drive API should be enabled. Click the API you want to enable. If you need help finding the API, use the search field.
5. Click ENABLE.

**Configurations in DefectDojo**

1. Click ‘Configuration’ from the left hand menu.
2. Click ‘Google Sheets Sync’.
3. Fill the form.

   a. Upload the downloaded json file into the **Upload Credentials file** field.

   b. Drive Folder Id

      a. Create a folder inside the Google drive of the same gmail account used to create the service account.

      b. Get the `client_email` from the downloaded json file and share the created drive folder with `client_email` giving **edit access**.

      c. Extract the folder id from the URL and insert it as the **Drive Folder Id**.

   c. Tick the **Enable Service** check box. *(Optional as this has no impact on the configuration, but you must set it to true inorder to use the feature. Service can be enabled or disabled at any point after the configuration using this check box)*

   d. For each field in the finding table there are two related entries in the form.

      a. In the drop down, select Hide if the column needs to be hidden in the Google Sheet, else select any other option based on the length of the entry that goes under the column.

      b. If the column needs to be protected in the Google Sheet, tick the check box. Otherwise leave it unchecked.

4. Click ‘Submit’.

Admin has the privilege to revoke the access given to DefectDojo to access Google Sheets and Google Drive data by simply clicking the **Revoke Access** button.

### Using Google Sheets Sync Feature

Before a user can export a test to a Google Spreadsheet, admin must Configure Google Sheets Sync and **Enable** sync feature. Depending on whether a Google Spreadsheet exists for the test or not, the User interface displayed will be different.

If a Google Spreadsheet does not exist for the Test:
If a Google Spreadsheet is already created for the Test:

After creating a Google Spreadsheet, users can review and edit Finding details using the Google Sheet. If any change is done in the Google Sheet users can click the **Sync Google Sheet** button to get those changes into DefectDojo.

### 2.1.18 Service Level Agreement (SLA)

DefectDojo allows you to maintain your security SLA and automatically remind teams whenever a SLA is about to get breached, or breaches.

Simply indicate in the **System Settings** for each severity, how many days teams have to remediate a finding.
SLA notification configuration

There are 5 variables in the settings.py file that you can configure, to act on the global behavior. By default, any findings across the instance that are in Active, Verified state will be considered for notifications.

```python
SLA_NOTIFY_ACTIVE = False
SLA_NOTIFY_ACTIVE_VERIFIED_ONLY = True
SLA_NOTIFY_WITH_JIRA_ONLY = False
SLA_NOTIFY_PRE_BREACH = 3
SLA_NOTIFY_POST_BREACH = 7
```

Setting both SLA_NOTIFY_ACTIVE and SLA_NOTIFY_ACTIVE_VERIFIED_ONLY to False will effectively disable SLA notifications.

You can choose to only consider findings that have a JIRA issue linked to them. If so, please set SLA_NOTIFY_WITH_JIRA_ONLY to True.

The SLA_NOTIFY_PRE_BREACH is expressed in days. Whenever a finding’s “SLA countdown” (time to remediate) drops to this number, a notification would be sent everyday, as scheduled by the crontab in settings.py, until the day it breaches.

The SLA_NOTIFY_POST_BREACH lets you define in days how long you want to be kept notified about findings that have breached the SLA. Passed that number, notifications will cease.

**Warning:** Be mindful of performance if you choose to have SLA notifications on non-verified findings, especially if you import a lot of findings through CI in ‘active’ state.

What notification channels for SLA notifications?

The same as usual. You will notice that an extra SLA breach option is now present on the Notification page and also in the Product view.
SLA notification with JIRA

You can choose to also send SLA notification as JIRA comments, if your product is configured with JIRA. You can enable it at the JIRA configuration level or at the Product level.

The Product level JIRA notification configuration takes precedence over the global JIRA notification configuration.

When is the SLA notification job run?

The default setup will trigger the SLA notification code at 7:30am on a daily basis, as defined in the settings.py file. You can of course modify this schedule to your context.

```
'compute-sla-age-and-notify': {
    'task': 'dojo.tasks.async_sla_compute_and_notify',
    'schedule': crontab(hour=7, minute=30),
}
```

**Note:** The celery containers are the ones concerned with this configuration. If you suspect things are not working as expected, make sure they have the latest version of your settings.py file.

