

Django REST Framework (DRF) Secure Code Guidelines

Anuar Manuel Nader Meljem

University Master's Degree in Cybersecurity and Privacy
Business security

Pau del Canto Rodrigo

Victor Garcia Font

Andreu Pere Isern Deyà

January 2023



Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-CompartirIgual
[3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-sa/3.0/es/)

ACKNOWLEDGEMENTS

I would like to thank my family and friends for their love and support throughout this journey. Their constant belief in me has been a constant source of motivation and inspiration.

I am especially grateful to Laura for her support, understanding and patience as I worked on this project. Her love and laughter kept me going during the challenging times.

I would also like thank H., F., M., V. and T. for their love and support inspired me to move forward. They are a constant source of love and joy.

I am grateful to my colleagues, especially to Sergio and Giuseppe, for their assistance during this project.

I would also like to thank my supervisor, Pau del Canto Rodrigo, for his valuable guidance and support. Without his help, this project would not have been possible.

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Django REST Framework (DRF) Secure Code Guidelines</i>
Nombre del autor:	<i>Anuar Manuel Nader Meljem</i>
Nombre del consultor/a:	<i>Pau del Canto Rodrigo</i>
Nombre del PRA:	<i>Victor Garcia Font Andreu Pere Isern Deyà</i>
Fecha de entrega (mm/aaaa):	<i>01/2023</i>
Titulación o programa:	<i>Máster Universitario en Ciberseguridad y Privacidad</i>
Área del Trabajo Final:	<i>Seguridad empresarial</i>
Idioma del trabajo:	<i>Inglés</i>
Palabras clave	<i>Django, Django REST Framework, Seguridad</i>

Resumen del Trabajo

El propósito de esta tesis, titulada "Django REST Framework (DRF) Secure Code Guidelines", es proporcionar orientación sobre cómo escribir código seguro al utilizar la librería *Django REST Framework* (DRF) para el desarrollo de interfaz de programación de aplicaciones (API por sus siglas en inglés de *Application Program Interface*). La investigación busca responder las siguientes preguntas: ¿Cuáles son las mejores prácticas de código seguro en el contexto de DRF? ¿Cuáles son las vulnerabilidades comunes de las que deben estar al tanto los desarrolladores que utilizan DRF, y cómo se pueden evitar?

Para responder a estas preguntas, la metodología de este trabajo incluye la investigación de las mejores prácticas de código seguro en el contexto de DRF, así como el examen de vulnerabilidades comunes y cómo evitarlas. Los resultados de esta investigación incluyen un conjunto de recomendaciones o directrices para el código seguro con DRF, que incluyen una lista de verificación y una presentación.

Los principales hallazgos de este trabajo incluyen un resumen de la importancia del código seguro en general, así como consideraciones específicas para los desarrolladores que utilizan DRF. La tesis también proporciona sugerencias para futuras investigaciones o áreas de mejora en el área de código seguro con DRF.

En general, esta tesis busca proporcionar orientación valiosa y recomendaciones para escribir código seguro con DRF en el contexto del desarrollo de APIs web. Siguiendo estas directrices, los desarrolladores

pueden asegurar que sus aplicaciones web basadas en DRF sean seguras y resistentes a las vulnerabilidades comunes.

Abstract

The purpose of the work titled "Django REST Framework (DRF) Secure Code Guidelines" is to provide guidance on how to write secure code when using the Django REST Framework (DRF) for web application development. The research aims to address the following questions: What are the best practices for secure coding in the context of DRF? What are the common vulnerabilities that developers using DRF should be aware of, and how can they be prevented?

To answer these questions, the methodology for this work involves researching best practices for secure coding in the context of DRF, as well as examining common vulnerabilities and how to prevent them. The results of this research include a set of recommendations or guidelines for secure coding with DRF, including a cheat sheet and a presentation.

The main findings of this work include a summary of the importance of secure coding in general, as well as specific considerations for developers using DRF. The thesis also provides suggestions for further research or areas for improvement in secure coding with DRF.

Overall, this thesis aims to provide valuable guidance and recommendations for writing secure code with DRF in the context of web application development. By following these guidelines, developers can ensure that their DRF-based web applications are secure and resistant to common vulnerabilities.

Index

1	Introduction	1
1.1	Context and justification of the work.....	1
1.2	Objectives	2
1.3	Study of the ethical, social, and environmental impact.....	3
1.4	Methodology.....	4
1.5	Work Plan.....	6
1.6	Summary of products obtained	8
1.7	Brief description of the other chapters of the work	8
2	Python API development state of the art.....	9
2.1	Django.....	9
2.2	REST APIs	11
2.3	Django REST Framework	12
3	Web security state of the art.....	14
3.1	OWASP Top 10.....	16
3.2	OWASP API Security Top 10	20
3.3	CWE/SANS TOP 25.....	22
3.4	SSDLC	24
4	DRF Security Guideline.....	26
4.1	Framework Security	26
4.2	Vulnerable dependencies.....	34
4.3	Dependency confusion.....	36
4.4	Secure Code	39
4.5	Secure Deployment.....	53
5	OWASP Cheat Sheet.....	56
6	Conclusions and future work.....	59
7	Glossary	60
8	Bibliography	62
9	Annexes	68
9.1	Appendix 1 - Django REST Framework (DRF) Cheat Sheet.....	68

Figure list

Figure 1. Gantt Chart of PEC 1, PEC 2 and PEC 3.....	6
Figure 2. Gantt Chart of PEC 4, Video Delivery and Defense	7
Figure 3. Average cost of all cyber-attacks in the US and Europe from 2021 to 2022 by country [24].....	14
Figure 4. Leading risks to businesses worldwide from 2018 to 2022 [26]	15
Figure 5. Icons of OWASP Top 10 – 2021. Source: OWASP.....	16
Figure 6. Comparison of OWASP Top 10 across the years	18
Figure 7. CWE Top 25 2021 vs 2022. Source: MITRE	23
Figure 8. Phases of SSDLC. Source: Snyk	25
Figure 9. Django supported versions. Source: Django Download	28
Figure 10. Twilio’s approach to prevent Dependency Confusion. Source: Twilio Blog	37
Figure 11. Screenshot of how the new DRF CS will look like	56
Figure 12. Screenshot of the Contributing Guide to propose a new CS	57
Figure 13. Screenshot of the issue I created to propose the new CS.....	57
Figure 14. Screenshot of the issue I created to propose the new CS.....	58

1 Introduction

Cisco Global 2021 Forecast Highlights [1] estimates that, globally, IP traffic will reach 278.1 Exabytes per month in 2021. One hour of standard definition streaming of Netflix [2] consumes 1 GB. One Exabyte approx. equals to 1,073,741,824 GB. According to a 2019 Akamai traffic review, API calls represent 83 percent of web traffic [3]. As we can see, APIs are critical for Today's Internet.

Python is an open-source programming language created by a Dutch programmer named Guido van Rossum in 1991. It is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation [4]. It runs on almost every operating system available today, with the latest updated version being released in March 2022 [5]. Python usage among developers is 48% according to Statista and It is the 4th most used [6] programming language only behind JavaScript (1st), HTML/CSS (2nd) and SQL (3rd).

Django is a very popular Python-based web framework that handles most of the heavy lifting of building a website. According to Stack Overflow “2022 Developer Survey”, Django is the most popular Python based Web Framework [7]. Django follows a philosophy of “batteries-included”, masking much of the underlying complexity, allowing for rapid and secure development. Some of the common web development problems that Django solves are authentication, connecting to a database, logic, security, among others.

There are also thousands of third-party packages that add functionality to Django itself, the most prominent of which is Django REST Framework, which allows developers to transform any existing Django project into a powerful web API.

1.1 Context and justification of the work

There are no public Secure Code Guidelines for Django nor for Django REST Framework (DRF). The lack of such guidelines significantly increases the cognitive load of developers when making decisions and increases the risk of adding security bugs to DRF applications.

Having security guidelines available, generally, empower developers to write better code, especially since the software will not have security vulnerabilities.

The guideline created with this work will also help security professionals (auditors, consultants, etc.) and engineering managers have a guideline to use

as a reference when checking that the software produced by the developing team is secure.

The desired results are specific guidelines for secure coding of REST APIs using Django Rest Framework (DRF). This should make it easy for people writing software to make better security decisions.

1.2 Objectives

The intention of this project is to generate guidelines for secure coding of REST APIs using Django Rest Framework (DRF).

The general objective of the project is to create a document with clear and concise guidelines on how to code secure APIs using DRF. The resulting guideline should be clear enough for people who are writing DRF applications to follow and to provide them enough detail and context in case they want to understand why something is being recommended. The specific objectives to be achieved are:

1. Research and document the state of the art of web security in 2022.
2. Understand and briefly explain the history, architecture and operation of Django.
3. Understand and briefly explain the history, architecture and operation of DRF.
4. Understand, analyze, and explain the DRF settings that have an impact on the security of the application.
5. Understand, analyze, and explain the DRF functions that might be used in an unsafe way.
6. Provide guidance on what aspects of an DRF application should be considered and reviewed to have a secure application.
7. Provide guidance on recommended settings, functions, and architectural approaches to follow to have a secure application.

The first objective will help us have a better context about what we should protect in our web applications.

The following two objectives help have context of the specific technologies that are covered in this project. The minimum depth that must be obtained from each objective must be sufficient to understand how they work, and to be able to use and configure them in a safe way. This means that it is not necessary to understand everything related to the frameworks.

Objectives four and five are focused on the secure usage of the technologies. They will be required to perform a study of literature, documents and if needed even reading the code. Objectives six and seven is where the expected product will be created.

1.3 Study of the ethical, social, and environmental impact

Sustainability

This work should not have a significant impact in terms of sustainability because it has no impact on the environment. The only requirement it has is a running computer and in most cases the computer will already be running with the existence of this work or not.

The work will be 100% digital, hence there is no impact on printing and shipping several copies of this document.

The negative impacts of this work can be countered with the saved hours of forensic analysis, urgent meetings, crisis, etc. that will be averted if developers follow the guidelines described here.

Ethical behavior and Social Responsibility

This work has a positive impact in terms of ethical behavior and social responsibility because it empowers developers to write safer Django applications, hence reducing the number of hacks, data loss, attacks on individual privacy and other negative impacts of unsafe software.

It is important to mention that this work can also have a negative impact since malicious actors can also use this to write safer code for nefarious tools. Since this impact exists on every open source project, and the benefits are always greater than the damages.

A possible mitigation might be the usage of Ethical licenses as described in <https://ethicalsource.dev/licenses/>. Anyhow such licenses are not commonly used, can be complex to understand and they are not industry accepted yet. Finally, the usage of such a license will not really have an impact to prevent hackers, crackers, and other malicious actors from using the knowledge in this work. It may stop unethical companies from using this guideline, but this will need to be enforced with lawsuit or trial and it is not easy nor cheap.

Diversity, Gender, and Human Rights

This work should not have a negative impact in terms of diversity, gender, or human rights. Given that it will be available for everyone, it could be considered a positive impact to reduce gender inequality and general inequality to give opportunities for everyone to learn.

After the work is done, I will consider working in a Spanish translation so more people could have access to this knowledge.

This work will be done with a mindful consideration of all genders gender perspective. Some of the actions to assure this, are the following:

- Consider every author, without considering the gender of the author.
- Make a final product that could be used by every user, no matter their gender.
- Constantly and actively consider that this work will not reinforce biases or stereotypes.
- Use an inclusive language when possible. I will use they/them pronouns to refer to individuals, that way we are not having a bias towards any gender.

The previous considerations also apply to other aspects as race, origin, sexual orientation, religion, or any other aspect that might be a factor of discrimination. This work should be as inclusive as possible.

1.4 Methodology

There are several strategies that could be followed to develop this project, some of those are:

- **Interviews** – Interviewing subject matter experts (SME) and gathering information about security concerns, best practices, and guidelines to write secure code.
- **Literature review/research** – Searching and reading relevant documents about the topic.
- **Code review** – Reviewing the Django and DRF code base.
- **Historic review of vulnerabilities** – Searching and reviewing vulnerabilities that have appeared in Django and DRF.

The **main methodology used to develop the project will be literature review**, including books, blog posts, documentation and other relevant documents, analysis and documenting relevant findings. It will have a circular flow, that is, it will be repeated as many times as necessary until the objectives set are achieved.

The other methodologies will not be used, because they are either very time consuming (code review), have significant dependencies on people's availability (interviews) or will have a limited visibility of the problem and will not necessarily help to prevent future problems (historic review of vulnerabilities).

This project will be delivered in six stages:

- **PEC 1. Work plan** – Will include an index, description of the problem to solve, goals, methodology, list of tasks, work plan, state of the art and study of the ethical, social, and environmental impact.
- **PEC 2. Follow up delivery** – Will include, at least, a revised version of the previous delivery plus the sections “State of the art” and “Security state of the art”
- **PEC 3. Follow up delivery** – Will include, at least, a revised version of the previous delivery plus the sections “Security Guidelines”
- **PEC 4. Final delivery** – Will have all the sections.
- **Video Delivery** – A video, of a maximum of 15 minutes, in which I will clearly and concisely summarize the work done and present the final product.
- **Defense** – Defense of the work and final product.

In the first 3 stages, after delivery, I will receive feedback from my tutor. I will also ask coworkers and SME some time so they can provide feedback.

1.6 Summary of products obtained

The following products will be obtained from this work:

- ✓ **Work memory.** Document with the results of the investigation and the secure code guidelines for Django REST Framework. It includes information about the history, and secure configuration of the technologies covered.
- ✓ **Cheat Sheet.** Document with practical recommendations about the findings and how to write secure DRF code.
- ✓ **Presentation.** Document with a summary of the work. It is used to give a presentation about the work.
- ✓ **Virtual presentation.** A video with sound, of a maximum duration of 15 minutes, where the work carried out throughout the semester and the results obtained are clearly and concisely synthesized. The investigation and its results are explained, and the final product is presented.

1.7 Brief description of the other chapters of the work

In chapter 1, the purpose and objectives of the work are explained. A brief contextualization and a justification of the reason for the work done.

In chapter 2, the state of the art of the main technologies covered: Django, REST APIs and Django REST Framework (DRF).

In chapter 3, the state of the art of security. This will include context to understand why the secure code guidelines are relevant.

In chapter 4, the security guidelines are presented. This will be the most technical chapter and it will include all relevant findings.

In chapter 5, explanation of OWASP Cheat Sheet project and a cheat sheet for DRF.

In chapter 6, the conclusions of the work carried out according to the results obtained are documented.

2 Python API development state of the art

2.1 Django

2.1.1 What it is

Django is a very popular, free, and open source, high-level Python-based web framework that encourages rapid development and clean, pragmatic design by handling the challenging parts of building a website: authentication, connecting to a database, logic, security, and so on. [8]

The framework relies on the traditional model-view-controller (MVC) architecture, although in Django it is called a model-template-view (MTV) architecture [9].

It is a collection of three important components: Model, View, and Template.

The Model acts as a data access layer which handles the data in the database.

The View is used to execute the business logic and interact with a model to carry data and render a template.

The Template is a presentation layer which handles the User Interface part completely.

In Django a project is a web application. An application, or also known as an “app”, is a Python package that provides a set of features and may be reused in various projects. From the Django documentation [10]:

Applications include some combination of models, views, templates, template tags, static files, URLs, middleware, etc. They're generally wired into projects with the `INSTALLED_APPS` setting and optionally with other mechanisms such as `URLconfs`, the `MIDDLEWARE` setting, or template inheritance.

The most popular online directory of applications is <https://djangopackages.org/> which is also an open-source project. That website helps find information about open-source Django Apps, Frameworks, and other useful software [11].

2.1.2 History

Django was initially developed in 2005 and named after the guitarist Django Reinhardt. From the Django documentation site, we can get the history [9]:

World Online, a newspaper Web operation, is responsible for building intensive Web applications on journalism deadlines. In the fast-paced newsroom, World Online often has only a matter of hours to take a complicated Web application from concept to public launch.

...

In summer 2005, World Online decided to open-source the resulting software, Django. Django would not be possible without a whole host of open-source projects – Apache, Python, and PostgreSQL to name a few – and we're thrilled to be able to give something back to the open-source community.

Development of Django is supported by an independent foundation established as a 501(c)(3) non-profit in 2008. Like most open-source foundations, the goal of the Django Software Foundation is to promote, support, and advance its open-source project: in our case, the Django Web framework [12].

It is quite popular, some of the global companies that use Django are [9] [13] Instagram, Belvo, National Geographic, Mozilla, Spotify, Pinterest among others.

