A Security Guide for Python Developers

kiwan
An Idera, Inc. Company

INDEX

Overview

A security footprint is the sum of a digital item's security requirements. For example, a program's security footprint would include all of the points where action is necessary to prevent data breaches or viruses.

A security footprint is the sum of a digital item's security requirements. For example, a program's security footprint would include all of the points where action is necessary to prevent data breaches or viruses.

Programming languages have security footprints, too. These footprints cover the points in which the language is vulnerable to malicious actors. Any program written in a given programming language must be designed with that language's security footprint in mind in addition to its own issues.

Python is no different. The language's security footprint includes three major points of risk:

Known Python Vulnerabilities

Within Python itself there are many known vulnerabilities. These are issues inherent to the language. Vulnerabilities typically occur because the language isn't specific enough. Python's flexibility is part of what makes it so popular, but it also allows developers to write unsecure code without realizing it.

For instance, Python has several vulnerabilities that occur when a program doesn't sanitize user inputs properly. These kinds of vulnerabilities allow users to force the program to take actions it shouldn't permit. However, you can easily fix these issues by making sure you're aware of known vulnerabilities within Python and coding appropriate preventative measures.

Direct Dependencies

As an open-source language, there are millions of packages available within Python. These packages can be used as direct dependencies, or pieces of code that are essential for a program to run.

Direct dependencies allow programmers to save time when writing code. They can rely on packages to handle basic tasks for them instead of reinventing the wheel. However, these dependencies also introduce potential security risks.

Since most programs use direct dependencies to perform basic operations, any problem within that package exists within the program, too. If a dependency's code includes a known vulnerability, the program running that dependency is at risk.

Indirect Dependencies

Packages can also become indirect dependencies. A package can have another package as a direct dependency. A program that uses the first package will automatically work with the second package even though it's not listed in the program's code. The program is indirectly dependent on the second package to function.

Just like direct dependencies, an indirect dependency can introduce security risks. However, indirect dependencies are harder to spot. Python programmers need to be aware of not only the packages they use in their programs but also the dependencies of those packages. For large programs, this can quickly balloon into dozens — or even hundreds — of packages to monitor. This line of linked packages is known as the software supply chain. An issue with any one part of the chain causes problems for every program and package downstream.

The 5 Most *Common Security Problems* in Python Projects

Within Python's security footprint, some issues are more common than others. There are five issues that make up the majority of Python security breaches, according to the **Open Web Application Security Project (OWASP)**. Recent studies suggest that these flaws are both the most dangerous to users and the easiest to fix with appropriate DevSecOps.

SQL and Command Injections

An SQL or command injection occurs when a hacker is able to input code into a program at some point and have the code pass through into its internal or backend systems. Malicious SQL injections will then force the program to do things like bypass authentication protocols or deliver confidential data to the hacker.

These injections are particularly easy in Python because of how the language handles user commands. Unless the program is written to "sanitize" user inputs, anything the user types into the program is directly passed to the command line. If the user chooses to input a string that the command line can read, it will then execute that code.

Sanitizing user inputs solves that problem. Sanitization is the process of ensuring that user input is only ever read as a string, not as code. It takes a minimal amount of extra effort, and it can significantly reduce your security risks.

Encoded Scripting

Python doesn't always handle **Unicode characters** the way you would expect. Hackers can use Unicode-encoded scripting to present strings that look like they accomplish one task while actually accomplishing another. This allows hackers to write code within a package that looks completely safe while actually embedding dangerous commands within the scripting.

The easiest way to prevent this issue is to only work with trusted packages and dependencies. Avoid working with unknown packages and developers. Combined with input sanitization, this will ensure malicious actors can't run code within your systems that you don't approve.

Cross-Site Scripting

Cross-Site Scripting, also known as XSS, is another form of injection vulnerability. An XSS attack occurs when a hacker injects code into a website that is not under their control. This code is then transmitted to the website's visitors where it performs malicious activities. It's known as "cross-site" scripting because the otherwise-benign website is being used as a bridge between the hacker and their victims.

There are two main ways XSS attacks occur. Either the hacker embeds the malicious code through a web request or similar untrusted source, or the script is hidden in dynamic content the website has not vetted.

For instance, advertisements were a significant source of XSS attacks for a long time. Websites that didn't carefully check the ads they showed users often presented dynamic ads that included XSS programs. Web app developers need to be cautious both with how they handle requests of any kind as well as the third-party code they allow to be displayed through their Python applications.