You can of course change this default by modifying that stanza.

Launching from the CLI

You can also invoke the SLA notification function from the CLI. For example, if run from docker-compose:

```
$ docker-compose exec uwsgi /bin/bash -c 'python manage.py sla_notifications'
```
2.2 Setting up Social Authentication via OAuth2 Providers

2.2.1 Auth0 OAuth2 Configuration

In the same way as with other Identity-Providers, it’s now possible to leverage Auth0 to authenticate users on DefectDojo.

1. Inside your Auth0 dashboard create a new application (Applications / Create Application / Single Page Web Application).

2. On the new application set the following fields:
   - Name: “Defectdojo”
   - Allowed Callback URLs: “https://the_hostname_you_have_dojo_deployed:your_server_port/complete/auth0/”

3. Copy the following info from the application:
   - Domain
   - Client ID
   - Client Secret

3. Now, edit the dojo/settings.py file and edit/replace the following information:
   - DD_SOCIAL_AUTH_AUTH0_OAUTH2_ENABLED=True
   - DD_SOCIAL_AUTH_AUTH0_KEY=(str, ‘YOUR_CLIENT_ID_FROM_STEP_ABOVE’),
   - DD_SOCIAL_AUTH_AUTH0_SECRET=(str, ‘YOUR_CLIENT_SECRET_FROM_STEP_ABOVE’),
   - DD_SOCIAL_AUTH_AUTH0_DOMAIN=(str, ‘YOUR_AUTH0_DOMAIN_FROM_STEP_ABOVE’),

5. Restart DefectDojo, and you should now see a Login with Auth0 button on the login page.

2.2.2 Google

New to DefectDojo, a Google account can now be used for Authentication, Authorization, and a DefectDojo user. Upon login with a Google account, a new user will be created if one does not already exist. The criteria for determining whether a user exists is based on the users username. In the event a new user is created, the username is that of the Google address without the domain. Once created, the user creation process will not happen again as the user is recalled by its username, and logged in. In order to make the magic happen, a Google authentication server needs to be created. Closely follow the steps below to guarantee success.

1. Navigate to the following address and either create a new account, or login with an existing one: Google Developers Console

2. Once logged in, find the key shaped button labeled Credentials on the left side of the screen. Click Create Credentials, and choose OAuth Client ID:
3. Select **Web Applications**, and provide a descriptive name for the client.
4. Add the pictured URLs in the **Authorized Redirect URLs** section. This part is very important. If there are any mistakes here, the authentication client will not authorize the request, and deny access.

5. Once all URLs are added, finish by clicking **Create**

Now with the authentication client created, the **Client ID** and **Client Secret Key** need to be copied over to `settings.py` in the project. Click the newly created client and copy the values:
In the **Environment** section at the top of settings.py, enter the values as shown below:

```plaintext
DD_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = (str, 'Client Key'),
DD_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = (str, 'Secret Key'),
```

In the **Authentication** section of settings.py, set `DD_GOOGLE_OAUTH_ENABLED` to `True` to redirect away from this README and actually authorize.

```plaintext
GOOGLE_OAUTH_ENABLED = True
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = env('DD_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY')
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = env('DD_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET')
```

To authorize users you will need to set the following:

- `SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_DOMAINS = ['example.com', 'example.org']`

or

- `SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_EMAILS = ['email@example.com']`

### 2.2.3 OKTA

In a similar fashion to that of Google, using OKTA as a OAuth2 provider carries the same attributes and a similar procedure. Follow along below.

1. Navigate to the following address and either create a new account, or login with an existing one: [OKTA Account Creation](#)
2. Once logged in, enter the **Applications** and click **Add Application**:
3. Select **Web Applications**.

4. Add the pictured URLs in the **Login Redirect URLs** section. This part is very important. If there are any mistakes here, the authentication client will not authorize the request, and deny access. Check the **Implicit** box.
as well.