2.1.3 Security

Security on Django can be grouped in four areas:

1. **Framework** – This includes all the CVEs related to the Django Framework. This is handled by the [Django Security Team](#) and from a developer perspective the action required is to keep the Django package updated.
2. **Deployment** – This is related to the deployment of a Django project. It includes the Django settings and the security of the infrastructure, like Database, web server, application server, cache, or any other infrastructure, where the Django application is deployed. It included proper configuration of the Django settings, proper and secure configuration of the servers, and keeping the operating system and applications updated.
3. **Secure code** – This is related to the correct and safe usage of the Django Framework. Even though Django does a pretty good job in

providing functions that are safe, It is possible to write insecure code in Django.

4. **Business logic** – This is related to flaws in the design and implementation of an application that allow an attacker to have an unintended behavior, potentially causing a malicious goal. This type of vulnerabilities can happen on any programming language and framework since they are vulnerabilities in the design and implementation.

This document will cover all the areas described above and will go into detail in the last two aspects.

2.2 REST APIs

An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software [14].

For web APIs the dominant architectural pattern is known as REST (REpresentational State Transfer), first proposed in 2000 by Roy Fielding in his dissertation thesis [15]. Today it is common for websites to adopt an API-first approach of formally separating the back end from the front-end. This allows a website to use a dedicated JavaScript front-end framework, such as React or Vue, giving a better user experience, allowing a decoupled architecture, along with possible separation of duties like front-end developers and back-end developers [15].

One of the main advantages of having the backend expose API is that when the current front-end frameworks are eventually replaced by newer, the backend API can remain the same. No major rewrite is required.

Another major benefit is that one single API can support multiple front ends written in different languages and frameworks. For example, JavaScript for web front ends, Java for Android apps and Swift for iOS apps. All can use the same API communicating with the same underlying database back-end [15].

2.3 Django REST Framework

2.3.1 What it is

From the Django REST Framework (DRF) GitHub README.md [16]:

Django REST framework is a powerful and flexible toolkit for building Web APIs.

Some reasons you might want to use REST framework:

- *The Web browsable API is a huge usability win for your developers.*
- *Authentication policies including optional packages for OAuth1a and OAuth2.*
- *Serialization that supports both ORM and non-ORM data sources.*
- *Customizable all the way down - just use regular function-based views if you don't need the more powerful features.*
- *Extensive documentation, and great community support.*

DRF is an app for Django that mimics many of Django's traditional conventions and makes it a great and easy option for building a REST API on top of a Django project.

It is the most popular Django app, for creating REST APIs, in GitHub with 24.3k stars.

2.3.2 History

According to a 2014 Kickstarter campaign, DRF was initially released in January 2011 by Tom Christie and was almost exclusively developed in his personal time [17].

Tom Christie created a Kickstarter campaign to get funding for the release of version 3.3 of DRF. The campaign was launched on July 17th, 2014, and it was last updated in October of 2015 [18]. The campaign was very successful with 440 backers pledging £32,650. The original goal was £4,000 and the biggest stretch goal was £12,000 for an Admin interface.

On October 28th, 2015, version 3.3 of DRF was released, making the Kickstarter campaign achieve its goal [19]. As of the writing of this section, the last version released of DRF was 3.14 [20].

Django REST Framework is a powerful and accessible way to build web APIs [15]. Some of the global companies that use DRF are [21] Sentry, Retool, Robinhood, Belvo, O'Reilly Media, Digital Ocean, among others.

2.3.3 Security

Security on DRF is very similar to security on Django as described in the section [2.1.3 Security](#), the four areas described there apply here too.

Even the DRF Security Policy states that “Security issues are handled under the supervision of the Django security team” and it should be reported to them [22].

3 Web security state of the art

According to Hiscox “Cyber Readiness Report 2022” Cyber-attacks have intensified in the past 12 months. 48% of companies reported a cyber-attack in the past 12 months and the median cost of an attack is just under \$17,000. Anyhow, attacks can be way more expensive, for example the single largest cyber-attack suffered in 2021 in Germany cost \$3,400,000 USD [23].

The average cost in thousand U.S. dollars per country:

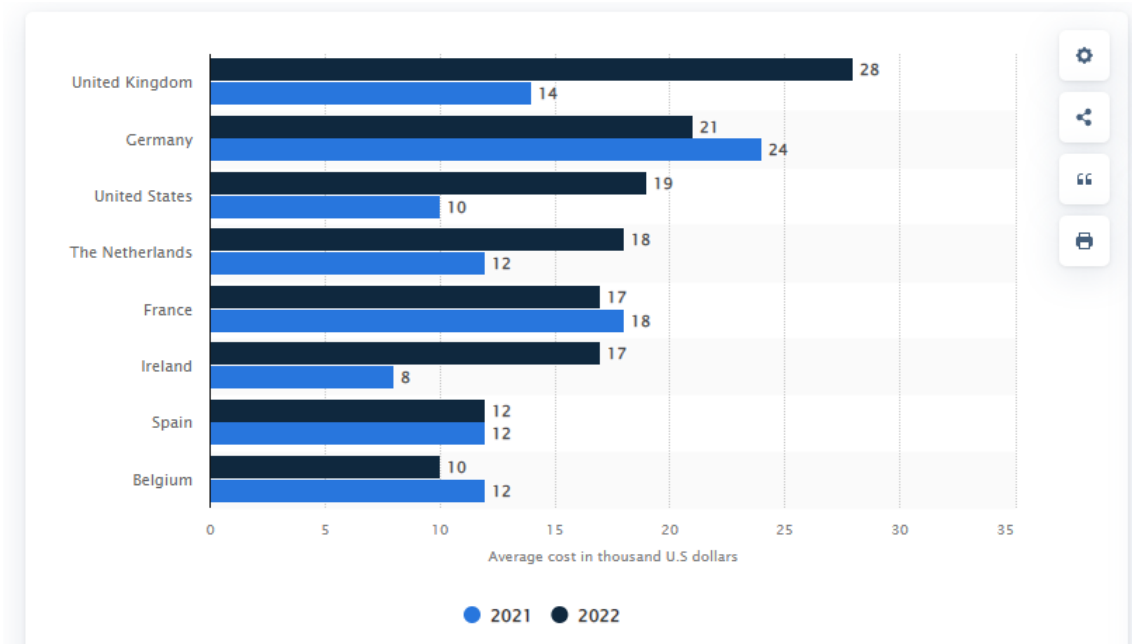


Figure 3. Average cost of all cyber-attacks in the US and Europe from 2021 to 2022 by country [24]

According to Imperva’s 2022 Report “Quantifying the Cost of API Insecurity”, who analyzed 117,000 cybersecurity incidents, API insecurity has annual losses of between \$41- 75 billion USD globally. Imperva estimated the total cyber loss, which represents any damage, loss, claim or cost directly or indirectly attributed to a cyber incident [25].

According to Allianz, Cyber Incidents is the leading risk for business worldwide in 2022 and has been quite important for the last 5 years:

Characteristic	2018	2019	2020	2021	2022
Cyber incidents (e.g. cyber crime, IT failure/outage, data breaches, fines and penalties)	40%	37%	39%	40%	44%
Business interruption (incl supply chain disruption)	42%	37%	37%	41%	42%
Natural catastrophes (e.g. storm, flood, earthquake, wildfire)	30%	28%	21%	17%	25%
Pandemic outbreak (e.g. health and workforce issues, restrictions on movement)	-	-	3%	40%	22%
Changes in legislation and regulation (e.g. trade wars and tariffs, economic sanctions, protectionism, Brexit, Euro-zone disintegration)	21%	27%	27%	19%	19%
Climate change/increasing volatility of weather	10%	13%	17%	13%	17%
Fire/explosion	20%	19%	20%	16%	17%
Market developments (e.g. volatility, intensified competition/new entrants, M&A, market stagnation, market fluctuation)	22%	23%	21%	19%	15%
Shortage of skilled workforce	-	9%	9%	8%	13%

Figure 4. Leading risks to businesses worldwide from 2018 to 2022 [26]

Reducing risk by having proper cyber security practices and having secure web applications, including APIs, is critical for any business additionally it can help them save a significant amount of money by preventing cybersecurity incidents.

The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. They are widely known for their OWASP Top 10 list, which highlights the most critical security concerns for web application security [27]. Sections [3.1 OWASP Top 10](#) and [3.2 OWASP API Security Top 10](#) will go into more detail of those lists.

The SANS Institute, officially the *Escal Institute of Advanced Technologies*, is a private U.S. for-profit company founded in 1989 that specializes in information security, cybersecurity training, and certifications. With the ongoing mission to empower cyber security professionals with the practical skills and knowledge they need to make our world a safer place. It is one of the most recognized institutions in the topic of cybersecurity training [28] [29]. Since 2009 SANS, with MITRE, have been making the list “CWE/SANS TOP 25 Most Dangerous Software Errors” [30]. Section [3.3 CWE/SANS TOP 25](#) will go into more detail of the SANS TOP 25 list.

Finally, a process that helps organizations develop secure software is called the *Secure Software Development Life Cycle (SSDLC)*. It includes activities such as security requirements gathering, security design, security testing, and security deployment. The benefits of using the SSDLC include improved security, reduced development costs, and improved software quality [31] [32]. Section [3.4 SSDLC](#) will go into detail of the SSDLC.

3.1 OWASP Top 10

The OWASP Top 10 is a standard awareness document for developers and web application security, created by the Open Web Application Security Project (OWASP) nonprofit foundation. It represents a broad consensus about the most critical security risks to web applications [33]. According to the OWASP Top 10 Archives, the first release was in 2003 and there have been other six releases: 2004, 2007, 2010, 2013, 2017 and 2021 [34].

The OWASP Top 10 2021 list [35] is:



Figure 5. Icons of OWASP Top 10 – 2021. Source: [OWASP](#)

The categories, with a brief explanation extracted from the OWASP site [33] are:

A01:2021-Broken Access Control – Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

A02:2021-Cryptographic Failures – Previously known as Sensitive Data Exposure, which was a broad symptom rather than a root cause. The renewed focus here is on failures related to cryptography which often leads to sensitive data exposure or system compromise.

A03:2021-Injection – The 33 CWEs mapped into this category have the second most occurrences in applications. Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. Cross-site Scripting is now part of this category too.

A04:2021 Insecure Design – Is a new category for 2021, with a focus on risks related to design flaws. Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” There is a difference between insecure design and insecure implementation. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

A05:2021 Security Misconfiguration – A vulnerability that occurs when the software is not properly configured from a security standpoint. Some examples of this are: Missing appropriate security hardening, unnecessary features are enabled or installed, default accounts and passwords, etc. The former category for XML External Entities (XXE) is now part of this category.

A06:2021 Vulnerable and Outdated Components – Was previously titled Using Components with Known Vulnerabilities.

A07:2021 Identification and Authentication Failures – Was previously Broken Authentication. Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.

A08:2021 Software and Data Integrity Failures – Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. Insecure Deserialization from 2017 is now a part of this larger category.

A09:2021 Security Logging and Monitoring – This category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Failures in this category can directly impact visibility, incident alerting, and forensics.

A10:2021 Server-Side Request Forgery (SSRF) – A vulnerability that occurs whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

Using as a base the document “2017-2003_Comparison” Christian Heinrich elaborated¹, I created a comparison list with every OWASP Top 10 release:

OWASP Top Ten	2003	2004	2007	2010	2013	2017	2021
Unvalidated Input	A1	A1 ^[9]	*	*	*	*	*
Buffer Overflows	A5	A5	*	*	*	*	*
Denial of Service	*	A9 ^[2]	*	*	*	*	*
Injection	A6	A6 ^[3]	A2	A1 ^[10]	A1	A1	A3
Cross Site Scripting (XSS)	A4	A4	A1	A2	A3	A7	A3 ^[25]
Broken Authentication and Session Management	A3	A3	A7	A3	A2	A2	A7 ^[22]
Insecure Direct Object Reference	*	A2	A4 ^[11]	A4	A4	A5 ^[20]	*
Cross Site Request Forgery (CSRF)	*	*	A5	A5	A8	*	*
Security Misconfiguration	A10	A10 ^{[3][5]}	*	A6	A5	A6	A5
Broken Access Control	A2	A2 ^[1]	A10 ^[3]	A8	A7 ^[6]	A5 ^[21]	A1
Unvalidated Redirects and Forwards	*	*	*	A10	A10	*	*
Information Leakage and Improper Error Handling	A7	A7 ^{[4][4]}	A6	A6 ^[8]	*	*	*
Malicious File Execution	*	*	A3	A6 ^[8]	*	*	*
Sensitive Data Exposure	A8	A8 ^{[8][5]}	A8	A7	A6 ^[17]	A3	A2 ^[22]
Insecure Communications	*	A10	A9 ^[7]	A9	*	*	*
Remote Administration Flaws	A9	*	*	*	*	*	*
Using Known Vulnerable Components	*	*	*	*	A9 ^{[10][9]}	A9	A6 ^[23]
Insecure Deserialization	*	*	*	*	*	A8	A8 ^[26]
XML External Entity (XXE)	*	*	*	*	*	A4	A5 ^[24]
Insufficient Logging & Monitoring	*	*	*	*	*	A10	A9 ^[27]
Identification and Authentication Failures	*	*	*	*	*	*	A7
Cryptographic Failure	*	*	*	*	*	*	A2
Insecure Design	*	*	*	*	*	*	A4
Software and Data Integrity Failures	*	*	*	*	*	*	A8
Security Logging and Monitoring Failures	*	*	*	*	*	A10	A9
Vulnerable and Outdated Components	*	*	*	*	A9	A9	A6
Server-Side Request Forgery	*	*	*	*	*	*	A10

Figure 6. Comparison of OWASP Top 10 across the years

Table notes:

- [1] Renamed “Broken Access Control” from T10 2003
- [2] Split “Broken Access Control” from T10 2003
- [3] Renamed “Command Injection Flaws” from T10 2003
- [4] Renamed “Error Handling Problems” from T10 2003
- [5] Renamed “Insecure Use of Cryptography” from T10 2003
- [6] Renamed “Web and Application Server” from T10 2003
- [7] Split “Insecure Configuration Management” from T10 2004
- [8] Reconsidered during T10 2010 Release Candidate (RC)
- [9] Renamed “Unvalidated Parameters” from T10 2003
- [10] Renamed “Injection Flaws” from T10 2007
- [11] Split “Broken Access Control” from T10 2004
- [12] Renamed “Insecure Configuration Management” from T10 2004

¹ Accessible at https://github.com/OWASP/Top10/tree/master/2017-2003_Comparison

- [13] Split "Broken Access Control" from T10 2004
- [14] Renamed "Improper Error Handling" from T10 2004
- [15] Renamed "Insecure Storage" from T10 2004
- [16] Renamed "Failure to Restrict URL Access" from T10 2010
- [17] Renamed "Insecure Cryptographic Storage" from T10 2010
- [18] Split "Insecure Cryptographic Storage" from T10 2010
- [19] Split "Security Misconfiguration" from T10 2010
- [20] Split "Broken Access Control" from T10 2013
- [21] "A4:2014-Insecure Direct Object References" and "A7:2013-Missing Function Level Access Control" merged into "A5:2017-Broken Access Control"
- [22] "A5:2017-Broken Authentication" now includes CWEs that are more related to identification failures and was renamed into "A7:2021-Identification and Authentication Failures"
- [23] "A3:2017-Sensitive Data Exposure" was renamed into "A02:2021-Cryptographic Failures", focusing on failures related to cryptography as it has been implicitly before
- [24] "A05:2021-Security Misconfiguration" now includes the former category for "A4:2017-XML External Entities (XXE)"
- [25] Cross-site Scripting is now part of this category
- [26] "A8:2017-Insecure Deserialization" is now a part of this larger category, named "A08:2021-Software and Data Integrity Failures"
- [27] "A09:2021-Security Logging and Monitoring Failures" was previously "A10:2017-Insufficient Logging & Monitoring". This category is expanded to include more types of failures. Failures in this category can directly impact visibility, incident alerting, and forensics
- [28] "A06:2021-Vulnerable and Outdated Components" was previously titled "Using Components with Known Vulnerabilities"

As we can see, there is a tendency change, and it makes sense when we see how technologies have changed from the early 2000s to today's technologies.

For example, buffer overflow no longer appears in the list, even that it was the risk rated as #1 in 2003, and that makes sense since those are vulnerabilities most common in the C and C++ languages and today very few web applications are written in those languages. The only common web technology written in C and C++, as of today, are web servers and those have already been "battle tested" for more than one decade.