TLS Certification Disabled

Transport Layer Security (TLS) certificates, also known as SSL or digital certificates, are a fundamental safety feature in online applications. These certificates allow web apps to encrypt data before sending it to another location. With TLS certificates enabled, data sent over the internet, including passwords and other confidential information, is kept secure.

Python programs can be written with TLS certification turned off. While this saves a small amount of computing power, it also puts the program at significant risk of attacks. Without TLS certificates, the program can't verify whether other sites and servers are trustworthy. Furthermore, the data they send will not be encrypted, potentially permitting "man-in-the-middle" attacks if a hacker intercepts data between the source and its destination.

Hardcoded Secrets

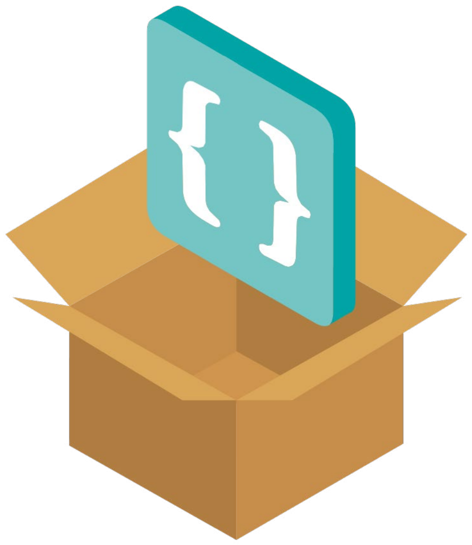
Once a hacker breaks into a system, their goal is usually to find sensitive information that they can use to further their goals. A common target of many hackers is "secret" data, including things like account names, passwords, or confidential paths and file names. This information can be used to accomplish further attacks or steal funds and accounts from their rightful owners.

That's why hardcoding secrets in Python is such a security weakness. "Hardcoding" is the act of leaving secret information unencrypted in the program's code. Anyone who looks at the script can simply scroll and find the sensitive data in plaintext.

The alternative to hardcoding is using encryption. With appropriate encryption, sensitive data is never stored in plaintext within the code. Hackers will instead discover a string of jumbled characters that require the encryption key to decode. This puts an extra layer of security between the most sensitive data your Python application contains and hackers who want to steal it.

How Python Packages *Impact* Project Security

Even the best developer can still fall victim to security flaws if they don't pay attention to the packages they use. Python's open-source nature provides a huge variety of packages for developers to use in their programs. However, many of these packages aren't maintained by their original creators.



This means that packages in both direct and indirect dependencies can be a major security hole. If a package's creator does not update it when a new vulnerability is identified, the flaw will remain. Any program that relies on that package will risk security breaches as long as the dependency remains.

Again, indirect dependencies can cause significant issues downstream. A basic package that hasn't been updated to patch a vulnerability could affect hundreds or thousands of programs. All it takes is a simple, outdated package being used as a direct dependency in a more recent and popular package to create cascading security flaws.

The 10 Most Common Vulnerable Python Packages and Their Security Flaws

The extent of your program's vulnerability depends on the packages you use. There are many packages that are maintained and kept up to date, making them perfectly secure additions to your code. However, some common packages have significant issues that you need to mitigate if you want to use them safely.

These packages include:

1. URL-lib3:

This HTTP client is the single most-downloaded package on PyPI because it provides some essential features for web requests, SSL/TSS verification, and more. The package is kept up to date, so you can trust that the most recent version is always safe. However, many projects and other packages use older versions of urllib3 that have high-risk injection vulnerabilities. If you choose to use urllib3, always make sure you're referencing the most recent version.

2. IP address:

This package is used to help programs manipulate IPv4 and IPv6 addresses effectively. However, the module also accepts leading zeros on IPv4 addresses. This permits hackers to bluff the IP address checks the package is designed to perform. As a result, any program with an IP address as a direct or indirect dependency is at risk of bluffed IP addresses and Denial of Service (DOS) attacks. No version of ip address is free from this issue.

3. Cryptography:

The cryptography package offers Python developers different primitives and cryptographic recipes they can use within their program. However, the fundamental structure of the package makes it vulnerable to memoryview attacks in which users can overwrite the contents. While this is not as great of a risk as other packages, cryptography's popularity may make it a target in future attacks. No version of the package is free from the issue.

