Group 1 | Ryan Brower | Bryce Sanders | Lance Riley | Everett Tsosie | Harrison Parrish | Jared Harper

CSE 453 Lab Report 07

# Introduction

As software developers, we understand the vital role that technology holds in our lives today. Most of our interactions and assets, from family photos to bank accounts, are layered behind complex software systems. Security should be at the forefront of software planning and development, to maintain confidentiality, integrity, and availability for end users. In this report we go over the most common attacks that threat actors use, and the best security practices and principles to help development.

# Security Practices & Principles

Security is not a one-time process and should always be considered throughout the development process. New features may introduce new vulnerabilities and it is up to the team to ensure the product remains trustworthy to customers. A proper threat model and methodology are key to preventing insecure design. The threat model takes account of potential security risks for a specific organization or application. Agile methodology is recommended because it contains iterative practices that re-check and perform regression tests on various parts of an application which allows for security reevaluation.

The best thing to do when beginning development is to design the software with security in mind. No amount of IT security will be able to protect sensitive data if the software is not designed securely. Secure design includes proper input validation and keeping all third-party software up to date with the latest security patches. Reducing the amount of third-party code in the software stack reduces the attack surface of the program, giving attackers less places to target. Data should always be transferred securely with up-to-date encryption methods.

Online resources, like the Open Worldwide Application Security Project (OWASP), keep up to date lists of the biggest threats to software security. Becoming familiar with these lists and up to date on how they change will help avoid the most prolific threats.

Security through obscurity means to give out as little information to users as possible. No user should be completely trusted, and information should be given on a need-to-know basis. The less attackers know about the inner workings of the system, the better. When giving error messages back to users, ensure the messages are vague.

Security depth refers to layering defenses on top of another. This reduces the amount of damage if one defensive layer fails. If one layer of defense protects against 90% of attacks, and another layer defends against 90%, that would mean about 99% of attacks would fail.

# Common Attacks

Keeping up to date with security threats and attacks is the first step to mitigating attack scenarios. OWASP keeps an up-to-date list of contemporary threats, the greatest of which is broken access control. This equates to privileged access to data or functionality that should otherwise be restricted. Many other attacks seen today serve as means to this end, with the end goal of access to unauthorized resources.

Another common attack is an injection attack, which takes advantage of dynamically generated SQL statements to run commands that aren’t intended by the host. Other input attacks include overflow and memory attacks, which run commands on the host computer and can crash the system, halting functionality for regular users and carrying out a denial-of-service attack. File upload abuse is yet another way to harm a target system, and another reminder of the importance of input validation.

Many software attacks are performed in a web environment, because of the browser’s access to both the target machine and the outside internet. These include cross-site scripting, a means of injecting executable HTML code into a web input, and session attacks, which focus on getting session tokens or credentials with the goal of impersonating a legitimate user, whether by snooping on the target or luring them onto a fake login form. Attackers also can spoof an authentication token or adjust a URL, to access pages or API endpoints that would otherwise be restricted.

Appendix

# Programming Foundations: Secure Coding

In the ever-evolving landscape of software development, one constant remains - the need for secure coding practices. Security is a multifaceted concern, with risks lurking at every stage of the Software Development Life Cycle (SDLC).

**Security and Risk Overview:**

The foundation of secure coding lies in understanding the multifarious risks that can affect software applications. In a dynamic threat landscape, consistency is indeed a rarity. As such, developers must embrace perpetual learning, as threats constantly evolve. The process of engaging in code reviews from a security perspective is not just a simple best practice; it's a crucial element of secure coding. Development managers, on the other hand, must comprehensively grasp the risks their teams are addressing. It's essential to recognize that cyberattacks can stem from various motivations, not limited to financial gain. Understanding these motives is crucial as it informs the security measures taken. Attack trees offer a strategic approach, helping identify potential exploits. Utilizing tools to test code for vulnerabilities is essential, as it provides valuable insights. By fostering constructive conversations between developers and security professionals, the entire team benefits. Recognizing that exhaustive testing is impractical, automation of testing in high-risk areas during each release proves highly effective.