Insecure communications are another risk that no longer appears in the Top 10 and that may be linked to the ease of use, along with the reduced costs of today's encryption technologies with comparison to early 2000s.

The risks that have been present on all releases of the Top 10 are "Injection", "Cross Site Scripting (XSS)", "Broken Authentication and Session Management", "Broken Access Control" and "Sensitive Data Exposure". All those risks have in common that they are more related with the proper design and coding of the web applications and not the technology used.

3.2 OWASP API Security Top 10

In 2019 OWASP released the first version of the “OWASP API Security Top 10”, a project very similar to “OWASP Top 10”, but with a sole focus on APIs. The “OWASP API Security Top 10 – 2019” risks, with a brief explanation, are [36] [37]:

API1:2019 Broken Object Level Authorization

Failures in the Authorization logic at the object level.

API2:2019 Broken User Authentication

Failures in the Authentication logic at the user level.

API3:2019 Excessive Data Exposure

Exposure of all object properties without considering their individual sensitivity.

API4:2019 Lack of Resources & Rate Limiting

Imposing no restrictions on the size or number of resources that can be requested by the client may lead to Denial of Service (DoS), but also leaves the door open to attacks such as brute force.

API5:2019 Broken Function Level Authorization

Failures in the Authentication logic at the function level. This may be caused by complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions.

API6:2019 Mass Assignment

Binding clients provided data to data models, without proper properties filtering based on an allow list, usually leads to Mass Assignment.

API7:2019 Security Misconfiguration

A vulnerability that occurs when the software is not properly configured from a security standpoint. Some examples of this are: open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.

API8:2019 Injection

Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query.

API9:2019 Improper Assets Management

APIs must be properly hosted and deployed, having an API version inventory play an important role. This vulnerability includes issues such as deprecated API versions and exposed debug endpoints.

API10:2019 Insufficient Logging & Monitoring

This category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Failures in this category can directly impact visibility, incident alerting, and forensics.

The following table is a recap of both lists:

Vulnerability	OWASP Top 10 – 2021	OWASP Top 10 API - 2019
Broken Access Control	Yes – A01:2021	Yes API1:2019 (object) API5:2019 (function)
Cryptographic Failures	Yes – A02:2021	Partially – API3:2019
Injection	Yes – A03:2021	Yes – API8:2019
Insecure Design	Yes – A04:2021	No
Security Misconfiguration	Yes – A05:2021	Yes – API7:2019
Vulnerable and Outdated Components	Yes – A06:2021	Partially – API9:2019
Identification and Authentication Failures	Yes – A07:2021	Yes – API2:2019
Software and Data Integrity Failures	Yes – A08:2021	No
Security Logging and Monitoring	Yes – A09:2021	Yes – API10:2019
Server-Side Request Forgery (SSRF)	Yes – A10:2021	No
Excessive Data Exposure	No	Yes – API3:2019
Lack of Resources & Rate Limiting	No	Yes – API4:2019
Mass Assignment	No	Yes – API6:2019
Improper Assets Management	No	Yes – API9:2019

3.3 CWE/SANS TOP 25

Since 2009 SANS, with MITRE, have been making the list “CWE/SANS TOP 25 Most Dangerous Software Errors” [30]. SANS last release of this list was in 2021² MITRE has released the 2022 list. The Top 25 security vulnerabilities, from the 2022 MITRE list, are [38]:

Rank	ID	Name	Score
1	CWE-787	Out-of-bounds Write	64.2
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11
4	CWE-20	Improper Input Validation	20.63
5	CWE-125	Out-of-bounds Read	17.67
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53
7	CWE-416	Use After Free	15.5
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56
11	CWE-476	NULL Pointer Dereference	7.15
12	CWE-502	Deserialization of Untrusted Data	6.68
13	CWE-190	Integer Overflow or Wraparound	6.53
14	CWE-287	Improper Authentication	6.35
15	CWE-798	Use of Hard-coded Credentials	5.66
16	CWE-862	Missing Authorization	5.53
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42
18	CWE-306	Missing Authentication for Critical Function	5.15
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85
20	CWE-276	Incorrect Default Permissions	4.84
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57
23	CWE-400	Uncontrolled Resource Consumption	3.56
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32

² <https://www.sans.org/top25-software-errors/> consulted on 27th October 2022

MITRE provide the following visual representation of the difference in 2021 and 2022 Top 25 lists:

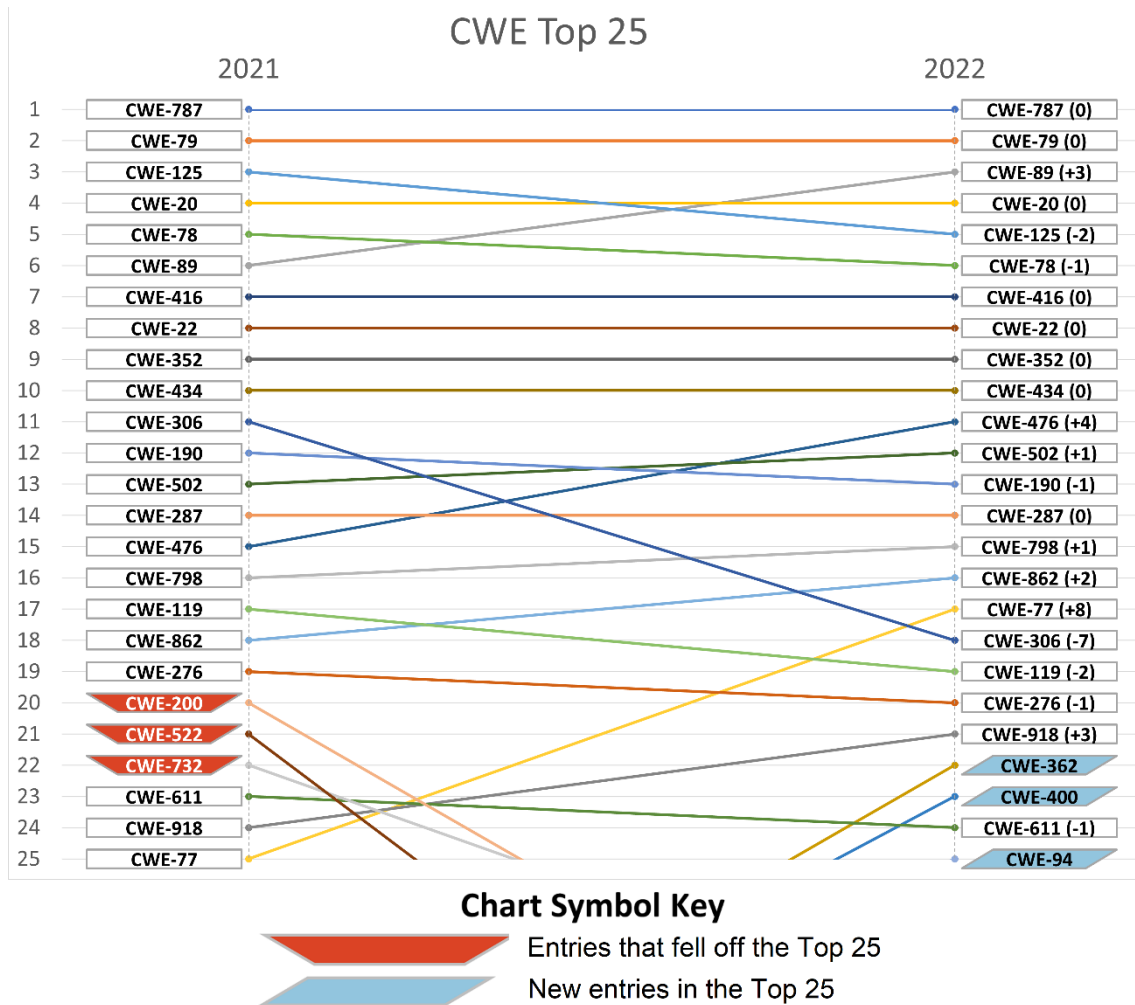


Figure 7. CWE Top 25 2021 vs 2022. Source: [MITRE](#)

As we can see, most CWEs remain present, they only shift a few positions up or down. The 3 new entries in the Top 25 are:

- **CWE-362** - Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
- **CWE-94** - Improper Control of Generation of Code ('Code Injection')
- **CWE-400** - Uncontrolled Resource Consumption

The 3 entries that fell off the Top 25 are:

- **CWE-200** - Exposure of Sensitive Information to an Unauthorized Actor
- **CWE-522** - Insufficiently Protected Credentials
- **CWE-732** - Incorrect Permission Assignment for Critical Resource

The next section will cover the SSDLC, which has stages that help reduce the chances of vulnerabilities being present in the application being developed.

3.4 SSDLC

The Secure Software Development Life Cycle (SSDLC) is a process that can be used by organizations to develop more secure software, it covers the entire development process. The SSDLC includes several security-focused activities that should be completed at each stage of the software development process. By following the SSDLC, organizations can improve the security of their software products and better protect themselves from potential attacks [39].

The SSDLC includes the following stages: requirement gathering and analysis (also known as planning), design, implementation (also known as build, development, or coding), testing (also known as verification), deployment (also known as release) and maintenance [39].

A brief explanation of each stage [40]:

Requirement gathering and analysis – The process of gathering and analyzing information about the software that is to be developed. Including expectations for the product and documenting them. This information can be used to identify potential security risks and to determine the appropriate security controls that should be implemented.

Design – The process of creating a detailed plan for the software. Threat modeling will be done to identify potential vulnerabilities. In this stage, it should also be specified how system authentication will be performed and where secure communication is required. The resulting plan should consider the security risks that have been identified and should include measures to mitigate those risks.

Implementation – The process of creating the actual software. During this stage, the security controls that were designed in the previous stage should be put in place. Here secure code guidelines should be provided to the development team. Ensuring the use of security libraries available in the framework and doing source code analysis (SAST) is part of this stage too.

Testing – The process of verifying that the software works as intended. This stage also includes testing the security controls to ensure that they are effective.

Deployment – The process of making the software available to users. This stage includes ensuring that the software is properly configured and that all security controls are in place.

Maintenance – The process of making changes to the software after it has been deployed. This stage includes ensuring that security controls are still in place and that the software remains compliant with security policies and procedures.

The stages of the SSDLC, with the security activities that could be performed are:

Secure Software Development Life Cycle (SSDLC)



Figure 8. Phases of SSDLC. Source: [Snyk](#)

The previous image does not include maintenance since some authors consider that to not be part of the scope or be included in Deployment.

This work will focus mainly on the stage **implementation** since that's when coding takes place. Secure Code Guidelines are critical for that stage since they empower developers to write safer code and to make less mistakes. Knowledge of this guide can benefit other stages too.

To have properly secure software, the team should do the full SSDLC, especially the earlier stages since that is where more information about the system, expectations and requirements is gathered.

4 DRF Security Guideline

Every programming language and framework has characteristics that make them unique.

Python is a high-level, general-purpose programming language. For example, some attacks like a Buffer Overflow (BOF), that are quite common in low-level languages such as C, will not happen in Python.

In this chapter I will only cover attacks that affect Django and Django REST Framework (DRF). This chapter will have the following sections:

Framework Security – An overview of the Django Security Policies and the Settings relevant for security in Django and DRF.

Vulnerable dependencies – An explanation of what dependencies are, how can we keep track of them and an explanation of some tools to keep track of the vulnerabilities and fix them.

Dependency confusion – An explanation of this type of attack, how it works and how to prevent it.

Secure Coding – This section covers the most common attacks on a Django and DRF app, maps them to the vulnerabilities described in the previous section and explains how to prevent them. It also covers some open source Static Application Security Testing (SAST) tools and briefly explains Business Logic attacks.

Secure Deployment – This section will provide a brief overview of how to safely deploy a DRF application. It will cover the settings that must be changed in a production environment, explain Django's system checks, and briefly explain how CI/CD can help improve the security posture of the application.

4.1 Framework Security

Let us remember, from the introduction, that Django follows a philosophy of “batteries-included”, that also includes that many security aspects are already taken care of by the Framework.

However, not everything is secure by default. In this section, I will explain the 3 aspects that are key to make an API developed with Django and Django REST Framework (DRF) secure:

1. **Django Security Policies** – Explain the Security Policies of the framework and project so we have context and understand what is expected from the development team of an application built on Django.
2. **Django Settings** – Explain the Django Settings relevant from a security point of view.
3. **DRF Settings** – Explain the DRF Settings relevant from a security point of view.

4.1.1 Django Security Policies

Django's development team has the following security policies [41]:

Reporting of Security Issues

Security issues should be reported via email (security@djangoproject.com) and they will reach Django Security team³. These emails can be encrypted with the public key ID 0xfcb84b8d1d17f80b.

Supported version

The Django team provides official security support for the following versions of Django:

- **The main development branch.** It is hosted on GitHub⁴ and it will become the next major release of Django. It is important to mention that security issues that only affect this version are fixed in public without going through the disclosure process.
- **The two most recent Django releases.**
- **Long-term support release.** Usually It is one at a time, anyhow there is an overlap, every 2 years, where 2 LTS releases are supported at the same time.

As of Today (November 12, 2022), the supported versions are:

- 3.2 – It is the long-term support (LTS) release.
- 4.0 – It is the second most recent Django release.
- 4.1 – It is the most recent Django release.
- 4.2 (WIP) – It is the current main development branch. As of today, it has a version of 4.2.a0⁵ (also known as 4.2.dev20221111083513). It will be released in April 2023.

³ <https://www.djangoproject.com/foundation/teams/#security-team>

⁴ <https://github.com/django/django/tree/main>

⁵ https://github.com/django/django/blob/main/django/_init_.py#L5

Finally, a diagram from the Django website with the supported versions [42]:

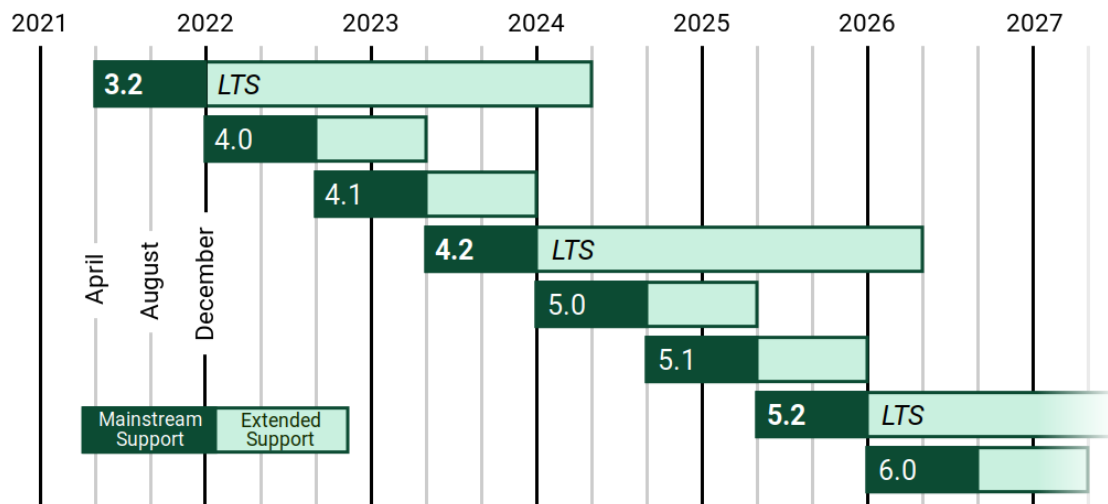


Figure 9. Django supported versions. Source: [Django Download](#)

The mainstream support includes security fixes, data loss bugs, crashing bugs, major functionality bugs in newly introduced features, and regressions from older versions of Django.

The extended support only includes security fixes and data loss bugs.

Disclosure of security issues

The Django security team process for taking a security issue from private discussion to public disclosure involves multiple steps. Approximately one week before public disclosure, two notifications are sent:

1. Notification on django-announce mailing list⁶ of the date and approximate time of the upcoming security release, as well as the severity of the issues.
2. Notification to a list of people and organizations who receive advance notification. This notification includes detailed information about the issue and remediation. The list is primarily composed of operating-system vendors and other distributors of Django. More information can be found on "[Who receives advance notification](#)".

On the day of disclosure, Django Security team will take the following steps:

1. **Apply the relevant patch** to Django's codebase.
2. **Issue the relevant release.** That is a new package on the Python Package Index, the Django website and tagging the new release in Django's git repository.

⁶ <https://docs.djangoproject.com/en/4.1/internals/mailling-lists/#django-announce-mailling-list>

3. **Post a public entry on the official Django development blog⁷**, with all the relevant information.
4. Post a **notice to the django-announce mailing list and oss-security@lists.openwall.com** mailing lists that link to the blog post.