5. Once all URLs are added, finish by clicking Done.

6. Return to the Dashboard to find the Org-URL. Note this value as it will be important in the settings file.
Now, with the authentication client created, the **Client ID** and **Client Secret** Key need to be copied over to settings.py in the project. Click the newly created client and copy the values:

![Client Credentials](image)

In the **Environment** section at the top of settings.py, enter the values as shown below:

```
DD_SOCIAL_AUTH_OKTA_OAUTH2_KEY = (str, 'Client Key'),
DD_SOCIAL_AUTH_OKTA_OAUTH2_SECRET = (str, 'Secret Key'),
DD_SOCIAL_AUTH_OKTA_OAUTH2_API_URL = (str, 'https://{your-url}/oauth2/default'),
```

In the **Authentication** section of settings.py, set **DD_OKTA_OAUTH_ENABLED** to **True** to redirect away from this README and actually authorize.

```
OKTA_OAUTH_ENABLED = True
SOCIAL_AUTH_OKTA_OAUTH2_KEY = env('DD_SOCIAL_AUTH_OKTA_OAUTH2_KEY')
SOCIAL_AUTH_OKTA_OAUTH2_SECRET = env('DD_SOCIAL_AUTH_OKTA_OAUTH2_SECRET')
SOCIAL_AUTH_OKTA_OAUTH2_API_URL = env('DD_SOCIAL_AUTH_OKTA_OAUTH2_API_URL')
```

If during the login process you get the following error: *The 'redirect_uri' parameter must be an absolute URI that is whitelisted in the client app settings.* and the `redirect_uri` HTTP GET parameter starts with `http://` instead of `https://` you need to add **SOCIAL_AUTH_REDIRECT_IS_HTTPS** = **True** in the **Authentication** section of settings.py.

### 2.2.4 Azure Active Directory Tenant Configuration

You can now use your corporate Azure Active Directory to authenticate users to Defect Dojo. Users will be using your corporate Azure AD account (A.K.A. Office 365 identity) to authenticate via OAuth, and all the conditional access rules and benefits from Azure Active Directory will also apply to the Defect Dojo Authentication. Once the user signs in, it will try to match the UPN of the user to an existing e-mail from a user in Defect Dojo, and if no match is found, a new user will be created in Defect Dojo, associated with the unique id/value of the user provided by your Azure AD tenant. Then, you can assign roles to this user, such as ‘staff’ or ‘superuser’
1. Navigate to the following address and follow instructions to create a new app registration
   • https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app
2. Once you register an app, take note of the following information:
   • Application (client) ID
   • Directory (tenant) ID
   • Under Certificates & Secrets, create a new Client Secret
3. Under Authentication > Redirect URIs, add a WEB type of uri where the redirect points to
   • http://localhost:8080/complete/azuread-tenant-oauth2/
   • OR
   • https://the_hostname_you_have_dojo_deployed:your_server_port/complete/azuread-tenant-oauth2/
4. Now, edit the dojo/settings.py file and edit/replace the following information:
   • DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY=(str, ‘YOUR_APPLICATION_ID_FROM_STEP_ABOVE’),
   • DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET=(str, ‘YOUR_CLIENT_SECRET_FROM_STEP_ABOVE’),
   • DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID=(str, ‘YOUR_DIRECTORY_ID_FROM_STEP_ABOVE’),
   • DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_ENABLED = True
5. Restart your Dojo, and you should now see a Login with Azure AD button on the login page which should magically work

### 2.2.5 Gitlab OAuth2 Configuration

In a similar fashion to that of Google and OKTA, using Gitlab as a OAuth2 provider carries the same attributes and a similar procedure. Follow along below.

1. Navigate to your Gitlab settings page and got to the Applications section
   • https://gitlab.com/profile/applications
   • OR
   • https://the_hostname_you_have_gitlab_deployed:your_gitlab_port/profile/applications
2. Choose a name for your application
3. For the Redirect URI, enter the DefectDojo URL with the following format
   • https://the_hostname_you_have_dojo_deployed:your_server_port/complete/gitlab/
4. Now, edit the dojo/settings.py file and edit/replace the following information:
   • DD_SOCIAL_AUTH_GITLAB_KEY=(str, ‘YOUR_APPLICATION_ID_FROM_STEP_ABOVE’),
   • DD_SOCIAL_AUTH_GITLAB_SECRET=(str, ‘YOUR_SECRET_FROM_STEP_ABOVE’),
   • DD_SOCIAL_AUTH_GITLAB_API_URL=(str, ‘https://gitlab.com’),
   • DD_SOCIAL_AUTH_GITLAB_OAUTH2_ENABLED = True
   Additionally, if you want to import your Gitlab projects as DefectDojo products, add the following line, still in dojo/settings.py:
   • DD_SOCIAL_AUTH_GITLAB_PROJECT_AUTO_IMPORT = True
5. Restart DefectDojo, and you should now see a Login with Gitlab button on the login page.
2.2.6 SAML 2.0