**Web Client Server Interaction Code Issues:**

Web applications, due to their very nature, are especially susceptible to security breaches. Anything interpreted can fall prey to injection attacks. The infamous Cross-Site Scripting (XSS) attacks persist on the OWASP Top Ten list. This form of attack involves an attacker injecting a script via a user input field, putting sensitive data at risk. Communication channel attacks pose another significant threat, especially when free, unencrypted Wi-Fi networks are in use. Understanding these vulnerabilities is vital for secure coding practices.

**Thick App and Client-Server Interaction Issues:**

Issues in thick applications and client-server interactions underscore the importance of effective error handling and logging. Often, error messages inadvertently expose sensitive information. Developers must adopt a defensive stance by limiting the data disclosed in these messages, especially during production. Even non-production systems can be potential attack vectors, emphasizing the importance of secure configurations. Techniques such as two-way certificate pinning, rigorous system patch management, and controlled file system access also contribute to a secure codebase.

**Crypto and Security Misuse Issues:**

Cryptography presents a significant challenge for many developers, and the risks associated with broken or misused cryptographic routines are pervasive. Secure coding practices demand the use of up-to-date algorithms and diligent parameter usage. Implementing cryptographic algorithms correctly is essential, as even minor errors can jeopardize security. A cardinal rule is to avoid developing homemade cryptographic solutions. Instead, developers should seek expert advice and adhere to industry standards. Proper key management and secure storage of cryptographic materials are also crucial components of this aspect of secure coding.

**Security in the SDLC:**

Building a culture of security starts with an emphasis on secure coding in the design phase. Developers must view their applications critically, considering potential vulnerabilities at every step of the Software Development Life Cycle. Security education, humility, and openness to learning from mistakes are essential elements of fostering a culture of security within a development team. A designated security champion within the team can inspire change, ensuring a collective commitment to security practices. Furthermore, threat modeling and the incorporation of security checks into every phase of the SDLC can help identify and mitigate security risks early in the development process.

**Conclusion:**

In the complex realm of software development, secure coding practices are essential to safeguard against an ever-present array of risks. Understanding the intricacies of these risks and vulnerabilities, leveraging automated testing, adopting security-conscious development practices, and fostering a culture of security are critical steps toward building robust and resilient software applications. In an age where data breaches and cyberattacks are becoming increasingly common, secure coding is not merely an option; it's a necessity for the modern software developer. This commitment to security not only protects valuable assets and sensitive data but also safeguards an organization's reputation and its users' trust.

# Programming Foundations: Web Security

**Overview:**

**Security is not a one-time process** and should always be considered throughout the development process. From the early stages to launch and beyond. You must frequently reevaluate security. New features may introduce new vulnerabilities. It’s the development team’s responsibility to ensure a trustworthy product to the customer. Not securing a website risks costs to customers such as data or monetary theft. A defaced web page can hurt a brand.

There are several types of hackers, all with different levels of expertise and agendas. Some hackers seek money or recognition. Other hackers have certain political agendas. Consideration of what kind of hackers we are defending ourselves against helps us build a threat model, a prioritized list of the risks and potential that an organization faces. A popular political website might defend against skilled hacktivists while a blogger would defend against simple automated hacking scripts. Although, it is impossible to achieve total security, so it’s important to use a thread model to prioritize what to defend against. **Overall security is only as strong as the weakest link.**

**Security Principles:**

Every program and every user must have the least amount of privilege needed to complete the job. You may limit what code is available for other code to use. Simple code is more secure. Using functions in a library would be far more secure than writing your own function to do the same thing. You may disable unused library functions. Anything you can disable would eliminate the need to keep it secure. Use deny lists or allow lists to regulate privileges of users accessing a system. Obscurity of information synergizes with the principle of least privilege, where you aim to give out as little information as possible.

Within a pool of users, there will be those who will attempt to give unexpected inputs to find exploits. Users should not be fully trusted. Test edge cases using lots of input, no input, or unexpected characters such as emojis or other language characters.

Create a map of data passageways. Keeping an inventory of where data enters your systems and how it moves helps you identify where the exposure points are. You may decide to create several layers of security checks to ensure data is valid and safe wherever it is being moved and processed. This creates depth to your security.

**Filter Input, Control Output:**

Going along with security depth, you can regulate requests from the internet. Only running requests whenever you are expecting them and where you are expecting them from. Input data from users should not be trusted as is but should be sanitized and validated first before being read. Label your own code by naming sanitized items as ‘clean’ or ‘safe’ inside your code to keep track of implemented safety measures. You may also decide to log errors, this would be helpful in identifying suspicious behavior such as failed login attempts.