The previous process can be considerably faster if it is believed to be particularly time-sensitive, for example due to a known exploit in the wild.

Finally, Django Security team classifies severity levels with the following logic:

High	Medium	Low
Remote code execution	Cross site scripting (XSS)	Sensitive data exposure
SQL injection	Cross site request forgery (CSRF)	Broken session management
	Denial-of-service attacks (DoS)	Unvalidated redirects/forwards
	Broken authentication	Issues requiring an uncommon configuration option

Django REST Framework has a very simple security policy [43]:

Security Policy

Reporting a Vulnerability

Security issues are handled under the supervision of the [Django security team](#).

Please report security issues by emailing security@djangoproject.com.

The project maintainers will then work with you to resolve any issues where required, prior to any public disclosure.

⁷ <https://www.djangoproject.com/weblog/>

4.1.2 Django Settings

According to Django documentation⁸, the core settings are over 180 different settings, this is very complex. It is not reasonable to ask every developer to review every setting and understand it.

The following list includes the settings I consider relevant from a security standpoint in a Django project that builds an API. For each setting, I give a brief description of what it controls and what value should it contain [44] [45] [46]:

DEBUG – A Boolean value that controls debug mode. It shall be False (default) in any production (or any Internet facing) system.

DEBUG_PROPAGATE_EXCEPTIONS – A Boolean value that controls if responses with status code 500 should propagate. It should be False (default) in any production (or any Internet facing) system.

SECRET_KEY – A secret key that is used to provide cryptographic signing, and should be set to a unique, unpredictable value. By default, it is empty, and it should never be hardcoded into the settings file.

SECRET_KEY_FALLBACKS – A list of fallback secret keys. This setting is used to allow rotation of the SECRET_KEY. Having multiple old key values in SECRET_KEY_FALLBACKS adds additional overhead to all checks that Do not match an earlier key. This list should be as small as possible, ideally empty.

It is important to understand and verify the settings of every module, either a 3rd party or a Django module like “django.contrib.auth” and the application used on the Django application. Not every module nor application has secure defaults.

A secret is any piece of information that gives access to information or resources. The most common types of secrets are Account credentials (username and password), API Keys, Passwords, SSH keys and Encryption keys.

It is important to never hardcode secrets on the settings file, or any other file, including the username or password to connect to the email provider (setting “EMAIL_HOST_PASSWORD”), database (setting “DATABASES”), cache (setting “CACHES”).

⁸ <https://docs.djangoproject.com/en/4.1/ref/settings/>

Secrets, including database URLs with username and password, should be handled by a Secret Manager like Hashicorp Vault⁹, AWS Secret Manager¹⁰, among others.

Other aspects that are important, from a security standpoint, of Django application configuration (settings) are:

Security Middleware

The setting **MIDDLEWARE** contains a list of middleware to use. The Middleware “*django.middleware.security.SecurityMiddleware*”¹¹ provides some security functionalities, anyhow none of them are very relevant for API projects since most of them are related to browser security and the security headers (HSTS, Cross origin, Content Type, etc.).

External Systems

The database should only be accessible to Django. It should never be exposed to the Internet. The same applies for the caching system.

HTTP Host header attacks protection

The settings “ALLOWED_HOSTS” and “USE_X_FORWARDED_HOST” should be properly set to prevent HTTP Host header attacks¹². If the Django application is deployed with a well architected approach¹³, it should not be a problem handled by Django and instead by the Application Load Balancer.

Logging

The settings “LOGGING” and “LOGGING_CONFIG” are related to how the application logs events. This is critical for any application. Generation, storage and analysis of logs should also be part of the well architected approach.

Secure Communications

Incoming traffic

Django has many settings to control HTTPS¹⁴ like “SECURE_HSTS_INCLUDE_SUBDOMAINS”, “SECURE_SSL_HOST”, among others. Anyhow It is recommended to not do HTTPS at the Django layer and handle that on something like a network load balancer, application load

⁹ <https://www.hashicorp.com/products/vault>

¹⁰ <https://aws.amazon.com/secrets-manager/>

¹¹ <https://docs.djangoproject.com/en/4.1/ref/middleware/#module-django.middleware.security>

¹² <https://docs.djangoproject.com/en/4.1/topics/security/#host-header-validation>

¹³ AWS Well-Architected: <https://aws.amazon.com/architecture/well-architected/>

Microsoft Azure Well-Architected Framework: <https://www.microsoft.com/azure/partners/well-architected>

¹⁴ <https://docs.djangoproject.com/en/4.1/topics/security/#ssl-https>

balancer, WAF or proxy [47]. It will be easier, and it is a better separation of responsibilities.

Outgoing traffic

Any outgoing traffic, for example to connect to the cache, the database, the email provider (Django setting “EMAIL_USE_TLS”), etc. should be done over an encrypted channel (TLS).

Many of the previous settings are related to a well architected application and are outside of the scope of this document.

Finally, many Django settings are not covered here since they are either not related to a regular API application (for example, cookies, CSRF or XSS prevention) or they should be handled by something else like an application load balancer, WAF or proxy (HTTP Headers, HTTPS, etc.). Anyhow those should be reviewed and properly configured if they are needed, for example for the admin panel.

4.1.3 DRF Settings

All the Django REST Framework (DRF) configuration is done under the namespace `REST_FRAMEWORK`, usually in the `settings.py` file, anyhow it can be in a different file if Django recognizes that file as part of the application settings. An example of the configuration is:

```
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': [
        'rest_framework.renderers.JSONRenderer',
    ],
    'DEFAULT_PARSER_CLASSES': [
        'rest_framework.parsers.JSONParser',
    ]
}
```

According to DRF documentation¹⁵ it has 40 different settings. The following list includes the settings I consider relevant from a security standpoint in a Django project that builds an API. For each setting, I give a brief description of what it controls and what value should it contain [15] [48] :

DEFAULT_AUTHENTICATION_CLASSES – A list of authentication classes that determines the default set of authenticators used when accessing the `request.user` or `request.auth` properties. In other words, what classes should be used to identify which user is authenticated.

Defaults are 'rest_framework.authentication.SessionAuthentication'¹⁶, 'rest_framework.authentication.BasicAuthentication'¹⁷, that means that by default it checks the session and basic authentication for the user. If more than 1 class is specified an OR operation is performed.

DEFAULT_PERMISSION_CLASSES – A list of permission classes that determines the default set of permissions checked at the start of a view. Permission must be granted by every class in the list, that is if more than 1 class is specified an AND operation is performed. Default is 'rest_framework.permissions.AllowAny'¹⁸, that means that by default every view allows access to everybody.

DEFAULT_THROTTLE_CLASSES – A list of throttle classes that determines the default set of throttles checked at the start of a view. Default is empty, that means that by default there is no throttling in place.

DEFAULT_PAGINATION_CLASS – The default class to use for queryset pagination. By default, pagination is disabled. Lack of proper pagination could lead to Denial of Service (DoS) in cases where there's a lot of data¹⁹.

The previous settings are part of the *API policy* settings, and they are applied to every *APIView* class-based view, or *@api_view* function-based view. Anyway, each view (either class or function based) can have a different authentication, permission, and throttling policy by configuring it on the view.

DRF also includes the authentication scheme “`rest_framework.authToken`”²⁰, it is highly recommended to not use it since it stores the keys in plaintext²¹.

¹⁵ <https://www.django-rest-framework.org/api-guide/settings/>

¹⁶ <https://www.django-rest-framework.org/api-guide/authentication/#sessionauthentication>

¹⁷ <https://www.django-rest-framework.org/api-guide/authentication/#basicauthentication>

¹⁸ <https://www.django-rest-framework.org/api-guide/permissions/#allowany>

¹⁹ For very large datasets Cursor Pagination is the best approach <https://www.django-rest-framework.org/api-guide/pagination/#cursorpagination>

²⁰ <https://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication>

4.2 Vulnerable dependencies

According to Synopsys “2022 Open Source Security and Risk Analysis Report”, 97% of the software projects reviewed contained open source components and 78% of the code in codebases was open source [49]. That means that most of the software that most companies use was not written by them. It is important to keep track of these dependencies because they can introduce security vulnerabilities or break compatibility with other software.

A software Bill of Materials (SBOM) is a list of all the software dependencies for a given piece of software. The National Telecommunications and Information Administration define a SBOM as [50]:

A Software Bill of Materials (SBOM) is a formal record containing the details and supply chain relationships of various components used in building software.

These components, including libraries and modules, can be open source or proprietary, free or paid, and the data can be widely available or access restricted.

The purpose of an SBOM is to help developers and security researchers understand what a piece of software is made of, and to identify any potential security vulnerabilities.

Python has many ways to manage dependencies such as *pip*, *Poetry*, *pyenv*, *setuptools*, *conda*, *easy_install*, *pipenv*, among others [51]. Anyhow, the most common and the current best practice is to use *pip* with a “requirements.txt” file. In order to keep it more organized, It is possible to have multiple requirements files [52] [53], for example:

- base.txt for packages used in all environments.
- local.txt for packages required for local development.
- ci.txt for packages required for the CI/CD pipeline.
- production.txt for the live production servers.

Since dependencies are much of the code in most codebases, it is critical to scan and make sure they do not have vulnerabilities. OWASP Top 10 – 2021 “**A06:2021-Vulnerable and Outdated Components**” is the category for the issue described in this section.

²¹ Discussed in [issue 4227](#) and verified in the [view](#) and [model](#).

There are several tools, known as Software composition analysis (SCA), to help with this, two of the most common are:

Github's Dependabot

According to Github's website [54]:

Dependabot is GitHub's supply chain security experience and makes it easy to find and fix vulnerable dependencies in your repository

Dependabot has powerful features such as:

- Automatic pull requests to fix security alerts as they happen.
- Easy to configure via a *dependabot.yml* file²².
- It is free for codebases hosted on Github.com

Snyk Open Source

According to Synk's website [55]:

Snyk Open Source provides a developer-first SCA solution, helping developers find, prioritize, and fix security vulnerabilities and license issues in open source dependencies.

Snyk Open Source has powerful features such as:

- Automatic pull requests to fix security alerts as they happen.
- Centralized reports.
- Integrate to the CLI and CI/CD tools.

Additionally, Snyk has a great free tool called Snyk Advisor²³ that gives valuable information about a package such as Popularity, Maintenance, Security and Community. This allows developers to compare packages and choose the best package for their needs, considering the security posture of the package.

It is important to mention that any Software composition analysis (SCA) that reviews the Django project dependencies and make sure there are no vulnerabilities in the dependencies used should be enough to prevent issues regarding Vulnerable dependencies.

²²<https://docs.github.com/en/code-security/dependabot/dependabot-version-updates/configuration-options-for-the-dependabot.yml-file>

²³ <https://snyk.io/advisor/python>

4.3 Dependency confusion

In February 2021, Alex Birsan published a blog post called “*Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies*”²⁴ where he explained a dependency chain attack that he named dependency confusion [56].

The attack works on several programming languages like JavaScript (NodeJS), Python and Ruby. It can be summarized as:

1. Find an internal package (e.g., shopify-cloud)
2. Publish a (malicious) package with the same name
3. Wait for the build pipeline to download your malicious package

For Python, the language which this work covers, Pip is the package-management system most widely used to install and manage software packages. When using the CLI argument “--extra-index-url”, a common way to reference an internal pip server for internal dependencies, works in the following way:

1. Checks whether library exists on the specified (internal) package index
2. Checks whether library exists on the **public** package index (PyPI)
3. Installs whichever version is found. If the package exists on both, it defaults to **installing from the source with the higher version number**.

That is dangerous since an attacker can create a malicious package with the same name, and a higher version number, as an internal package and execute code on the machine.

The problems of installing another package than the expected one are many, including loss of desired functionality, the software not behaving as expected, unknown bugs among many others. Anyhow, the biggest problem is that the undesired package may be malicious and do nefarious activities on the server such as steal secrets, data, delete files, launch attacks against other servers, etc.

According to Checkmarx researcher Yehuda Gelb [57]:

A worrying feature in pip/PyPi allows code to automatically run when developers are merely downloading a package. Also, this feature is alarming due to the fact that a great deal of the malicious packages we are finding in the wild use this feature of code execution upon installation to achieve higher infection rates.

²⁴ <https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610>

The previous is extremely dangerous since malicious package can cause damages just by being downloaded (`pip download <package_name>`)²⁵, even before execution or even installation.

There are ways to mitigate the execution of code when downloading a package, anyhow the best mitigation is to make sure all the packages used are not malicious.

According to Microsoft whitepaper “3 ways to mitigate risk when using private package feeds”, this attack can be prevented [58]:

For Python: Use the index-url option in pip’s configuration file or command line to specify the feed, overriding the default. Avoid the extra-index-url option, which is additive and may lead to having multiple indexes.

The previous recommendation should be used along an internal and trusted server and have that server include all packages needed.

A correct approach, especially for a company with many software projects is the one Twilio described in their blog post called “Dependencies, Confusions, and Solutions: What Did Twilio Do to Solve Dependency Confusion”, that is summarized in the following image [59]:


- 
- Naming Conventions For All Internal Packages
 - Block Proxying of Certain Packages
 - Internal Package Manager As Single Source
 - Restrict Deployed Hosts From Accessing the Registry
 - Controls For Laptop Access

Figure 10. Twilio’s approach to prevent Dependency Confusion. Source: [Twilio Blog](#)

A brief explanation of each step:

Naming Convention for All Internal Packages – There should be a clear convention for naming all internal packages. Something like “company_” that all internal packages follow. Do all the renaming as needed and help teams do this

²⁵ This behavior is not a bug but rather a feature in the pip design

if needed. Old packages not following the internal naming convention should be deleted.

Block Proxying of Certain Packages – Any public package that has a name collision with internal packages should be blocked. If possible, we should also block known malicious packages or any other package that does not follow the company guidelines (package health, license, etc.)

Internal Package Manager as Single Source – Every build should exclusively use the Internal Package Manager.

Restrict Deployed Hosts from Accessing the Registry – Deployed hosts should not access the registry (nor public nor internal) and only access the build artifact.

Controls For Laptop Access – Same restrictions, especially accessing only the internal registry, should apply to laptops if possible.

To prevent using malicious packages, either because of dependency confusion or just because the package is malicious, you can use the following tools that help detect malicious packages:

DataDog's Guarddog – GuardDog is a CLI tool that allows it to identify malicious PyPI packages. It runs a set of heuristics on the package source code (through Semgrep rules) and on the package metadata [60] [61].

Snyk Open Source Vulnerability Database – Snyk has a database about Open Source Vulnerabilities²⁶, including known packages with malware. An example of a package with malware can be found in <https://security.snyk.io/package/pip/aes44>.

Safety DB – Safety DB is a database, licensed under CC BY-NC-SA 4.0, of known security vulnerabilities in Python packages. The data is made available by pyp.io and synced with this repository once per month [62].

²⁶ <https://security.snyk.io/>

4.4 Secure Code

On Section “4.4.1 Secure Code” I will cover the most common attacks on a Django and Django REST Framework app, on what list they are present and the methods to prevent them.

On Section “4.4.2. Open Source SAST” I will cover some open source Static Application Security Testing (SAST) tools to detect the vulnerabilities covered in this section.

On Section “4.4.3. Business Logic” I will explain Business Logic attacks.

4.4.1 Secure Code

Django, along with Django REST Framework, follow the philosophy of “batteries included” so they take care of many things, security included. In order to give developers flexibility to build whatever solution they want, for example that could require a very specific SQL query, it also includes functions to have full control of certain situations. Nevertheless, the incorrect usage of such functions could lead to vulnerabilities.

Django Documentation has a page named “Security in Django” with an overview on some attacks and protections, I will briefly go over them [63]:

4.4.1.1 Cross site scripting (XSS) protection

This attack does not apply for APIs.

It is present in OWASP Top 10, inside **A03:2021-Injection** and in the 2022 MITRE list, finding #2 **Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**.

Django by default escape²⁷ specific characters which are particularly dangerous to HTML.

The ways to prevent it are:

1. If you are using templates, every attribute value should be quoted.
2. Be careful when using *is_safe* and *mark_safe* with custom template tags.
3. Do not turn *autoescape* off
4. Add Content Security Policy Headers

²⁷ <https://docs.djangoproject.com/en/4.1/ref/templates/language/#automatic-html-escaping>

4.4.1.2 Cross site request forgery (CSRF) protection

This attack does not apply for APIs.

No longer present in OWASP Top 10, last year it appeared was 2013.

Present in the 2022 MITRE list, finding #9 **Cross-Site Request Forgery (CSRF)**.