4. Pillow:

Pillow is an imaging library that allows a project to use and implement a wide variety of image extensions. Like urllib3, the package is kept up to date, but older versions with critical security flaws are still in use. Versions prior to the 8.1.1 release are vulnerable to DOS attacks through improperly monitored image sizes.

5. PyYAML:

The PyYAML package allows a program to parse and emit YAML, making it easier for humans to understand data produced by Python. Unfortunately, the incredibly popular PyYAML package also permits hackers to force programs to execute malicious code and bypass access controls with ease. Versions after 6.0 are safe, but older versions are still in use. Developers need to make sure they keep their PyYAML version updated to keep their apps safe.

6. Jinja2:

The jinja2 package is a popular, lightweight standalone template engine Python developers can use to simplify development operations. Versions prior to 2.11.3 handled str.format and str.format_map improperly. These issues gave hackers a way to escape the sandbox and interact with backend and server-side systems. Updating your jinja2 packages to the latest version is essential to avoid these kinds of attacks.

7. Pygments:

This package is a syntax-highlighting package that makes it easier to spot when a Python program is being written correctly. However, pygments was found to use regular expressions within its programming. This permits DOS attacks when hackers overload the regular expression and force the package to stop working. Pygment versions 2.7.4 and later do not include this error.

8. Requests:

This HTTP client had several significant security errors in earlier versions. Packages prior to 2.20.0 made it easy for hackers to access credentials in several ways. Requests sent HTTP authorizations to HTTP URLs, permitting sniffers to spot credentials in the request. Similarly, hackers could read the Proxy-Authorization and Authorization headers to obtain sensitive information like netrc passwords. Updating to a more recent version of the requests package is essential for security.

9. RSA:

This package supports RSA implementation, offering extra encryption to Python developers. However, versions before 4.7 did not handle ciphertexts securely. This gives hackers a path to decrypting the information rsa is intended to protect. All programs that include rsa as a direct or indirect dependency should immediately upgrade to the most recent version to remove this vulnerability.

10. Django:

This package offers a high-level Python Web framework that developers can use to support their web apps and write programs more quickly. The issue with django is that versions prior to 3.1.13 are vulnerable to SQL attacks. The package doesn't handle GIS functions correctly. Hackers can use this to force programs using django to perform malicious actions. Updating django to the most recent version is enough to resolve this issue.

The *Risks* of Python Containers

One way developers choose to simplify the development operations process in Python is by using containers. A container is an object that holds other objects, such as a list, dictionary, or set. Containers organize these other objects and allow developers to reference them in other places within the program.

You can also create container images, which contain a prepackaged application and all its dependencies in one place. These images are the easiest way to get a Python application running on new devices. They can also make it easy to import a significant number of packages into your own program at once. That's why containers are a fundamental part of modern programming.

Still, containers and container images can also pose problems. Poorly constructed containers can be "leaky," giving hackers potential access to the data they contain. Without proper security measures, container images can make your program even more vulnerable to breaches. Common flaws in container security include:

- **Reliance on unsecured base or parent images:** Containers are frequently created using "base" images that are blank but include some basic formatting or "parent" images that come with many already-included dependencies. If you use a base or parent image that has security flaws, those issues will immediately affect your program as well.
- **Lack of container visibility:** Containers are designed to be dynamic and allow developers to adjust to changing needs over time. However, that flexibility can also make it easy to lose track of what's actually included within a container or image. If you don't know what's in all of your containers, you can't successfully manage their security.
- **Broad container communication:** Many containers are intended to communicate with other containers. This communication is critical to their function, but it needs to include careful security filters. Without appropriate filtering, any hacker that can access one container will also be able to enter any other container you use.

This doesn't mean you should avoid using containers. These tools are too useful to ignore. However, it does mean that you need to implement safety and security measures when you use containers, Otherwise, you risk serious breaches.

Best Practices for Mitigating Python Security Risks

Now that you understand the various ways Python programs can suffer from security vulnerabilities, you can focus on preventing them. Following a few best practices can help you keep your Python program secure and free from common flaws that could lead to major security breaches or stolen information.

Keep Python Up to Date

The most important thing you can do for your security is to keep your version of Python updated. When new vulnerabilities are detected, the Python community steps into action to revise the language and ensure that Python remains as secure as possible. However, the work done by community members to keep Python secure only affects your programs if you keep your versions up to date.