Keep code and credentials private. Avoid giving users the inner workings of your web application, the less they know the better. There is a public directory that serves as the access point for the website, and a separate private directory that isn’t accessible by the user which processes the PHP and outputs HTML to the user, giving no hint to what the PHP looks like. Credentials should never be hardcoded. Use variables and store credentials in a separate file, encrypt it to add more depth to security. Vague error messages also avoid giving users useful information. Don’t return a message stating “Incorrect password” or a hacker will realize they’ve acquired a valid username.

**The Most Common Attacks:**

Many attackers target credentials using methods such as brute-force, dictionary attacks (brute-force but prioritizes words in a dictionary), and credential stuffing (saving and prioritizing previously discovered passwords.) It’s important to use strong passwords. Longer passwords are more effective than short complex ones.

URL manipulation is when a curious user plays with URL endpoints, which may lead to unchecked access to a restricted resource. The best ways to prevent Insecure Direct Object References is by using access control to verify the privileges for access and/or change out Direct Object Reference for Indirect Object Reference. Indirect Object Reference would create a separate object the user can interface with while hiding the object used in the database.

SQL Injection is an attack method where SQL requests are built and inserted into the SQL query string, allowing attackers to run arbitrary commands. They may change data, probe the database, or even elevate their access to the database. Sanitizing data and denying access privileges to database can be a solution.

Cross-site scripting (XSS) is an attack where the attacker injects code onto a webpage that other browsers will execute. There are a few methods to do so, such as reflected XSS that runs when a page is loaded, Stored XSS which runs when remote data is received from storage, and DOM-based XSS which runs when a JavaScript event is triggered on the page itself. Another variation of XSS is Cross-site request forgery (CSRF) when an attacker tricks a user’s browser to make requests to another site. This can abuse cookies saved on a computer to perform actions only the victim can perform. Use CSRF tokens to ensure submitted requests are valid.

Cookies are easily visible and easily stolen. Cookies store session data for users, but an attacker can listen in on cookie data when it is in transit. The attacker can also use XSS to retrieve cookie data risking forging requests or modifying the cookie for other purposes. Sensitive data should be stored on the server side guarded by the session, so the data cannot be observed in transit or in storage. Although, a session ID cookie is needed to access the session data, which itself needs to be secured. Encryption is a good way to make traffic and storage unreadable. Another security method is to run a checksum algorithm which is reanalyzed at the receiving again and checked to see if the same checksum is calculated, indicating the cookie was not modified.

Session hijacking is achieved when a session ID is stolen. This allows the hacker to access all session data. It is done by eavesdropping on open Wi-Fi networks.

Session fixation is like hijacking, but instead of stealing a session, the attacker provides a session to the victim and waits for the victim to login and authenticate the session.

Remote Code execution (RCE) is when an attacker executes internal operating system commands on a server. This gives them the power to read, add, modify, or delete files. It is the hardest hack to pull off. The vulnerability comes from whether the developer uses powerful system functions within code. Some programming languages let you disable those powerful functions altogether. Data sanitization, validation, and allow lists can help.

File upload abuse is when attackers abuse public file features of a website. Malware could be uploaded pretending to be a different file type. SPAM, spyware, ransomware, and other malware types can be uploaded. There are several validation techniques to prevent this abuse. This includes checking files size, type, extension, allowing only certain file types, or even checking the first few lines of the file which often includes file information. You can also scan files with antivirus software.

Denial of Service is when legitimate users can no longer access a service or resource, eliminating availability. Distributed Denial of Service is often a large amount of zombie computers to overwhelm a network. First, understand if this attack is in your threat model. There are companies that offer DDOS mitigation services. Throttling or rate limiting slows down malicious activity. Filtering applies rules to incoming traffic, certain requests are allowed, and others are rejected. “Sink-holing” redirects traffic to a new destination that may capture and analyze the traffic. Blackholing is redirecting traffic to nowhere.

**Other points:**

* Update your software. When security patches are released, hackers can be quick to exploit the vulnerabilities being patched.
* Back up your data. Ransomware, hosting servers going down, hard drive failure are risks to data availability.
* Secure your domain. Domains can be hijacked, use multifactor authentication for your domain registrar and wherever your DNS configuration is hosted.
* Secure your server.
* Join the security community.