In a similar direction OAuth, this SAML addition provides a more secure perogative to SSO. For definitions of terms used and more information, see the plugin plugin homepage

1. Navigate to your SAML IdP and find your metadata
2. Edit the dojo/settings.py file:
   - DD_SAML2_ENABLED=(bool, True),
   - If the metadata can be accessed from a url, try the DD_SAML2_METADATA_AUTO_CONF_URL
   - DD_SAML2_METADATA_AUTO_CONF_URL=(str, 'https://your_IdP.com/metadata.xml'),
   - Otherwise, download a copy of the metadata into an xml file, and list the path in DD_SAML2_METADATA_LOCAL_FILE_PATH
   - DD_SAML2_METADATA_LOCAL_FILE_PATH=(str, '/path/to/your/metadata.xml'),
   - Fill in DD_SAML2_ASSERTION_URL and DD_SAML2_ENTITY_ID to match the specs of you IdP.
   - Configure the remaining optional fields to your desire.
4. In the “Authentication” section of the settings.py, do the following
   - Find the “SAML_2_AUTH” dictionary
   - Comment out the metadata collection method that was not used.
   - For example, if METADATA_AUTO_CONF_URL was used, comment the METADATA_LOCAL_FILE_PATH line.
5. Restart DefectDojo, and you should now see a Login with SAML button on the login page.

NOTE: In the case when IDP is configured to use self signed certificate, than CA needs to be specified by define environments variable REQUESTS_CA_BUNDLE that points to the path of public CA certificate.

2.2.7 User Permissions

When a new user is created via the social-auth, the default permissions are only active. This means that the newly created user can’t log in with the SAML aka

is_superuser = True

Exception for Gitlab OAuth2: with DD_SOCIAL_AUTH_GITLAB_PROJECT_AUTO_IMPORT set to True in dojo/settings.py, where a new user is created via the Gitlab social-auth, he has one permission: add_engagement. It allows him to create further engagements on his products via API v1.

2.2.8 Other Providers

In an effort to accommodate as much generality as possible, it was decided to implement OAuth2 with the social-auth ecosystem as it has a library of compatible providers with documentation of implementation. Conveniently, each provider has an identical procedure of managing the authenticated responses and authorizing access within a given application. The only difficulty is creating a new authentication client with a given OAuth2 provider.
CHAPTER 3

API and settings Documentation

3.1 DefectDojo API Documentation

**Warning:** API v1 is deprecated and will be no longer maintained after 12-31-2020. API v1 will be removed on 06-30-2021. Please move to API v2 and raise issues for any unsupported operations.

DefectDojo’s API is created using Tastypie. The documentation of each endpoint is available within each DefectDojo installation at /api/v1/doc/ and can be accessed by choosing the API Docs link on the user drop down menu in the header.

The documentation is generated using Tastypie Swagger, and is interactive.
To interact with the documentation, a valid Authorization header value is needed. Visit the /api/key/ view to generate your API Key and copy the header value provided.

**API Key**

Below is your current API Key, to regenerate click the button below.

Your current API key is: 7278aeefdc3c93b601f692440e37c44aa856e86
Your current API Authorization Header value is: ApiKey jay7958:7278aeefdc3c93b601f692440e37c44aa856e86
Has your key been exposed? Are you ready for a new one?

Generate New Key

That’s great, but how do I make calls to the API? Take a look at the API Documentation.