Regularly check the Python hub to make sure you're using the most recent stable release. This ensures that anything you write has the most protection possible against malicious actors. You should do the same for all of your dependencies, too, since they are one of the most vulnerable points in Python programs' security footprints.

Be Wary of New Packages

As an open-source community, anyone can contribute new Python packages for other people to use. That's what makes the language so versatile. However, it's also a potential vulnerability. The size of the community means that it's not hard for an inexperienced programmer or a hacker to upload a package that has unexpected vulnerabilities in it.

You can avoid these risky packages by working with trusted alternatives. Look for packages that have been used and reviewed by many other Python developers. These are more likely to work well. More importantly, any flaws are more likely to be discovered and fixed because of the size of the user-base. That makes them a safer choice for security-minded developers.

Pay Attention to New Vulnerabilities

It's essential to stay up to date on the state of Python security vulnerabilities. New fundamental vulnerabilities are being discovered multiple times a year. When penetration testing brings these issues to light, you need to take immediate action to prevent hackers from using the new information to attack your program.

The Python hub and community are excellent at spreading the news about new vulnerabilities. Check up on the state of the language regularly by visiting the Python hub site. Major new vulnerabilities will usually be announced somewhere on the front page, so everyone knows they need to take action.

Be Smart About Secrets

Whether you're using Python or any other programming language, you should never hardcode secrets like passwords. In fact, it's best practice to keep passwords and other sensitive information out of your source code entirely. Store this information in a different file if you want to manage it yourself. The separation makes it significantly harder for malicious actors to find the data in the first place.

Furthermore, secrets should be encrypted whenever possible. Properly using a package like cryptography can help you keep passwords from falling into the wrong hands. Even if a hacker does manage to find the data, they won't be able to do anything with it without the encryption key, keeping your program and users safe.

Use a Virtual Environment

Virtual environments make it easier for you to keep every program you write in Python independent. A virtual environment acts as an isolation chamber for everything involved within a Python program. The scripts, packages, and libraries used by the program are all interpreted entirely within the virtual environment, keeping them separate from any other programs you're developing.

As a result, it's easier to keep individual projects secure. Virtual environments remove global Python versions and dependencies from the equation. That means that a package you use in one program won't automatically be used in another. As a result, any dependencies with security flaws can only affect the programs they connect with instead of all of them.

Format Strings Safely

Any data that enters your program from an external source is a potential security risk. This is the most vulnerable access point for XSS and SQL attacks. When you're accepting external data, it's essential to sanitize the string before doing anything else with it.

There are several Python libraries you can use that will handle data sanitization for you. For example, **bleach** is an HTML-sanitizing library that works off of a white-list, automatically stripping markups and attributes from user input. You can also work with frameworks like **django**, which come with native sanitization tools.

Turn Off Debug Before Going Live

Debug is a useful feature when you're developing a program. However, the in-depth error messages debug provides give hackers valuable information about how they could possibly harm your app. Always turn off debug before sending an app to production. It's a small change that puts one more hurdle between hackers and the systems you want to protect.

Protect Your Python Programs With *Kiuwan*

If you're dedicated to protecting your Python programs, Kiuwan is there to help. Kiuwan provides an all-encompassing security solution for Python applications of all kinds. You can use Kiuwan's platform to perform security tasks like:

- **QA code analysis**
- **Governance and lifecycle monitoring**
- **Application vulnerability detection**
- **Local security code scanning**

With Kiuwan, you don't have to guess about things like known vulnerabilities or flaws in your direct and indirect dependencies. Kiuwan will monitor your programs, identify risks, suggest mitigation solutions, and help you develop more secure programs with less stress.

It's time to stop worrying about security flaws and start working with a partner that will support your DevSecOps needs from development through production. If you're ready to learn more about how Kiuwan can support secure Python development within your organization, you can schedule your demo today.

YOU KNOW CODE, WE KNOW SECUIRTY!

GET IN TOUCH:

Headquarters
2950 N Loop Freeway W, Ste 700
Houston, TX 77092, USA

United States **+1 732 895 9870**
Asia-Pacific, Europe, Middle East and Africa **+44 1628 684407**
contact@kiuwan.com
Partnerships: **partners@kiuwan.com**