# Developing Secure Software

Understanding Software Security:

Software security is an attempt to minimize the creation of flaws and bugs as well as preventing artificial security threats. part Software security professionals is explaining the threat of software security due to its importance in Guarding essential systems in our day to day lives. An important part of software security is knowing the terminology. We can’t possibly know every threat so a practice of classifying the threat’s level of important so we can manage the risks of such attacks.

Software Security Threats:

While there is an obvious threat of hackers to software security there are other factors to consider. Computer hardware can be sabotaged or fall victim to natural disasters such as earthquakes or tsunamis. Countermeasures to these would be to avoid a single point of failure, an uninterruptible power supply, and physical security preventing theft and damage to the hardware. Another threat to software security would be at the code level. Bad code either put in either deliberately or by accident like logic bombs or buffer overflow attacks can make the software explicitly vulnerable. To mitigate this security oversight, peer code review, and automated code analysis are tools that must be used to identify vulnerabilities in the code. Design patterns deal with things like SQL vulnerabilities and input validation.

Secure Software Design:

To build secure software, it is usually best to have a plan to follow and go over it before the coding stage. This lessens the possibility of making a large mistake and having to start all over again. There are several techniques to help with the design of the software. These techniques involve UML or unified modeling Language. Recurring problems in security have design solutions called security patterns with two types. Design patterns for countermeasures at the design level. And the second type is patterns for software architectures.

Secure Coding:

This section mostly goes over examples of attacks and possible countermeasures. These include buffer overflow attacks, Broken authorization and management, insecure direct object references & and Sensitive data exposure.

Testing For Security:

This Section Goes Over testing techniques. Black Box testing is where the testers don’t have access to the source code. White box testing is where they have full access to the source code. Grey box testing is a hybrid of the two.

* **Static analysis:** the analysis of the source code.
* **Dynamic Analysis:** Tests the software while it is running.
* **Penetration testing:** also called ethical hacking, this involves testing the system by simulating attacks done by malicious actors.
* **vulnerability management:** systematic tracking of vulnerabilities to minimize their impact.

Recent Developments and Future Decisions:

* **DevOps and software Security:** DevOps is the combination of development and operations which is usually an automated process to test the software for vulnerabilities.
* **Cloud Security:** Because you do not own the physical servers, any service deployed to a cloud service needs extra encryption and monitoring.
* **Developer-friendly software security:** there should be security tools made for software engineers to easily understand.
* **IoT and software Security:** the internet of things will provide new security challenges.
* **Rules and regulations:** new government regulations are affecting development.

# CompTIA Security+ (SY0-601) Cert Prep: 2 Secure Code Design and Implementation

One of the best sources when looking at application security is OWASP's top 10 list. It is a list that has the top 10 most common security risks in applications and software.

The top risk is Broken Access Control which is when developers fail to check on the back-end whether a user is authorized to access a particular function of an application, this also has insecure direct object references which occur when a developed exposes some details of how an underlying application function works and doesn’t perform proper security checks for the application.

Crypto Graphic Failure happens when an insecure web application accidentally exposes sensitive information to eavesdroppers due to poorly configured cryptography. This may be as simple as accidentally placing a customer file on a publicly accessible portion or website. Encryption or iMac or when web server administrations fail to implement the HTTPS protocol to encrypt information that has been sent over. The Internet.

Injection flaws occur when an attacker is able to insert code into a request that's sent to a website and then trick that website into passing the code along to a back-end server where it's executed. The most common example of this is the SQL injection attack against databases.

This all leads back to an insecure design; all traces of security issues are traced back to the initial creation of the code. These are issues that arise when an organization does not properly identify and satisfy security or requirements when it conducts Insufficient Threat Modeling, and when it fails to follow security best practices.

Some of the main coding practices that you should implement into your software development plan Rs listed.