To use your API Key you need to specify an Authorization header. Example:

```python
# As a header:
# Format is 'Authorization: ApiKey <username>:<api_key>
Authorization: ApiKey jay7958:7278aeefdc3c93b601f692440e37c44aa856e86
```

Here is a simple python example against the /users endpoint

```python
import requests

url = 'http://127.0.0.1:8000/api/v1/users/
headers = {'content-type': 'application/json'},
'Authorization': 'ApiKey jay7958:7278aeefdc3c93b601f692440e37c44aa856e86'}

r = requests.get(url, headers=headers, verify=True) # set verify to false if ssl cert is self-signed

for key, value in r._dict__.items():
    print key
    print value
    print '------------'
```

Return to the /api/v1/doc/ view to paste your key in the form field and click **Explore**. Your authorization header value will be captured and used for all requests.

Each section allows you to make calls to the API and view the Request URL, Response Body, Response Code and Response Headers.

Currently the following endpoints are available:

- Engagements
- Findings
- Products
- Scan Settings
- Scans
• Tests
• Users

3.1.1 Authentication

The API uses header authentication with API key. The format of the header should be:

Authorization: ApiKey <username>:<api_key>

For example:

Authorization: ApiKey jay7958:c8572a5adf107a693aa6c72584da31f4d1f1dcff

3.1.2 Sample Code

Here are some simple python examples and their results produced against the /users endpoint:

```python
import requests
url = 'http://127.0.0.1:8000/api/v1/users'
headers = {'content-type': 'application/json',
           'Authorization': 'ApiKey jay7958:c8572a5adf107a693aa6c72584da31f4d1f1dcff'}
r = requests.get(url, headers=headers, verify=True)  # set verify to False if ssl cert is self-signed
for key, value in r.__dict__.iteritems():
    print key
    print value
    print '------------------'
```

This code will display the list of all the users defined in DefectDojo. The json object result of the above code is:

```json
{
    "meta": {
        "limit": 20,
        "next": null,
        "offset": 0,
        "previous": null,
        "total_count": 3
    },
    "objects": [
        {
            "first_name": "Greg",
            "id": 22,
            "last_login": "2018-10-28T08:05:51.925743",
            "last_name": "",
            "resource_uri": "/api/v1/users/22/",
            "username": "greg.dev"
        },
        {
            "first_name": "Andy",
            "id": 29,
            "last_login": "2019-05-28T08:05:51.925743",
            "last_name": "",
            "resource_uri": "/api/v1/users/29/",
            "username": "andy.dev"
        }
    ]
}
```

(continues on next page)
Here is another example against the /users endpoint, we apply the condition(username__contains=jay) which will filter and display the list of the users whose username includes jay:

```python
import requests
url = 'http://127.0.0.1:8000/api/v1/users/?username__contains=jay'
headers = {'content-type': 'application/json',
           'Authorization': 'ApiKey jay7958:c8572a5adf107a693aa6c72584da31f4d1f1dcff'}
r = requests.get(url, headers=headers, verify=True) # set verify to False if ssl cert is self-signed
for key, value in r.__dict__.iteritems():
    print key
    print value
    print '------------------'

The json object result of the above code is:

```
Here is a simple python POST example for creating a new product_type:

```python
import requests

url = 'http://127.0.0.1:8000/api/v1/product_types/'
data = {
    'name':'Spartans Dev Team',
    "critical_product": "true",
    "key_product": "true"
}
headers = {'content-type': 'application/json',
           'Authorization': 'ApiKey jay7958:c8572a5adf107a693a6c72584da31f4d1f1dcff
           '
           r = requests.post(url, json = data, headers=headers, verify=True) # set verify to False if ssl cert is self-signed

print("The response status code :\$s"%r.status_code)
print("The response text is :\$s"%r.text)
```

See Tastypie’s documentation on interacting with an API for additional examples and tips.

See defectdojo_api project, a Python API wrapper for DefectDojo (a utility to call the API using python)

### 3.1.3 Manually calling the API

Tools like Postman can be used for testing the API.