Django has built-in protection against most types of CSRF attacks. The important thing to prevent this is to deploy the Middleware “CsrfViewMiddleware” and be extra careful when marking a view with the `csrf_exempt` decorator since that will disable CSRF protections for that view.

4.4.1.3 SQL injection (SQLi) protection

It is present in OWASP Top 10, inside **A03:2021-Injection**, OWASP API Security Top 10, inside **API8:2019 Injection** and in the 2022 MITRE list, finding #3 **Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**.

Django by default protects SQL queries to injections by using query parameterization.

Django also provides developers power to write raw queries or execute custom SQL. Incorrect usage of these powers is the only case where a SQLi can happen.

Developers should avoid using the dangerous methods `raw()`²⁸, `extra()`²⁹ and custom SQL³⁰ (via `cursor.execute()`). If those methods are unavoidable for the application, no user input should be used. If that is unavoidable too, proper cleaning and filtering should be done on those inputs, anyhow the general recommendation is avoid doing this since there is a chance the filtering is not done properly or there is a way to bypass it.

²⁸ <https://docs.djangoproject.com/en/4.1/topics/db/sql/#executing-raw-queries>

²⁹ <https://docs.djangoproject.com/en/4.1/ref/models/querysets/#django.db.models.query.QuerySet.extra>

³⁰ <https://docs.djangoproject.com/en/4.1/topics/db/sql/#executing-custom-sql>

4.4.1.4 Clickjacking protection

This attack does not apply for APIs.

Not present in OWASP Top 10, nor 2022 MITRE list.

Django has clickjacking protection with the middleware “`django.middleware.clickjacking.XFrameOptionsMiddleware`”, that will add an HTTP Header to prevent the site being rendered inside a frame.

4.4.1.5 SSL/HTTPS

Django has several settings to enforce the use of HTTPS, but I suggest this is done on a separate layer of the network architecture, for example being handled by a proxy, a load balance, a CDN, a WAF, etc.

The proper configuration of HTTPS is outside the scope of this work.

What should be verified is that the site is only accessible by HTTPS, and it has secure configuration, sites like <https://www.ssllabs.com/> can help verify that.

4.4.1.6 Host header validation

Not present in OWASP Top 10, nor 2022 MITRE list.

This should be handled by the external layer of the network architecture, for example being handled by a proxy, a load balance, a CDN, a WAF, etc.

Anyhow, a proper configuration on Django is an extra security measure that could be done. This is easily prevented by setting the proper value on the settings field `ALLOWED_HOSTS`³¹ and Django will reject any request with a host not present in that list.

4.4.1.7 Referrer policy

This attack does not apply for APIs.

Not present in OWASP Top 10, nor 2022 MITRE list.

Django has the setting “`SECURE_REFERRER_POLICY`” that allows to configure the Referrer Policy header³², this might be relevant for privacy concerns regarding users browsing.

³¹ https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-ALLOWED_HOSTS

³² <https://docs.djangoproject.com/en/4.1/ref/middleware/#referrer-policy>

4.4.1.8 Cross-origin opener policy

This attack does not apply for APIs.

Not present in OWASP Top 10, nor 2022 MITRE list.

Django has the setting “SECURE_CROSS_ORIGIN_OPENER_POLICY” that allows to configure the Cross-Origin Opener Policy header³³.

4.4.1.9 Session security

This attack does not apply for APIs.

This is related to Cookie security³⁴ and it is not a good practice to have APIs using Cookies or sessions, especially for REST APIs since they shall be stateless. Every request shall be stateless and isolated from other requests.

Not present in OWASP Top 10, nor 2022 MITRE list.

4.4.1.10 User-uploaded content

This attack rarely applies to APIs.

Not present in OWASP Top 10, nor 2022 MITRE list.

If the API handles user-uploaded content, especial considerations³⁵ must be taken to make sure this is done safely. It is best if this could be done and handled by a specialized third party like a storage service.

4.4.1.11 Additional security topics

Django mentions the following considerations about security³⁶:

- ✓ *Make sure that your Python code is outside of the web server’s root. This will ensure that your Python code is not accidentally served as plain text (or accidentally executed).*
- ✓ *Take care with any user uploaded files.*
- ✓ *Django does not throttle requests to authenticate users. To protect against brute-force attacks against the authentication system, you may consider deploying a Django plugin or web server module to throttle these requests.*

³³https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-SECURE_CROSS_ORIGIN_OPENER_POLICY

³⁴ <https://docs.djangoproject.com/en/4.1/topics/http/sessions/#topics-session-security>

³⁵ Some of them described in <https://docs.djangoproject.com/en/4.1/topics/security/#user-uploaded-content>

³⁶ <https://docs.djangoproject.com/en/4.1/topics/security/#additional-security-topics>

- ✓ *Keep your SECRET_KEY, and SECRET_KEY_FALLBACKS if in use, secret.*
- ✓ *It is a good idea to limit the accessibility of your caching system and database using a firewall.*
- ✓ *Take a look at the Open Web Application Security Project (OWASP) Top 10 list which identifies some common vulnerabilities in web applications. While Django has tools to address some of the issues, other issues must be accounted for in the design of your project.*
- ✓ *Mozilla discusses various topics regarding web security. Their pages also include security principles that apply to any system.*

Additionally, the following are some protections worth discussing [44] [64].

4.4.1.12 Remote Code Execution (RCE) protection

It is present in OWASP Top 10, inside **A03:2021-Injection** and **A05:2021-Security Misconfiguration** in OWASP API Security Top 10, inside **API8:2019 Injection** and in the 2022 MITRE list, findings: #4 **Improper Input Validation**, #6 **Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**, #12 **Deserialization of Untrusted Data**, #17 **Improper Neutralization of Special Elements used in a Command ('Command Injection')** and #25 **Improper Control of Generation of Code ('Code Injection')**.

The following python built-ins functions execute code: `eval()`, `exec()` and `execfile()`. If the application makes use of this functions, no arbitrary user input should ever reach them³⁷.

The `pickle` module³⁸ is another unsafe module of Python. Only data where we have complete control, and there is no way for a user to tamper with should be unpickled. It is trivial to construct a malicious pickle data file which will execute arbitrary code during unpickling [65]. This includes the `pandas.read_pickle`³⁹ which is also unsafe.

If the Django app uses PyYAML⁴⁰, the method `load()` should be used carefully since it can also lead to RCE. The easiest mitigation is to use the method

³⁷ It's hard to properly sanitize this, especially if a wide array of values is allowed. A nice write-up of how a blocklist approach can be bypassed is present at https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html

³⁸ The pickle module implements binary protocols for serializing and de-serializing a Python object structure.

³⁹ https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_pickle.html

⁴⁰ <https://pyyaml.docsforge.com/>

`safe_load()`, if it is not possible, the parameter “Loader=yaml.SafeLoader” should be used.

4.4.1.13 Mass assignment protection

It is present in OWASP API Security Top 10, inside **API6:2019 Mass Assignment** and in the 2022 MITRE list, finding #4 **Improper Input Validation**.

The mass assignment vulnerability happens when database fields that are intended to be updated only through special processes are open to general update processes. When using ModelForms, always use `Meta.fields` (allow list approach) and never use `Meta.exclude` (blocklist approach) since it is easy to forget or make a mistake (for example, when updating the model in the future) that could lead to this type of vulnerability.

Additionally, do not use:

```
ModelForms.Meta.exclude = “__all__”
```

4.4.1.14 Identification and Authentication Failures protection

Failures in Authentication are present in OWASP Top 10, inside **A07:2021 Identification and Authentication Failures**, in OWASP API Security Top 10, inside **API2:2019 Broken User Authentication** and in the 2022 MITRE list, finding #14 **Improper Authentication** and finding #18 **Missing Authentication for Critical Function**.

Django REST Framework has the setting “**DEFAULT_AUTHENTICATION_CLASSES**”. Proper configuration of this setting is key to prevent this vulnerability.

The setting expects “A list or tuple of authentication classes”. The default value is:

```
['rest_framework.authentication.SessionAuthentication',  
 'rest_framework.authentication.BasicAuthentication']
```

Using classes without bugs is also key to prevent this vulnerability.

From DRF documentation [66]:

If no class authenticates, `request.user` will be set to an instance of `django.contrib.auth.models.AnonymousUser`, and `request.auth` will be set to `None`.

It is important to know that just checking if the request has a user is not enough since the user `AnonymousUser` might be set.

Finally, the authentication setting can also be overwritten on a class using the `APIView` class-based views and modifying the variable `authentication_classes`. On a function based view, it could be modified with the decorators `@api_view` and `@authentication_classes`.

4.4.1.15 Authorization Failures protection

Failures in Authorization are present in OWASP Top 10, inside **A01:2021-Broken Access Control**, in OWASP API Security Top 10, inside **API1:2019 Broken Object Level Authorization** and **API5:2019 Broken Function Level Authorization** and in the 2022 MITRE list, finding #16 **Missing Authorization** and finding #20 **Incorrect Default Permissions**.

Django REST Framework has the setting “**DEFAULT_PERMISSION_CLASSES**” proper configuration of this setting is key to prevent this vulnerability.

The setting expects “A list or tuple of permission classes”. The default value is:

```
[ 'rest_framework.permissions.AllowAny' ]
```

It is important to change the default value if the functions or view require some level of authorization. Using classes without bugs is also key to prevent this vulnerability.

From DRF documentation [67]:

Before running the main body of the view each permission in the list is checked. If any permission check fails, an exceptions.PermissionDenied or exceptions.NotAuthenticated exception will be raised, and the main body of the view will not run.

The authorization setting can also be overwritten on a class using the APIView class-based views and modifying the variable permission_classes.

On a function based view, it could be modified with the decorators @api_view and @permission_classes. All the above could be grouped as Function Level Authorization.

DRF also support object-level permission, which is used to determine if a user should be allowed to act on a particular object, which will typically be a model instance. For views where the method “get_object()” is override, there’s a need to explicitly call the “.check_object_permissions(request, obj)” method, otherwise this vulnerability will exist. An example of a proper get_object metod:

```
def get_object(self):
    obj = get_object_or_404(self.get_queryset(), pk=self.kwargs["pk"])
    self.check_object_permissions(self.request, obj)
    return obj
```

4.4.1.16 Resources & Rate Limiting protection

It is present in OWASP API Security Top 10, inside **API4:2019 Lack of Resources & Rate Limiting** and in the 2022 MITRE list, finding #23 **Uncontrolled Resource Consumption**.

Django REST Framework has the setting “**DEFAULT_THROTTLE_CLASSES**” proper configuration of this setting is key to prevent this vulnerability.

The setting expects “A list or tuple of throttle classes”. The default value is empty. For most installations It is important to change the default value.

Throttling is like permissions because it determines if a request should be authorized. The main difference is that throttling relies on temporary limits, based on time such as second, minute, hour or day. Authentication is a key component for a proper throttling strategy. Authentication can be either user based or anonymous usage and rely on the client IP addresses for throttling.

Finally, the throttling setting can also be overwritten on a class using the `APIView` class-based views and modifying the variable `throttle_classes`. On a function based view, it could be modified with the decorators `@api_view` and `@throttle_classes`.

4.4.1.17 Server-Side Request Forgery protection

It is present in OWASP Top 10, inside **A10:2021 Server-Side Request Forgery (SSRF)** and in the 2022 MITRE list, finding #21 **Server-Side Request Forgery (SSRF)**.

Preventing of SSRF attacks on applications using DRF is not different that on other languages or frameworks⁴¹. Proper validation of URLs should be done before making a request.

Django includes the class `URLValidator`⁴² and the validator `validate_ipv4_address()`⁴³.

⁴¹ OWASP provides a guide to prevent SSRF on https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

⁴² <https://docs.djangoproject.com/en/4.1/ref/validators/#django.core.validators.URLValidator>

⁴³ <https://docs.djangoproject.com/en/4.1/ref/validators/#validate-ipv4-address>

4.4.1.18 Excessive Data Exposure protection

It is present in OWASP API Security Top 10, inside **API3:2019 Excessive Data Exposure**.

The excessive data exposure vulnerability, in DRF, happens when the serializer displays too much information about a model. It is also more present on public endpoints.

To prevent this vulnerability, the team should do a thorough review of what fields are returned on each view.

Finally, if the serializer is inheriting from *ModelSerializer*, it is highly recommended to not use the `exclude` Meta property since a change in the model, for example adding a sensitive attribute, could lead to this vulnerability [68].

4.4.1.19 Summary tables

OWASP API Security Top 10

OWASP API Security Top 10	How to prevent it
API1:2019 Broken Object Level Authorization	On class-based views verify the variable <code>authentication_classes</code> . On a function based view, verify the decorators <code>@api_view</code> and <code>@authentication_classes</code>
API2:2019 Broken User Authentication	Verify the setting value <code>DEFAULT_AUTHENTICATION_CLASSES</code>
API3:2019 Excessive Data Exposure	Verify the data returned by the serializer
API4:2019 Lack of Resources & Rate Limiting	Verify the setting value <code>DEFAULT_THROTTLE_CLASSES</code> On class-based views verify the variable <code>throttle_classes</code> . On a function based view, verify the decorators <code>@api_view</code> and <code>@throttle_classes</code>
API5:2019 Broken Function Level Authorization	Verify the setting value <code>DEFAULT_PERMISSION_CLASSES</code> On class-based views verify the variable <code>authentication_classes</code> . On a function based view, verify the decorators <code>@api_view</code> and <code>@authentication_classes</code>
API6:2019 Mass Assignment	Don't use <code>meta.exclude</code> , instead use <code>meta.fields</code> Never use <code>ModelForms.Meta.fields = "__all__"</code>
API7:2019 Security Misconfiguration	Review Django Settings (especially <code>DEBUG</code> and <code>SECRET_KEY</code>) and the whole content of the settings file
API8:2019 Injection	Don't use any user controlled variables in the methods: <i>raw</i> , <i>extra</i> and <i>cursor.execute</i> (SQLi) <i>eval</i> , <i>exec</i> , <i>execfile</i> , <i>pickle.load</i> and <i>yaml.load</i> (RCE)
API9:2019 Improper Assets Management	Not related to code. Do proper asset management.
API10:2019 Insufficient Logging & Monitoring	Review the application is doing proper logging and there are alerts and monitors on those logs.

OWASP Top 10 2021

OWASP Top 10 2021	How to prevent it?
A01:2021-Broken Access Control	<p>Verify the setting value <code>DEFAULT_PERMISSION_CLASSES</code></p> <p>On class-based views verify the variable <code>permission_classes</code> On a function based view, verify the decorators <code>@api_view</code> and <code>@permission_classes</code></p>
A02:2021-Cryptographic Failures	Django can handle TLS, anyhow it's best to do it with a proxy or webserver before Django
A03:2021-Injection	<p>Don't use any user controlled variables in the methods: <i>raw</i>, <i>extra</i> and <i>cursor.execute</i> (SQLi) <i>eval</i>, <i>exec</i>, <i>execfile</i>, <i>pickle.load</i> and <i>yaml.load</i> (RCE)</p>
A04:2021 Insecure Design	Not related to code. Do a secure design
A05:2021 Security Misconfiguration	Review Django Settings (especially <code>DEBUG</code> and <code>SECRET_KEY</code>) and the whole content of the settings file
A06:2021 Vulnerable and Outdated Components	Not related to code. Update the components
A07:2021 Identification and Authentication Failures	<p>Verify the setting value <code>DEFAULT_AUTHENTICATION_CLASSES</code></p> <p>On class-based views verify the variable <code>authentication_classes</code> On a function based view, verify the decorators <code>@api_view</code> and <code>@authentication_classes</code></p>
A08:2021 Software and Data Integrity Failures	Not related to code. Do signature verification before updating or installing software.
A09:2021 Security Logging and Monitoring	Review the application is doing proper logging and there are alerts and monitors on those logs.
A10:2021 Server-Side Request Forgery (SSRF)	Do proper URL validation before making a request.

4.4.2 Open Source SAST

A Static Application Security Testing (SAST) tool is a software security analysis tool that can be used to identify vulnerabilities in software applications. SAST tools can help to improve the security of software applications by identifying potential security issues early in the software development process [69].

There are many different SAST options each will have their unique attributes such as speed, complexity to set up, quality of findings, price, among others. I will cover three open source solutions:

Bandit – Bandit is a tool designed to find common security issues in Python. To do this Bandit processes each file, builds an Abstract Syntax Tree (AST)⁴⁴ from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report. Bandit was originally developed within the OpenStack Security Project and later rehomed to PyCQA [70].