* **Input validation** - This technique filters user input, making sure that the input provided by the user does not contain malicious or otherwise unexpected values. Examples (Whitelisting, and Blacklisting)
* **Parameterized queries** - This protects applications against injection attacks. In a parameterized query, the client does not directly send SQL code to the database server. The client sends arguments to the server, which then inserts those arguments into a precompiled query template.
* **Authentication and session management issues** -
  + Never store passwords in plain text. (Hashed, salted)
  + Developers and system architects must ensure that passwords are encrypted in transit so that an eavesdropper viewing network traffic doesn't see your user password being sent from client to server.
  + Require Transport Layer Security or TLS. TLS encrypts the entire web session using the HTTPS protocol and prevents an eavesdropper from seeing any of the session contents, including the user password.
* **Output encoding** - Output Encoding replaces dangerous characters with safer ones that still visually look the same, reducing the chance of an attack while still preserving all functionality.
* **Error and exception handling** - Error and Exception Handling allow the software system to continue functioning even when encountering issues or edge cases, still providing regular use to all other users.
* **Code signing** - Code Signing is a way for developers to ensure that code or software came from a legitimate source. This uses digital signatures to verify that code is legitimate and hasn’t been tampered with.
* **Database security** - Database Security can be achieved with the use of Normal Forms and Normalization. This prevents inconsistencies and anomalies, ensuring that all data is of expected types as it is put in.
* **Data deidentification** - Data Deidentification is the practice of going through datasets and removing identifying data from sensitive datasets. This way, even if sensitive data is hacked and stolen, it can’t be traced to the owners.
* **Data obfuscation** - Data Obfuscation is a way to protect data by transforming it into a type or format that is unreadable. Salting and Hashing allows data to be protected and secure, even if the values are stolen.

# CompTIA Security+ Notes (2)

1 – Software Lifecycles

* Software Security relies on a secure application platform.
* Security should be involved in every software development stage.
* Beginning with design, when you translate business or consumer needs into design, incorporate and plan for secure systems.
* **Waterfall Model** – Rigid linear design process. If one step fails, you go to the previous, but only one step at a time.
* **Spiral Model** – A Circular process that repeats the same steps but at an increasingly larger scale.
* **Agile Model** – A repeated model that focuses on incremental progress and allows developers to refine the changes that are made.
* Maturity Models give businesses a way to evaluate the state of their software product.
* **CMMI** – 5 tier software maturity model; Initial, Managed, Defined, Quantitatively Managed, and Optimized.
* Choose the maturity model that best suits your use case.
* Software Development is never finished for a product. Plan for long-term maintenance and support.
* Using distinct and encapsulated code environments allows us to test and prepare the code for production.
* Developers and IT teams have different goals and different reactions to changes in the code. DevOps was created to bridge the two and relates very closely to agile methodology.

2 – Software QA

* Manual Code Reviews are a great way to weed out bugs and inconsistencies in software.
* Thorough Test Suites can keep functionality to exact specifications and maintain security within the software.
* Static tests are examining and cross-comparing static code to the design or requirements.
* Dynamic tests are executing the code or parts of it to verify end functionality according to expectation.
* Fuzz testing – testing a module by entering many different key-value pairs as test parameters. This has the purpose of uncovering bugs or issues that are unexpected and unplanned for.
* Code Repositories are where software products are stored, in such a way as they can be shared and collaborated on by many developers at a time. Repositories keep everyone on the same page and reduce the amount of dead code. Git and version control allow developers to publish and explain changes so that everyone is aware of every change in the moment.
* Application Management. Applications can be whitelisted or blacklisted to keep systems safe and prevent unwanted software execution. You can set custom policies for application control, which keep your development environments safe.
* Third-Party Code comes in the form of libraries or modules that can be imported to your software project. If they are trusted and kept up to date, they can improve the security of your code.

3 – Application Attacks

* OWASP keeps an up-to-date registry of all contemporary software attacks.
* Software attacks can be consequential. Everything has software elements, from our phones to airplanes, which can be very consequential if hacked.
* Application Hardening keeps software from attacks – Type and scope of encryption, users with access to the application, access granted to authorized users.
* SQL Injection threatens software applications. Attackers can modify data to include SQL commands and can manipulate or remove sensitive data.
* Cross-Site Scripting (XSS) are scripts that run on your browser, sometimes without your knowing. This can be a way to attack your systems or perform an attack from your system. Request Forgery is when a script from one site accesses data from your browser or other pages open on it.
* Directory Traversal is an attack that happens when a script accesses a system’s file structure. The location can be manipulated to files or folders that should be restricted to an end user.
* Overflow attacks target a machine’s memory, offering inputs larger than the space expected for them. This can cause a program to crash, or ruin stored data.
* Cookies are information and analytics stored by the server about your browser and site use. Knowing what cookies are being stored is the path to being safe when handling them.
* Session Hijacking is when an attacker takes a session token and uses it to impersonate a legitimate user.
* Code Execution Attacks allow an attacker to run commands on a target system through the web.
* Privilege Escalation is an end to many software attacks, allowing an attacker administrative access to a target machine.
* Driver Manipulation is when a user installs a malicious driver, which has low-level access to the target operating system.
* Memory Vulnerabilities allow an attacker to execute their own code through memory manipulation. It can lead to the application consuming all of a machine’s memory.
* Race Conditions are when two parties access the same data, and if unprepared, a system can manipulate or ruin data in memory.