Example for importing a scan result:

- **Verb**: POST
- **URI**: http://localhost:8080/api/v1/importscan/
- **Headers tab**: add the authentication header
  - **Key**: Authorization
  - **Value**: ApiKey jay7958:c8572a5adf107a693a6c72584da31f4d1f1dcff
- **Body tab**
  - select “form-data”, click “bulk edit”. Example for a ZAP scan:

```text
verified: true
active: true
lead:/api/v1/users/1/
tags:test
scan_date: 2019-04-30
scan_type: ZAP Scan
minimum_severity: Info
engagement:/api/v1/engagements/1/
```

- **Body tab**
– Click “Key-value” edit
– Add a “file” parameter of type “file”. This will trigger multi-part form data for sending the file content
– Browse for the file to upload
• Click send

3.2 DefectDojo API v2 Documentation

DefectDojo’s API is created using Django Rest Framework. The documentation of each endpoint is available within each DefectDojo installation at /api/v2/doc/ and can be accessed by choosing the API v2 Docs link on the user drop down menu in the header.

The documentation is generated using Django Rest Framework Swagger, and is interactive.

To interact with the documentation, a valid Authorization header value is needed. Visit the /api/v2/key/ view to generate your API Key (Token <api_key>) and copy the header value provided.

**Defect Dojo API v2**

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/v2/engagements/</td>
<td>List engagements</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/engagements/</td>
<td>Create engagement</td>
</tr>
<tr>
<td>GET</td>
<td>/api/v2/engagements/&lt;id&gt;/</td>
<td>Retrieve engagement by ID</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/engagements/&lt;id&gt;/</td>
<td>Update engagement</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/engagements/&lt;id&gt;/</td>
<td>Replace engagement</td>
</tr>
</tbody>
</table>

Return to the /api/v2/doc/ and click on Authorize to open Authorization form. Paste your key in the form field provided and click on Authorize button. Your authorization header value will be captured and used for all requests.

Each section allows you to make calls to the API and view the Request URL, Response Body, Response Code and Response Headers.
Currently the following endpoints are available:

- Engagements
- Findings
- Products
- Scan Settings
- Scans
- Stub Findings
- Tests
- Tool Configurations
- Tool Product Settings
- Tool Types
- Users

### 3.2.1 Authentication

The API uses header authentication with API key. The format of the header should be:

```
Authorization: Token <api.key>
```

For example:
3.2.2 Sample Code

Here are some simple python examples and their results produced against the /users endpoint:

```python
import requests

url = 'http://127.0.0.1:8000/api/v2/users'
headers = {'content-type': 'application/json',
          'Authorization': 'Token c8572a5adf107a693aa6c72584da31f4d1f1dcff'}

r = requests.get(url, headers=headers, verify=True)  # set verify to False if ssl cert is self-signed

for key, value in r.__dict__.iteritems():
    print key
    print value
    print '------------------'
```

This code will return the list of all the users defined in DefectDojo. The json object result looks like:

```
[
    {
        "first_name": "Tyagi",
        "id": 22,
        "last_login": "2019-06-18T08:05:51.925743",
        "last_name": "Paz",
        "resource_uri": "/api/v1/users/22/",
        "username": "dev7958"
    },
    {
        "first_name": "saurabh",
        "id": 31,
        "last_login": "2019-06-06T11:44:32.533035",
        "last_name": "",
        "resource_uri": "/api/v1/users/31/",
        "username": "saurabh.paz"
    }
]
```

Here is another example against the /users endpoint, this time we will filter the results to include only the users whose user name includes jay:

```python
import requests

url = 'http://127.0.0.1:8000/api/v2/users/?username__contains=jay'
headers = {'content-type': 'application/json',
          'Authorization': 'Token c8572a5adf107a693aa6c72584da31f4d1f1dcff'}

r = requests.get(url, headers=headers, verify=True)  # set verify to False if ssl cert is self-signed

for key, value in r.__dict__.iteritems():
    print key
    print value
    print '------------------'
```

Here is another example against the /users endpoint, this time we will filter the results to include only the users whose user name includes jay:
The json object result is:

```json
[
  {
    "first_name": "Jay",
    "id": 22,
    "last_login": "2015-10-28T08:05:51.925743",
    "last_name": "Paz",
    "resource_uri": "/api/v1/users/22/",
    "username": "jay7958"
  },
  {
    "first_name": "",
    "id": 31,
    "last_login": "2015-10-13T11:44:32.533035",
    "last_name": "",
    "resource_uri": "/api/v1/users/31/",
    "username": "jay.paz"
  }
]
```

See Django Rest Framework’s documentation on interacting with an API for additional examples and tips.