Semgrep – Semgrep is a fast, open-source, static analysis engine for finding bugs, detecting vulnerabilities in third-party dependencies, and enforcing code standards. Developed by “Return To Corporation” (usually referred to as r2c) and open-source contributors. It works based on rules⁴⁵, which can focus on security, language best practices, or something else. Creating a rule is easy and semgrep is very powerful. For Django there are 29 rules⁴⁶.

PyCharm Security – Pycharm-security is a plugin for PyCharm, or JetBrains IDEs with the Python plugin. The plugin looks at Python code for common security vulnerabilities and suggests fixes. It can also be executed from a Docker container⁴⁷. It has about 40 checks and some are Django specific [71].

All 3 tools run locally or in the build environment, there is no need to upload code.

Finally, each project should be confident they are using a SAST tool that’s updated, maintained, and has good findings. If possible, running a secondary SAST once a month is useful to detect gaps in findings.

⁴⁴ <https://docs.python.org/3/library/ast.html>

⁴⁵ Found at the registry: <https://semgrep.dev/explore>

⁴⁶ <https://semgrep.dev/p/django>

⁴⁷ <https://pycharm-security.readthedocs.io/en/latest/installation.html#installation-from-docker>

4.4.3 Business Logic

The National Vulnerability Database (NVD) categorizes Business Logic Errors as [72]:

CWE-840: Business Logic Errors – Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.

An easy way to understand business logic flaws is:

“Ways of using the legitimate processing flow of an application in a way that results in a negative consequence to the organization.”

Business logic flaws will rarely be detected by automatic scanners or can be detected with a “checklist security approach”. They require true understanding of the business, the system and how they interact.

They are often the most critical in terms of consequences, as they are deeply tied into the company's process, anyhow they Do not follow a pattern⁴⁸.

⁴⁸ Examples of this vulnerabilities can be found in Jeremiah Grossman's slides “Seven Business Logic Flaws that put your Website at Risk” found at https://owasp.org/www-pdf-archive/FROCo8_JeremiahGrossman_BizLogicFlaws.pdf

4.5 Secure Deployment

Special considerations should be taken before deploying an application to production, especially if the application should be accessible by everyone on the Internet.

In this section I will cover the basics for doing a secure deployment of a Django and Django REST Framework app.

It is important to mention that each application is unique and has its own architecture, business requirements, security requirements, technical capabilities, resources, among others. It is best to have their own process, anyhow this section should help as a starting point.

4.5.1 Settings

As explained in section “4.1.2 Django Settings”, the Django settings *DEBUG*, *DEBUG_PROPAGATE_EXCEPTIONS*, *SECRET_KEY*, *SECRET_KEY_FALLBACKS* are very important⁴⁹ for a production environment.

Secrets **should never be hardcoded** on the settings file or any other file since it is a poor practice because it can compromise security, make maintenance difficult, and hinder portability. It is recommended to handle secrets with a secret manager.

As explained in section “4.1.3 DRF Settings” DRF settings will hardly change. Anyhow, if there is an authentication, permission or throttle class that’s only used for local development or debugging, for example to allow all access, it should be removed before the application is deployed in production.

Settings such as *ALLOWED_HOSTS*, *CACHES*, *DATABASES*, *EMAIL_BACKEND* among others tend to be specific per each environment.

A good practice that some teams follow is to have separate files, all inside a folder called settings, for each environment. An example of that is:

- [settings/base.py](#) the basic common configuration used in all environments.
- [settings/local.py](#) with the specific settings required for local development. This usually includes debug features.
- [settings/ci.py](#) with the specific settings required for the CI/CD pipeline. Should be very similar to production, but it may have additional classes to allow testing or more verbose errors.

⁴⁹ The most important settings are *DEBUG* and *SECRET_KEY*.

- [settings/production.py](#) for the live production servers.

4.5.2 System Checks

Django Documentation has a page for the system check framework [73] that says:

The system check framework is a set of static checks for validating Django projects. It detects common problems and provides hints for how to fix them. The framework is extensible so you can easily add your own checks.

Checks can be triggered explicitly via the check command. Checks are triggered implicitly before most commands, including runserver and migrate. For performance reasons, checks are not run as part of the WSGI stack that is used in deployment. If you need to run system checks on your deployment server, trigger them explicitly using check.

It's a good practice to generate a list of requirements, especially security requirements, that are needed for each deployment and create a system check for each one of those. System checks should be part of a deployment process and in case a system check fails, the whole deployment should fail.

Django has good documentation on how to write new system check: <https://docs.djangoproject.com/en/4.1/topics/checks/>

4.5.3 CI/CD

An in-depth analysis of CI/CD is outside of the scope of this document since It is a huge topic. CI/CD stands for three things [74]:

- **Continuous integration.** Automated build and test processes ensure that code in the main branch is always production-quality.
- **Continuous delivery.** Every code change that passes the CI process are automatically published to a production-like environment. Deployment into the live production environment may require manual approval but is otherwise automated. The goal is that your code should always be ready to deploy into production.
- **Continuous deployment.** Code changes that pass the previous two steps are automatically deployed into production.

Not every environment with continuous delivery has continuous deployment since there may be regulatory or business requirements where a manual approval is required.

A proper CI/CD pipeline gives the business, and developers, enough confidence about their code and their deployment process that they can push new code multiple times per hour.

A robust CI/CD pipeline should also include security checks. For a Django application, the following are a good idea to have:

- **SAST analysis** – As explained in section “4.4.2 Open Source SAST”. Paid tools are also available.
- **Dependency checks** – The checks explained in sections “4.2 Vulnerable dependencies” and “4.3 Dependency confusion”.
- **System Checks** – As explained in section “4.5.2 System Checks”.

Additional steps could be added like DAST, an inhouse regression testing framework and any other tools the team feel needed.

Having a robust CI/CD pipeline requires many teams to communicate and work together. Security should never impose, or in any way force, their tools into a CI/CD pipeline since that would create friction and teams will try to find a way to bypass them if they keep adding friction.

The best approach is to add security tools in a gradual manner and have a timeline that adjusts with the business. For example, having a 6 month plan to add all 3 tools:

1. **Communicate** the plan with the teams and ask for feedback.
2. Start with the tools that will cause **lower friction** and less work for the teams. Probably creating a first iteration of custom system checks.
3. Continue with **SAST on warning mode** (not failing the pipeline).
4. Ask teams to work on the findings. **Provide help and resources** so they can fix the issues.
5. Integrate the **Dependency checks**.
6. **Repeat step 4**, but with the dependency checks.
7. **Establish and share a deadline** when the SAST will be a blocking factor in the pipeline.
8. **Establish and share a deadline** when the Vulnerability checks will be a blocking factor in the pipeline.

In every step keep an open mindset and a good communication channel with every other team involved. Having a weekly office hour for teams to reach out and solve questions is a good practice.

5 OWASP Cheat Sheet

The OWASP Cheat Sheet Series is a collection of documents that provide guidance on how to secure web applications and APIs. The series includes cheat sheets on a variety of topics, including secure coding practices, input validation and sanitization, authentication, and authorization, and more. It's another project of OWASP.

Each cheat sheet in the series provides a concise overview of a particular security concept, along with practical tips and examples to help developers implement secure coding practices in their projects. The cheat sheets are intended to be used as a reference resource, rather than a comprehensive guide, and are designed to be easily accessible and useful for developers at all levels of experience.[75]

As of December 2022, there is no Cheat Sheet for Django REST Framework. In this chapter I will present the Cheat Sheet I created and the experience of submitting it for review.

Cheat Sheet

I created a Cheat Sheet, and once it is accepted, it will look like:

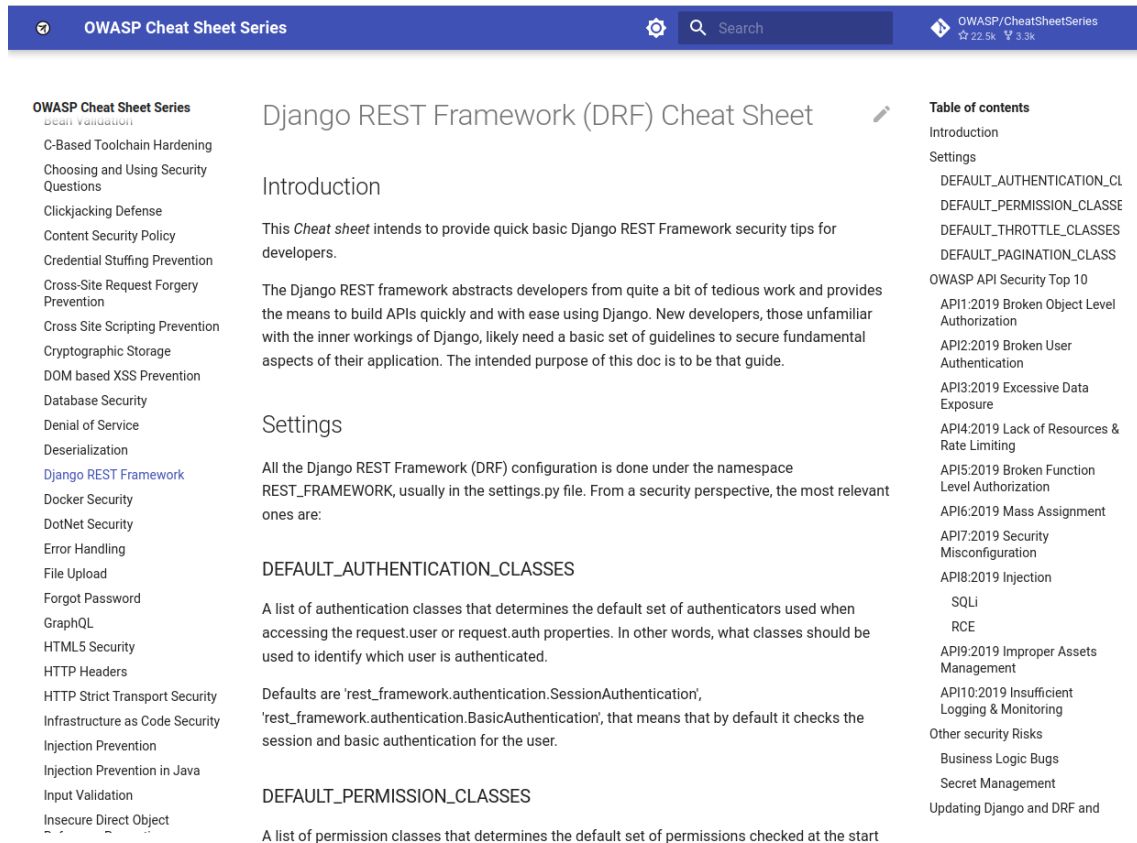


Figure 11. Screenshot of how the new DRF CS will look like

The complete text is in “9.1 Appendix 1 - Django REST Framework (DRF) Cheat Sheet”.

To submit a new Cheat Sheet, the project has a Contributing Guide⁵⁰ that I followed. From the guide:

To propose changes to the existing cheat sheets or the creation of a new one, the process is as follows:

1. Create an new [issue](#) using either:
 - o The `new_cheatsheet_proposal` template if you want to propose a new cheat sheet.
 - o The `update_cheatsheet_proposal` template if you want to modify a existing cheat sheet.
2. Once the issue has been discussed and approved:
 - i. Fork and clone this repository.
 - ii. Either:
 - o Create the cheat sheet using the [new cheat sheet template](#).
 - o Modify the target cheat sheet in case of an update or refactor.
 - iii. Submit your [Pull Request](#).
 - iv. Verify that the CI job [applied on your Pull Request](#) does not fail!
 - o If you believe they're failing due to something that's not your fault (such as another untouched file), add a comment in the Pull Request.

Figure 12. Screenshot of the Contributing Guide to propose a new CS

The issue I created⁵¹ to propose the new Cheat Sheet (CS) looked like:

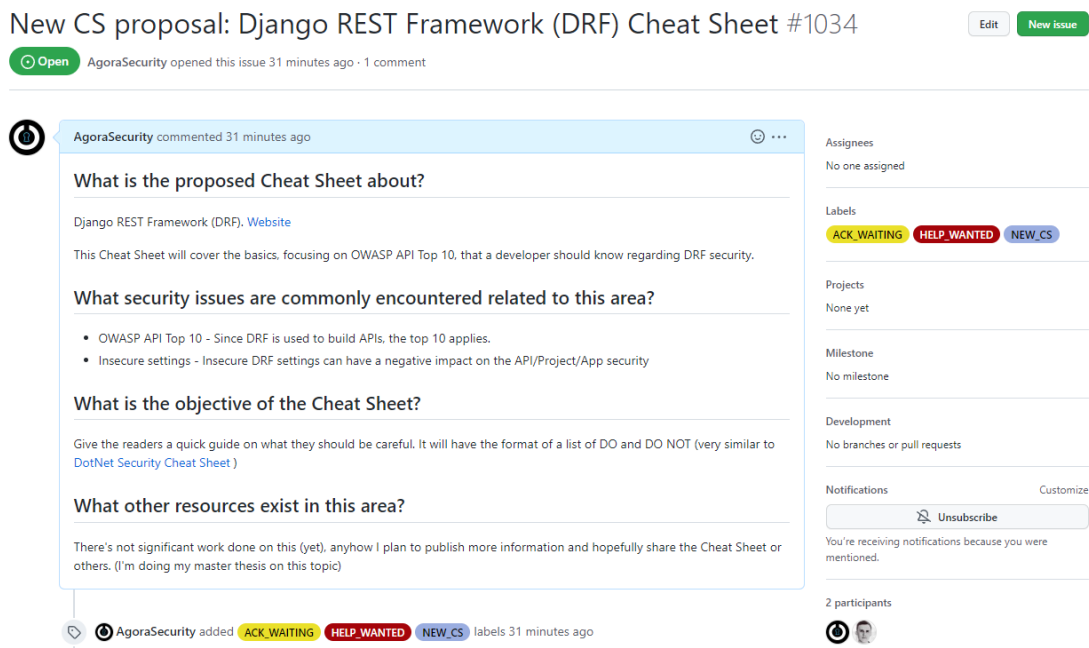


Figure 13. Screenshot of the issue I created to propose the new CS

⁵⁰ <https://github.com/OWASP/CheatSheetSeries/blob/master/CONTRIBUTING.md>

⁵¹ <https://github.com/OWASP/CheatSheetSeries/issues/1034>

On Dec 26th, 2022, I created a PR to introduce the first version of the draft of the DRF Cheat Sheet and it was accepted as a draft on January 2nd, 2023:

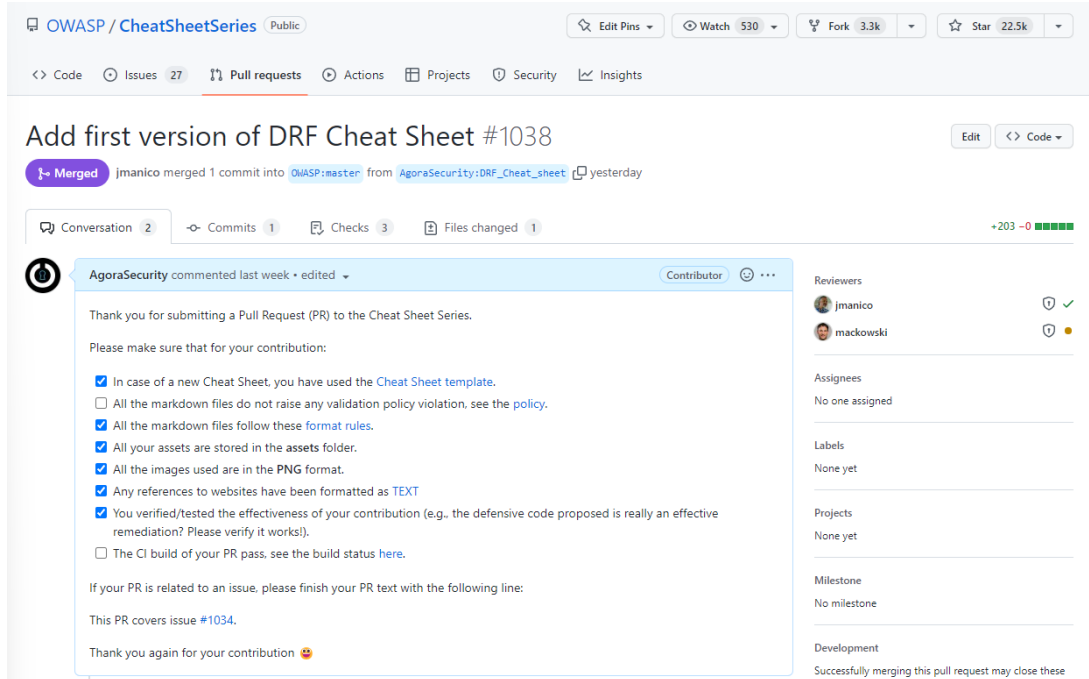


Figure 14. Screenshot of the issue I created to propose the new CS

On January 4th, 2023, the [PR](#) that published the draft was accepted and the “Django REST Framework (DRF) Cheat Sheet” is officially published⁵²:

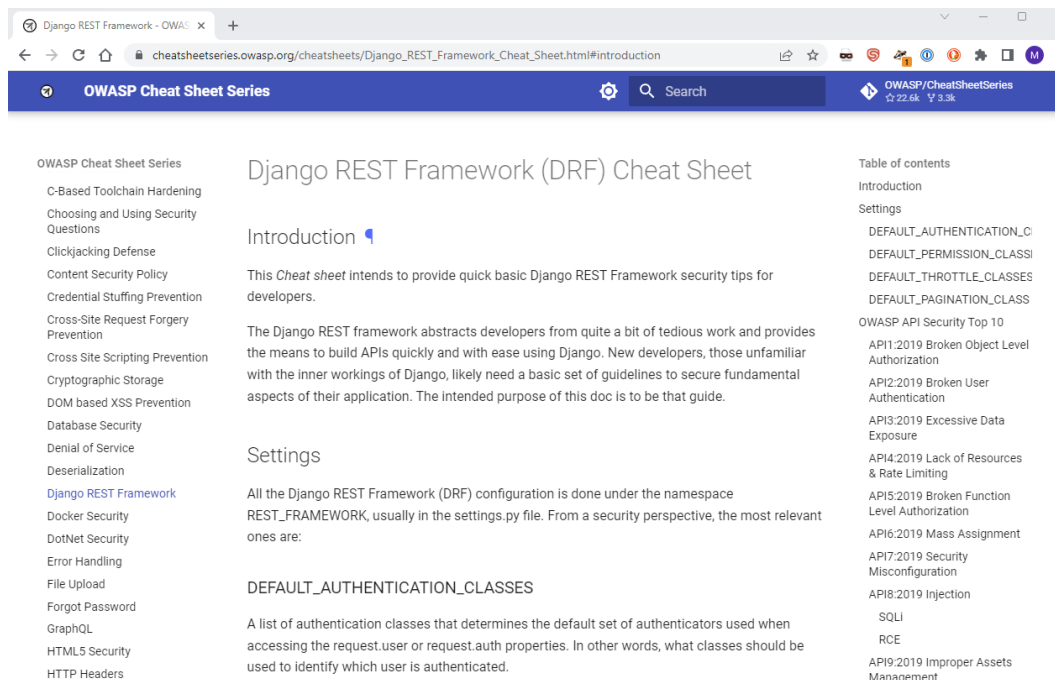


Figure 15. Screenshot of the issue I created to propose the new CS

⁵² At the URL:

https://cheatsheetseries.owasp.org/cheatsheets/Django_REST_Framework_Cheat_Sheet.html#introduction

6 Conclusions and future work

In conclusion, the work successfully achieved its objectives of providing guidance on how to write secure code when using the Django REST Framework (DRF). However, the timeline for this work was not without challenges. The main section, "DRF Security Guidelines," took longer to complete than expected, particularly the portion on "Secure Code." Additionally, there were a few changes made to the organization and presentation of the document, as is often the case with initial planning.

The methodology for this work was followed, except for interviews. Due to a lack of availability of experts, it was not possible to conduct interviews as part of the research process. Anyhow a more in-depth analysis of the literature was done along with in-depth research of publicly available information.

Despite these challenges, the research for this thesis resulted in the development of a set of recommendations or guidelines for secure coding with DRF, including a cheat sheet and a presentation. These guidelines provide valuable information for developers looking to ensure the security of their DRF-based APIs.

Additionally, the ethical, social, and environmental impacts of this work were all positive. By promoting the use of secure coding practices, this work helps to protect the privacy and security of users of web applications built with DRF. This can have a positive impact on society, as it helps to prevent data breaches and other security incidents that can have serious consequences.

There are several potential areas for future work in this field. One possibility is to create a vulnerable API where the guidelines could be tested and refined. Another option is to develop a course that teaches developers how to use these guidelines in practice. Additionally, translating this work into Spanish and other languages could help to make the guidelines more widely accessible to a global audience. Finally, creating guidelines for secure network segmentation, implementing a firewall, using a WAF, and restricting Django admin access using a firewall could further enhance the security of the system.

Overall, the "Django REST Framework (DRF) Secure Code Guidelines" provide a valuable resource for developers looking to write secure code with DRF in the context of web application development. With further research and development, these guidelines have the potential to become an even more valuable tool for ensuring the security of web applications built with DRF.

7 Glossary

API – An application programming interface (API) is a way for two or more computer programs to communicate with each other [14].

Buffer overflow (BoF) – Also known as buffer overrun is a type of vulnerability where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations. This often leads to a Denial of Service (DoS) condition, anyhow it can also lead to an attacker being able to execute code on the targeted system.

CVE – The Common Vulnerabilities and Exposures (CVE) system provides a reference-method for publicly known information-security vulnerabilities and exposures. MITRE's documentation defines CVE Identifiers (also called "CVE names", "CVE numbers", "CVE-IDs", and "CVEs") as unique, common identifiers for publicly known information-security vulnerabilities in publicly released software packages [75].

Denial of Service (DoS) – Is an attack meant to shut down a machine, network or system making it inaccessible to its intended users.

Django – Python-based free and open-source web framework that handles most of the heavy lifting of building a website. It follows the model–template–views (MTV) architectural pattern.

Django REST Framework – Django REST framework is a powerful and flexible toolkit for building Web APIs.

HTTP – The HyperText Transfer Protocol (HTTP) is the underlying network protocol that enables transfer of hypermedia documents on the Web, typically between a browser and a server so that humans can read them [76].

HTTPS – The secure variant of HTTP. It is an extension of HTTP, also referred to as HTTP over TLS.

MITRE – An American not-for-profit organization that manages federally funded research and development centers (FFRDCs) supporting various U.S. government agencies in the aviation, defense, healthcare, homeland security, and cybersecurity fields, among others [77].

Python – High-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Programming Language – A system of notation for writing computer programs. Some examples are C, Python, Java, Ruby and Rust.

REST API – The dominant architectural pattern for APIs and it means REpresentational State Transfer.

SANS Institute – Also known as just “SANS” is a private U.S. for-profit company founded in 1989 that specializes in information security, cybersecurity training, and certifications.

Secret – Credentials are often called “secrets”. Any private piece of information that acts as a key to unlock protected resources or sensitive information in tools, applications, etc. should be considered a secret. The most common types of secrets are Account credentials (username and password), API Keys, Passwords, SSH keys and Encryption keys.

Vulnerability - A weakness in a piece of computer software which can be used to access things one should not be able to gain access to [75].

TLS – Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network. The protocol can be used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTPS remains the most publicly visible [78].

YAML – YAML is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted. YAML stands for *yet another markup language* or *YAML ain't markup language* (a recursive acronym), which emphasizes that YAML is for data, not documents.

8 Bibliography

- [1] Cisco, "Global - 2021 Forecast Highlights," Cisco, 2021. [Online]. Available: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2021_Forecast_Highlights.pdf. [Accessed 9 October 2022].
- [2] Netflix, "How to control how much data Netflix uses," [Online]. Available: <https://help.netflix.com/en/node/87>. [Accessed 09 October 2022].
- [3] Akamai, "[state of the internet] / security Retail Volume 5, Issue 2," 02 2019. [Online]. Available: <https://www.akamai.com/site/it/documents/state-of-the-internet/state-of-the-internet-security-retail-attacks-and-api-traffic-report-2019.pdf>.
- [4] Wikipedia contributors, "Python (programming language)," Wikipedia, The Free Encyclopedia., 16 October 2022. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Python_\(programming_language\)&oldid=1116394427](https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1116394427). [Accessed 16 October 2022].
- [5] L. S. Vailshery, "Statista," Statista, 5 May 2022. [Online]. Available: Python - Statistics & Facts. [Accessed 09 October 2022].
- [6] Statista, "Most used programming languages among developers worldwide as of 2022," 2022. [Online]. Available: <https://www-statista-com.eu1.proxy.openathens.net/statistics/793628/worldwide-developer-survey-most-used-languages/>. [Accessed 09 October 2022].
- [7] Stack Overflow, "2022 Developer Survey," Stack Overflow, 2022. [Online]. Available: <https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe>. [Accessed 09 October 2022].
- [8] Django Project, "The web framework for perfectionists with deadlines | Django," Django Project, [Online]. Available: <https://www.djangoproject.com/>. [Accessed 14 October 2022].
- [9] Django Project, "FAQ: General," Django , [Online]. Available: <https://docs.djangoproject.com/en/4.1/faq/general/#why-does-this-project-exist>. [Accessed 14 October 2022].
- [10] Django Project, "Applications," [Online]. Available: <https://docs.djangoproject.com/en/4.1/ref/applications/>. [Accessed 14 October 2022].
- [11] Django Packages, "Django Packages : Frequently Asked Questions," Django Packages, [Online]. Available: <https://djangopackages.org/faq/>. [Accessed 14 October 2022].
- [12] Django Project, "About the Django Software Foundation," Django, [Online]. Available: <https://www.djangoproject.com/foundation/>. [Accessed 14 October 2022].
- [13] Trio Blog, "9 Examples of Companies Using Django in 2022," Trio , [Online]. Available: <https://www.trio.dev/blog/django-applications>. [Accessed 14 October 2022].
- [14] Wikipedia contributors, "API," Wikipedia, The Free Encyclopedia, 12 October 2022. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=API&oldid=1115721711>.

- [Accessed 14 October 2022].
- [15] W. S. Vincent, Django for APIs: Build web APIs with Python and Django, WelcomeToCode, 2022.
 - [16] encode, "encode/django-rest-framework: Web APIs for Django. 🍷," Github.com, [Online]. Available: <https://github.com/encode/django-rest-framework>. [Accessed 14 October 2022].
 - [17] T. Christie, "Posts - Django REST framework 3 by Tom Christie - Kickstarter," Kickstarter, [Online]. Available: <https://www.kickstarter.com/projects/tomchristie/django-rest-framework-3/posts>. [Accessed 14 October 2022].
 - [18] T. Christie, "Django REST framework 3 by Tom Christie - Kickstarter," Kickstarter, [Online]. Available: <https://www.kickstarter.com/projects/tomchristie/django-rest-framework-3>. [Accessed 14 October 2022].
 - [19] T. Christie, "3.3 Release," Kickstarter, [Online]. Available: <https://www.kickstarter.com/projects/tomchristie/django-rest-framework-3/posts/1391664>. [Accessed 14 October 2022].
 - [20] Django REST framework, "Django REST framework 3.14," Django REST framework, [Online]. Available: <https://www.django-rest-framework.org/community/3.14-announcement/>. [Accessed 14 October 2022].
 - [21] Django REST framework, "Funding - Django REST framework," Django REST framework, [Online]. Available: <https://fund.django-rest-framework.org/topics/funding/#our-sponsors>. [Accessed 14 October 2022].
 - [22] encode, "Security Policy," Github, [Online]. Available: <https://github.com/encode/django-rest-framework/security/policy>. [Accessed 16 October 2022].
 - [23] Hiscox Group, "22054 - Hiscox Cyber Readiness Report 2022-EN_0.pdf," 2022. [Online]. Available: https://www.hiscoxgroup.com/sites/group/files/documents/2022-05/22054%20-%20Hiscox%20Cyber%20Readiness%20Report%202022-EN_0.pdf. [Accessed 17 October 2022].
 - [24] Hiscox; Forrester Research, "Average costs of all cyber attacks in the United States and Europe from 2021 to 2022, by country," statista, [Online]. Available: <https://www-statista-com.eu1.proxy.openathens.net/statistics/1327147/median-cost-attacks-in-cyber-security-united-states-europe/>. [Accessed 17 October 2022].
 - [25] Imperva, "Quantifying the Cost of API Insecurity," Imperva, Marsh-McLennan, 2022.
 - [26] Allianz, "Leading risks to businesses worldwide from 2018 to 2022," statista, [Online]. Available: <https://www-statista-com.eu1.proxy.openathens.net/statistics/422171/leading-business-risks-globally/>. [Accessed 17 October 2022].
 - [27] Open Web Application Security Project® (OWASP), "Open Web Application Security Project® (OWASP)," Open Web Application Security Project® (OWASP), [Online]. Available: <https://owasp.org/>. [Accessed 17

- October 2022].
- [28] SANS Institute, "About SANS Institute," SANS Institute, [Online]. Available: <https://www.sans.org/about/>. [Accessed 17 October 2022].
 - [29] Wikipedia contributors, "SANS Institute," Wikipedia, The Free Encyclopedia., 7 October 2022 . [Online]. Available: https://en.wikipedia.org/w/index.php?title=SANS_Institute&oldid=1114709978. [Accessed 17 October 2022].
 - [30] Centro Criptológico Nacional, "CWE/SANS publica la lista de los 25 errores de software más peligrosos," CCN CERT, 10 January 2011. [Online]. Available: <https://www.ccn-cert.cni.es/seguridad-al-dia/noticias-seguridad/895-cwesans-publica-la-lista-de-los-25-errores-de-software-mas-peligrosos.html>. [Accessed 17 October 2022].
 - [31] Wikipedia contributors, "Systems development life cycle," Wikipedia, The Free Encyclopedia., 4 October 2022 . [Online]. Available: https://en.wikipedia.org/w/index.php?title=Systems_development_life_cycle&oldid=1114002735. [Accessed 17 October 2022].
 - [32] Wikipedia contributors, "Application security," Wikipedia, The Free Encyclopedia., 26 August 2022 . [Online]. Available: https://en.wikipedia.org/wiki/Application_security. [Accessed 17 October 2022].
 - [33] OWASP, "OWASP Top Ten," Open Web Application Security Project, [Online]. Available: <https://owasp.org/www-project-top-ten/>. [Accessed 23 October 2022].
 - [34] OWASP , "Top10/archives at master · OWASP/Top10," Github, [Online]. Available: <https://github.com/OWASP/Top10/tree/master/archives>. [Accessed 23 October 2022].
 - [35] B. Glas, "The Release of the OWASP Top 10:2021," OWASP, 24 September 2021. [Online]. Available: <https://www.owasptopten.org/the-release-of-the-owasp-top-10-2021>. [Accessed 23 October 2022].
 - [36] OWASP, "OWASP API Security Top 10 2019," OWASP, 2019.
 - [37] Open Web Application Security Project, "OWASP API Security Project," OWASP , [Online]. Available: <https://owasp.org/www-project-api-security/>. [Accessed 27 October 2022].
 - [38] The MITRE Corporation, "2022 CWE Top 25 Most Dangerous Software Weaknesses," MITRE , [Online]. Available: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html. [Accessed 27 October 2022].
 - [39] Snyk, "Secure Software Development Lifecycle (SSDLC)," Snyk Blog, [Online]. Available: <https://snyk.io/learn/secure-sdlc/>. [Accessed 28 October 2022].
 - [40] E. Johnson, "Securing the Software Development Lifecycle," SANS, 15 April 2015. [Online]. Available: <https://www.sans.org/blog/securing-the-software-development-lifecycle/>. [Accessed 28 October 2022].
 - [41] Django Project, "Django's security policies," Django Project, [Online]. Available: <https://docs.djangoproject.com/en/4.1/internals/security/>. [Accessed 12 November 2022].
 - [42] Django Project, "Download," Django Project, [Online]. Available:

- <https://www.djangoproject.com/download/>. [Accessed 12 November 2022].
- [43] T. Christie, "Security Policy," Github, 16 March 2022. [Online]. Available: <https://github.com/encode/django-rest-framework/security/policy>. [Accessed 12 November 2022].
- [44] D. R. G. a. A. R. Greenfeld, Two Scoops of Django 3.x, Feldroy Shop PS Feldroy LLC., 2021.
- [45] Django Project, "Settings," Django Project, [Online]. Available: <https://docs.djangoproject.com/en/4.1/ref/settings/>. [Accessed 13 November 2022].
- [46] W. Vincent, "Django Best Practices: Security," Learn Django, 24 October 2022. [Online]. Available: <https://learndjango.com/tutorials/django-best-practices-security>. [Accessed 13 November 2022].
- [47] AWS Amazon, "Application Load Balancer," AWS Amazon, [Online]. Available: <https://aws.amazon.com/elasticloadbalancing/application-load-balancer/>. [Accessed 13 November 2022].
- [48] Django REST Framework, "Settings," Django REST Framework, [Online]. Available: <https://www.django-rest-framework.org/api-guide/settings/>. [Accessed 14 November 2022].
- [49] Synopsys, "2022 Open Source Security and Risk Analysis Report," 2022. [Online]. Available: <https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2022.pdf>. [Accessed 14 November 2022].
- [50] National Telecommunications and Information Administration, "SBOM FAQ," 16 November 2022. [Online]. Available: https://www.ntia.gov/files/ntia/publications/sbom_faq_-_20201116.pdf. [Accessed 14 November 2022].
- [51] L. Irsigler, "A (soft) introduction to Python dependency management," Snyk, 14 September 2021. [Online]. Available: <https://snyk.io/blog/introduction-to-python-dependency-management/>. [Accessed 14 November 2022].
- [52] A. R. G. Daniel Roy Greenfeld, A Wedge of Django, Feldroy, 2021.
- [53] riptutorial, "Using multiple requirements files," riptutorial, [Online]. Available: <https://riptutorial.com/django/example/8561/using-multiple-requirements-files>. [Accessed 14 November 2022].
- [54] B. O'Shea, "How Dependabot empowers you to keep your projects secure," Github, 6 April 2022. [Online]. Available: <https://github.blog/2022-04-06-how-dependabot-empowers-you-to-keep-your-projects-secure/>. [Accessed 14 November 2022].
- [55] Snyk, "Open Source Security Management | SCA Tool | Snyk," Snyk, [Online]. Available: <https://snyk.io/product/open-source-security-management/>. [Accessed 14 November 2022].
- [56] A. Birsan, "Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies," Medium, 9 February 2021. [Online]. Available: <https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610>. [Accessed 15 November 2022].
- [57] Y. Gelb, "Automatic Execution of Code Upon Package Download on

- Python Package Manager," Checkmarx Security, 29 August 2022. [Online]. Available: <https://medium.com/checkmarx-security/automatic-execution-of-code-upon-package-download-on-python-package-manager-cd6ed9e366a8>. [Accessed 19 November 2022].
- [58] Microsoft, "3 ways to mitigate risk when using private package feeds," [Online]. Available: <https://azure.microsoft.com/mediahandler/files/resourcefiles/3-ways-to-mitigate-risk-using-private-package-feeds/3%20Ways%20to%20Mitigate%20Risk%20When%20Using%20Private%20Package%20Feeds%20-%20v1.0.pdf>. [Accessed 15 November 2022].
- [59] L. EPPALAGUDEM, "Dependencies, Confusions, and Solutions: What Did Twilio Do to Solve Dependency Confusion," Twilio Blog, 03 August 2021. [Online]. Available: <https://www.twilio.com/blog/avoiding-dependency-confusion-attacks>. [Accessed 18 November 2022].
- [60] DataDog, "Github's GuardDog," DataDog, [Online]. Available: <https://github.com/DataDog/guarddog>. [Accessed 19 November 2022].
- [61] C. T.-D. Ellen Wang, "Finding malicious PyPI packages through static code analysis: Meet GuardDog," DataDog, 15 November 2022. [Online]. Available: <https://securitylabs.datadoghq.com/articles/guarddog-identify-malicious-pypi-packages/>. [Accessed 19 November 2022].
- [62] pyupio, "safety-db," Github, 1 November 2022. [Online]. Available: <https://github.com/pyupio/safety-db>. [Accessed 26 November 2022].
- [63] Django Project, "Security in Django," Django Project, [Online]. Available: <https://docs.djangoproject.com/en/4.1/topics/security/>. [Accessed 20 November 2022].
- [64] W. S. Vincent, Django for Professionals, <https://djangoforprofessionals.com>, 2020.
- [65] Python Software Foundation, "pickle — Python object serialization," Python Software Foundation, 21 November 2022. [Online]. Available: <https://docs.python.org/3/library/pickle.html>. [Accessed 21 November 2022].
- [66] Django REST Framework, "Authentication," Django REST Framework, [Online]. Available: <https://www.django-rest-framework.org/api-guide/authentication/>. [Accessed 25 November 2022].
- [67] Django REST Framework, "Permissions," Django REST Framework, [Online]. Available: <https://www.django-rest-framework.org/api-guide/permissions/>. [Accessed 25 November 2022].
- [68] Django REST Framework, "Serializers," Django REST Framework, [Online]. Available: <https://www.django-rest-framework.org/api-guide/serializers/#specifying-which-fields-to-include>. [Accessed 25 November 2022].
- [69] mikecodase, "codase," codase, 8 February 2022. [Online]. Available: <https://codase.com/sast-tools-everything-you-need-to-know/>. [Accessed 26 November 2022].
- [70] PyCQA, "bandit," Github, 28 October 2022. [Online]. Available: <https://github.com/PyCQA/bandit>. [Accessed 26 November 2022].

- [71] pycharm-security, "PyCharm Python Security plugin," Read The Docs, [Online]. Available: <https://pycharm-security.readthedocs.io/en/latest/index.html>. [Accessed 26 November 2022].
- [72] MITRE, "CWE CATEGORY: Business Logic Errors," MITRE, [Online]. Available: <https://cwe.mitre.org/data/definitions/840.html>. [Accessed 26 November 2022].
- [73] Django Project, "System check framework," Django Project, [Online]. Available: <https://docs.djangoproject.com/en/4.1/topics/checks/>. [Accessed 27 November 2022].
- [74] Microsoft, "CI/CD for microservices architectures," Microsoft, [Online]. Available: <https://learn.microsoft.com/en-us/azure/architecture/microservices/ci-cd>. [Accessed 27 November 2022].
- [75] OWASP, "OWASP Cheat Sheet Series," OWASP, [Online]. Available: <https://cheatsheetseries.owasp.org/index.html>. [Accessed 20 December 2022].
- [76] Wikipedia contributors, "Common Vulnerabilities and Exposures," Wikipedia, The Free Encyclopedia., 15 July 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Common_Vulnerabilities_and_Exposures&oldid=1098348060. [Accessed 16 October 2022].
- [77] Mozilla Foundation, "HTTP," Mozilla Foundation, [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/HTTP>. [Accessed 16 October 2022].
- [78] Wikipedia contributors, "Mitre Corporation," Wikipedia, The Free Encyclopedia., 21 August 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Mitre_Corporation&oldid=1105777011. [Accessed 16 October 2022].
- [79] Wikipedia contributors, "Transport Layer Security," Wikipedia, The Free Encyclopedia., 17 September 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=1110721112. [Accessed 16 October 2022].

9 Annexes

9.1 Appendix 1 - Django REST Framework (DRF) Cheat Sheet

The source code of the Cheat Sheet is:

```
# Django REST Framework (DRF) Cheat Sheet

## Introduction

This Cheat sheet intends to provide quick basic Django REST Framework security tips for developers.

The Django REST framework abstracts developers from quite a bit of tedious work and provides the means to build APIs quickly and with ease using Django. New developers, those unfamiliar with the inner workings of Django, likely need a basic set of guidelines to secure fundamental aspects of their application. The intended purpose of this doc is to be that guide.

## Settings

All the Django REST Framework (DRF) configuration is done under the namespace REST_FRAMEWORK, usually in the settings.py file. From a security perspective, the most relevant ones are:

### DEFAULT_AUTHENTICATION_CLASSES

A list of authentication classes that determines the default set of authenticators used when accessing the request.user or request.auth properties. In other words, what classes should be used to identify which user is authenticated.

Defaults are 'rest_framework.authentication.SessionAuthentication', 'rest_framework.authentication.BasicAuthentication', that means that by default it checks the session and basic authentication for the user.

### DEFAULT_PERMISSION_CLASSES

A list of permission classes that determines the default set of permissions checked at the start of a view.

Permission must be granted by every class in the list. Default is 'rest_framework.permissions.AllowAny', that means that by **default every view allows access to everybody.**

### DEFAULT_THROTTLE_CLASSES
```

A list of throttle classes that determines the default set of throttles checked at the start of a view.

****Default is empty****, that means that by default there is no throttling in place.

DEFAULT_PAGINATION_CLASS

The default class to use for queryset pagination. ****Pagination is disabled by default.**** Lack of proper pagination could lead to Denial of Service (DoS) in cases where there's a lot of data.

OWASP API Security Top 10

The [OWASP API Security Top 10](<https://owasp.org/www-project-api-security/>) is a list of the most critical security risks for APIs, developed by the [Open Web Application Security Project (OWASP)](<https://owasp.org/>). It is intended to help organizations identify and prioritize the most significant risks to their APIs, so that they can implement appropriate controls to mitigate those risks.

This section is based on this. Your approach to securing your web API should be to start at the top threat A1 below and work down, this will ensure that any time spent on security will be spent most effectively spent and cover the top threats first and lesser threats afterwards. After covering the top 10 it is generally advisable to assess for other threats or get a professionally completed Penetration Test.

API1:2019 Broken Object Level Authorization

When using object-level permissions:

DO: Validate that the object can be accessed by the user using the method ``.check_object_permissions(request, obj)``. Example:

```
```python
def get_object(self):
 obj = get_object_or_404(self.get_queryset(), pk=self.kwargs["pk"])
 self.check_object_permissions(self.request, obj)
 return obj
```
```

DO NOT: Override the method ``.get_object()`` without checking if the request should have access to that object.

API2:2019 Broken User Authentication

DO: Use the setting value `DEFAULT_AUTHENTICATION_CLASSES` with the correct classes for your project.

DO: Have authentication on every non-public API endpoint.

DO NOT: Overwrite the authentication class on a class-based (variable ``authentication_classes``) or function-based (decorator ``authentication_classes``) view unless you are confident about the change and understand the impact.

API3:2019 Excessive Data Exposure

DO: Review the serializer and the information you are displaying.

If the serializer is inheriting from `ModelSerializer` DO NOT use the `exclude Meta` property.

DO NOT: Display more information than the minimum required.

API4:2019 Lack of Resources & Rate Limiting

DO: Configure the setting `DEFAULT_THROTTLE_CLASSES`.

DO NOT: Overwrite the throttle class on a class-based (variable ``throttle_classes``) or function-based (decorator ``throttle_classes``) view unless you are confident about the change and understand the impact.

EXTRA: If possible rate limiting should also be done with a WAF or similar. DRF should be the last layer of rate limiting.

API5:2019 Broken Function Level Authorization

DO: Change the default value (``rest_framework.permissions.AllowAny``) of `DEFAULT_PERMISSION_CLASSES`.

DO NOT: Use ``rest_framework.permissions.AllowAny`` except for public API endpoints.

DO: Use the setting value `DEFAULT_PERMISSION_CLASSES` with the correct classes for your project.

DO NOT: Overwrite the authorization class on a class-based (variable ``permission_classes``) or function-based (decorator ``permission_classes``) view unless you are confident about the change and understand the impact.

API6:2019 Mass Assignment

When using `ModelForms`:

DO: Use `Meta.fields` (allow list approach).

DO NOT: Use `Meta.exclude` (block list approach).

DO NOT: Use `ModelForms.Meta.fields = "__all__"`

API7:2019 Security Misconfiguration

DO: Setup Django settings `DEBUG` and `DEBUG_PROPAGATE_EXCEPTIONS` to False.

DO: Setup Django setting `SECRET_KEY` to a random value. Never hardcode secrets.

DO: Have a repeatable hardening process leading to fast and easy deployment of a properly locked down environment.

DO: Have an automated process to continuously assess the effectiveness of the configuration and settings in all environments.

DO: Ensure API can only be accessed by the specified HTTP verbs. All other HTTP verbs should be disabled.

DO NOT: Use default passwords

API8:2019 Injection

DO: Validate, filter, and sanitize all client-provided data, or other data coming from integrated systems.

SQLi

DO: Use parametrized queries.

TRY NOT TO: Use dangerous methods like `raw()`, `extra()` and custom SQL (via `cursor.execute()`).

DO NOT: Add user input to dangerous methods (`raw()`, `extra()`, `cursor.execute()`).

RCE

DO NOT: Add user input to dangerous methods (`eval()`, `exec()` and `execfile()`).

DO NOT: Load user-controlled pickle files. This includes the pandas method `pandas.read_pickle()`.

DO NOT: Load user-controlled YAML files using the method `load()`.

DO: Use the `Loader=yaml.SafeLoader` for YAML files.

API9:2019 Improper Assets Management

DO: Have an inventory of all API hosts and document important aspects of each one of them, focusing on the API environment (e.g., production, staging, test, development), who should have network access to the host (e.g., public, internal, partners) and the API version.

DO: Document all aspects of your API such as authentication, errors, redirects, rate limiting, cross-origin resource sharing (CORS) policy and endpoints, including their parameters, requests, and responses.

API10:2019 Insufficient Logging & Monitoring

DO: Log all failed authentication attempts, denied access, and input validation errors with sufficient user context to identify suspicious or malicious accounts.

DO: Create logs in a format suited to be consumed by a log management solution and should include enough detail to identify the malicious actor.

DO: Handle logs as sensitive data, and their integrity should be guaranteed at rest and transit.

DO: Configure a monitoring system to continuously monitor the infrastructure, network, and the API functioning.

DO: Use a Security Information and Event Management (SIEM) system to aggregate and manage logs from all components of the API stack and hosts.

DO: Configure custom dashboards and alerts, enabling suspicious activities to be detected and responded to earlier.

DO: Establish effective monitoring and alerting so suspicious activities are detected and responded to in a timely fashion.

DO NOT: Log generic error messages such as: `Log.Error("Error was thrown");` rather log the stack trace, error message and user ID who caused the error.

DO NOT: Log sensitive data such as user's passwords, API Tokens or PII.

Other security Risks

Below is a list of security risks for APIs not discussed in the OWASP API Security Top 10.

Business Logic Bugs

Any application in any technology can contain business logic errors that result in security bugs. Business logic bugs are difficult to impossible to detect using automated tools. The best ways to prevent business logic security bugs are to do threat model, security design review, code review, pair program and write unit tests.

Secret Management

Secrets should never be hardcoded. The best practice is to use a Secret Manager. For more information review OWASP [[Secrets Management Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html)](https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html)

Updating Django and DRF and Having a Process for Updating Dependencies

An concern with every application, including Python applications, is that dependencies can have vulnerabilities.

One good practice is to audit the dependencies your project is using.

In general, it is important to have a process for updating dependencies. An example process might define three mechanisms for triggering an update of response:

- Every month/quarter dependencies in general are updated.
- Every week important security vulnerabilities are considered and potentially trigger an update.
- In EXCEPTIONAL conditions, emergency updates may need to be applied.

The Django Security team has a information on [[How Django discloses security issues](https://docs.djangoproject.com/en/4.1/internals/security/#how-django-discloses-security-issues)](<https://docs.djangoproject.com/en/4.1/internals/security/#how-django-discloses-security-issues>).

Finally, an important aspect when considering if a new dependency should be added or not to the project is the "Security Health" of the library. How often it's updated? Does it have known vulnerabilities? Does it have an active community? etc. Some tools can help with this task (E.g. [[Snyk Advisor](https://snyk.io/advisor/python)](<https://snyk.io/advisor/python>))

SAST Tools

There are several excellent open-source static analysis security tools for Python that are worth considering, including:

Bandit - [[Bandit](https://bandit.readthedocs.io/en/latest/)](<https://bandit.readthedocs.io/en/latest/>) is a tool designed to find common security issues in Python. To do this Bandit processes each file, builds an Abstract Syntax Tree (AST) from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished

scanning all the files it generates a report. Bandit was originally developed within the OpenStack Security Project and later rehomed to PyCQA.

Semgrep - [Semgrep](<https://semgrep.dev/>) is a fast, open-source, static analysis engine for finding bugs, detecting vulnerabilities in third-party dependencies, and enforcing code standards. Developed by “Return To Corporation” (usually referred to as r2c) and open-source contributors. It works based on rules, which can focus on security, language best practices, or something else. Creating a rule is easy and semgrep is very powerful. For Django there are 29 rules.

PyCharm Security - [Pycharm-security](<https://pycharm-security.readthedocs.io/en/latest/index.html>) is a plugin for PyCharm, or JetBrains IDEs with the Python plugin. The plugin looks at Python code for common security vulnerabilities and suggests fixes. It can also be executed from a Docker container. It has about 40 checks and some are Django specific.

Related Articles and References

- [Django’s security policies](<https://docs.djangoproject.com/en/4.1/internals/security/>)
- [Security in Django](<https://docs.djangoproject.com/en/4.1/topics/security/>)