4 – Secure Coding Practices

* Input Validation – Sanitizing input strings of command language. This is the first line of defense when dealing with any malicious input. Consists of either whitelisting or blacklisting practices.
* Parameterized queries allow customers access to regular use but doesn’t use custom inputs, keeping a layer of security in the system.
* Authentication and Session Management – Hashing and encryption allows users to connect securely and makes it difficult or impossible to impersonate them.
* Output Encoding replaces dangerous characters with safer ones that still visually look the same, reducing the chance of an attack while still preserving all functionality.
* Error and Exception Handling allows the software system to continue functioning even when encountering issues or edge cases, still providing regular use to all other users.
* Code Signing is a way for developers to ensure that code or software came from a legitimate source. This uses digital signatures to verify that code is legitimate and hasn’t been tampered with.
* Database Security can be achieved with the use of Normal Forms and Normalization. This prevents inconsistencies and anomalies, ensuring that all data is of expected types as it is put in.
* Data Deidentification is the practice of going through datasets and removing identifying data from sensitive datasets. This way, even if sensitive data is hacked and stolen, it can’t be traced to the owners.
* Data Obfuscation is a way to protect data by transforming it into a type or format that is unreadable. Salting and Hashing allows data to be protected and secure, even if the values are stolen.

# Secure Coding in Python

**Introduction**

While not the most common language used in web-based application development, Python is still quite popular and is utilized by platforms such as YouTube, Instagram, Spotify, and more. Like any other programming language, *python needs to be designed, implemented, and maintained with security in mind*. No amount of IT security can make up for the developers not doing this. This is best accomplished by following Python best practices which are driven by real-world threats (see [OWASP Top 10](https://owasp.org/www-project-top-ten/)).

**Best Practices**

A good rule to follow when including open-source technologies (libraries, packages, frameworks, etc.) in projects is to ask the question “Would I pay $1,000 of my own money for this technology?” This helps to declutter the software stack which reduces the “attack surface” of the web application, decreasing its vulnerability. When using open-source technologies, it is crucial to stay up to date with the latest security patches. Never use components with known vulnerabilities. Tools like Pipenv allow you to install packages and easily manage them and perform security checks. When using APIs, it is important to remember that data is often more exposed when being exchanged and sensitive data should be secured.

**Python Pitfalls**

Python is an easy to learn, multipurpose programming language. It has lots of built-in functions that make it easy to write code quickly. It must be used with caution though, as there are many easy ways to write insecure code. The following is a list of the most common pitfalls that should be avoided when using Python:

* Python uses dynamic typing. This results in what are known as “Truthy” and “Falsey” values, which are non-Boolean values that can be coerced into “True” and “False” by the Python interpreter if checked implicitly. An example of this is “if [1, 2, 3]:”. A list that contains values is “Truthy” and will be interpreted by Python as “True”. Explicitly checking for Boolean values will avoid “Truthy” and “Falsey” values being accepted where they should not be.
* Python’s built-in assertion engine is great for debugging and testing but should never be used in business logic. This is because of Python’s ability to run in “Optimized Mode” which removes assertions from the code. If “assert” is used in place of a basic “if” statement, this logic will be removed.
* The “pickle” Module is a Python module that implements a data format like JSON but should be avoided because it is highly susceptible to remote code execution.

**Conclusion**

When secured properly, Python can be a very powerful tool for developing web application systems quickly. Using web development frameworks such as Django can be very useful, as many basic web security measures are already implemented. So long as best practices are followed, and pitfalls are avoided, there is no reason that smoothly operating and secure web applications cannot be developed using Python.