### 3.2.3 Manually calling the API

Tools like Postman can be used for testing the API.

Example for importing a scan result:

- **Verb:** POST
- **URI:** http://localhost:8080/api/v2/import-scan/
- **Headers tab:** add the authentication header
  - Key: Authorization
  - Value: Token c8572a5adf107a693aa6c72584da31f4d1f1dcff
- **Body tab**
  - select “form-data”, click “bulk edit”. Example for a ZAP scan:

```plaintext
enagement:3
verified:true
active:true
lead:1
tags:test
scan_date:2019-04-30
scan_type:ZAP Scan
minimum_severity:Info
skip_duplicates:true
close_old_findings:false
```

- **Body tab**
  - Click “Key-value” edit
  - Add a “file” parameter of type “file”. This will trigger multi-part form data for sending the file content
  - Browse for the file to upload
• Click send

3.3 DefectDojo settings.py variables

For more info on custom settings and use of custom settings during development, please see: [settings.py documentation](https://github.com/DefectDojo/django-DefectDojo/blob/master/dojo/settings/settings.py) and [extra settings](https://github.com/DefectDojo/django-DefectDojo/blob/master/docker/extra_settings/README.md)

**Note:** To complete

- **DDAUTHORIZED_USERS_ALLOWCHANGE:** Grants Active users (e.g regular users) the ability to perform changes for the Products they are authorized.
- **DDAUTHORIZED_USERS_ALLOWDELETE:** Grants Active users (e.g regular users) delete powers for the Products they are authorized.
- **DD_SITE_URL:**
- **DDDEBUG:**
- **DDDJANGO_METRICS_ENABLED:**
- **DDLOGIN_REDIRECTURL:**
- **DDDJANGADMIN_ENABLED:**
- **DDSESSION_COOKIE_HTTPONLY:**
- **DD_CSRF_COOKIE_HTTPONLY:**
- **DD_SECURE_SSL_REDIRECT:**
- **DD_SECURE_HSTS_INCLUDE_SUBDOMAINS:**
- **DD_SECURE_HSTS_SECONDS:**
- **DDSESSION_COOKIE_SECURE:**
- **DD_CSRF_COOKIE_SECURE:**
- **DD_SECURE_BROWSER_XSS_FILTER:**
- **DD_SECURE_CONTENT_TYPE_NOSNIFF:**
- **DD_TIME_ZONE:**
- **DD_LANG:**
- **DDWKHTMLTOPDF:**
- **DD_TEAM_NAME:**
- **DDADMINS:**
- **DD_PORT_SCAN_CONTACT_EMAIL:**
- **DD_PORT_SCAN_RESULT_EMAIL_FROM:**
- **DD_PORT_SCAN_EXTERNAL_UNIT_EMAIL_LIST:**
- **DD_PORT_SCAN_SOURCE_IP:**
- **DD_WHITENOISE:**
• DD_TRACK_MIGRATIONS:
• DD_SECURE_PROXY_SSL_HEADER:
• DD_TEST_RUNNER:
• DD_URL_PREFIX:
• DD_ROOT:
• DD_LANGUAGE_CODE:
• DD_SITE_ID:
• DD_USE_I18N:
• DD_USE_L10N:
• DD_USE_TZ:
• DD_MEDIA_URL:
• DD_MEDIA_ROOT:
• DD_STATIC_URL:
• DD_STATIC_ROOT:
• DD_CELERY_BROKER_URL:
• DD_CELERY_BROKER_SCHEME:
• DD_CELERY_BROKER_USER:
• DD_CELERY_BROKER_PASSWORD:
• DD_CELERY_BROKER_HOST:
• DD_CELERY_BROKER_PORT:
• DD_CELERY_BROKER_PATH:
• DD_CELERY_TASK_IGNORE_RESULT:
• DD_CELERY_RESULT_BACKEND:
• DD_CELERY_RESULT_EXPIRES:
• DD_CELERY_BEAT_SCHEDULE_FILENAME:
• DD_CELERY_TASK_SERIALIZER:
• DD_FORCE_LOWERCASE_TAGS:
• DD_FOOTER_VERSION: Optionally pass a custom version string displayed in the footer of all pages (base.html template). Defaults to the version configured in django-DefectDojo/setup.py
• DD_MAX_TAG_LENGTH:
• DD_DATABASE_ENGINE:
• DD_DATABASE_HOST:
• DD_DATABASE_NAME:
• DD_TEST_DATABASE_NAME:
• DD_DATABASE_PASSWORD:
• DD_DATABASE_PORT:
• DD_DATABASE_USER:
• DD_SECRET_KEY:
• DD_CREDENTIAL_AES_256_KEY:
• DD_DATA_UPLOAD_MAX_MEMORY_SIZE:
• DD_SOCIAL_AUTH_TRAILING_SLASH:
• DD_SOCIAL_AUTH_AUTH0_OAUTH2_ENABLED:
• DD_SOCIAL_AUTH_AUTH0_KEY:
• DD_SOCIAL_AUTH_AUTH0_SECRET:
• DD_SOCIAL_AUTH_AUTH0_DOMAIN:
• DD_SOCIAL_AUTH_AUTH0_SCOPE:
• DD_SOCIAL_AUTH_GOOGLE_OAUTH2_ENABLED:
• DD_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY:
• DD_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET:
• DD_SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_DOMAINS:
• DD_SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_EMAILS:
• DD_SOCIAL_AUTH_OKTA_OAUTH2_ENABLED:
• DD_SOCIAL_AUTH_OKTA_OAUTH2_KEY:
• DD_SOCIAL_AUTH_OKTA_OAUTH2_SECRET:
• DD_SOCIAL_AUTH_OKTA_OAUTH2_API_URL:
• DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_ENABLED:
• DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY:
• DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET:
• DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID:
• DD_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_RESOURCE:
• DD_SOCIAL_AUTH_GITLAB_OAUTH2_ENABLED:
• DD_SOCIAL_AUTH_GITLAB_KEY:
• DD_SOCIAL_AUTH_GITLAB_SECRET:
• DD_SOCIAL_AUTH_GITLAB_API_URL:
• DD_SOCIAL_AUTH_GITLAB_SCOPE:
• DD_SAML2_ENABLED:
• DD_SAML2_METADATA_AUTO_CONF_URL:
• DD_SAML2_METADATA_LOCAL_FILE_PATH:
• DD_SAML2_ASSERTION_URL:
• DD_SAML2_ENTITY_ID:
• DD_SAML2_DEFAULT_NEXT_URL:
• DD_SAML2_NEW_USER_PROFILE:
• DD_SAML2_ATTRIBUTES_MAP:
• DD_DISABLE_FINDING_MERGE:
• DD_AUTHORIZED_USERS_ALLOW_CHANGE:
• DD_AUTHORIZED_USERS_ALLOW_DELETE:
• DD_AUTHORIZED_USERS_ALLOW_STAFF:
• DD_SLA_NOTIFY_ACTIVE: Consider “Active” findings for SLA notifications.
• DD_SLA_NOTIFY_ACTIVE_VERIFIED_ONLY: Consider “Active” and “Verified” findings only for SLA notifications.
• DD_SLA_NOTIFY_WITH_JIRA_ONLY: Considers findings that have a JIRA issue linked.
• DD_SLA_NOTIFY_PRE_BREACH: Number of days to notify before breaching the SLA.
• DD_SLA_NOTIFY_POST_BREACH: Number of days to keep notifying after the SLA has been breached.
• DD_EMAIL_URL, default:
4.1 Defect Dojo Burp-Plugin

This is Burp Plugin to export findings directly to Defect Dojo.

4.1.1 Installation

In order for the plugin to work, you will need to have Jython set up in Burp Suite Pro. To use this plugin before it appears in the BApp Store you will need to do the following:

1. Go to Extender and select the Extensions tab
2. Click on Add, select Extension Type: to be Python and select the DefectDojoPlugin.py

4.1.2 Usage
5.1 How to write a DefectDojo parser

Please read https://github.com/DefectDojo/django-DefectDojo/tree/dev/doc/guide_to_parser_writing.md