Cybersecurity Final Project

Securing Pickle Files with HMACs

Ethan Dowalter

May 4, 2023

Abstract

The "pickle" package in python enables users to export almost any type of object containing data to an external file. This file can then be simply loaded into memory whenever the need for the object arises, as opposed to regenerating the object each time. The main application of this package is for objects which are computationally expensive to generate, such as machine learning models. This document describes a python package which streamlines the process of ensuring the integrity of a pickle file by attaching a hash-based message authentication code (HMAC) with it.

1 Purpose

By default there is no way to verify the integrity of a pickle file when loading it, so I am aiming to construct a new python package to give users the ability to safely store and share pickle files without having to worry about their integrity being compromised either at rest or in transit.

2 Scope

Due to this project's nature of only solving one very specific problem, its scope is small. The package has just two functions which essentially replace the "pickle.dump()" and "pickle.load()" functions part of the regular pickle package. Their replacements are very similar, but have the added functionality of writing/reading the pickle file with an HMAC. This HMAC is derived from a secure hashing algorithm (SHA3 most likely) which takes the bytes of the pickled object as well as a secret key as input. This hash is then appended to the end of the pickle file so its integrity can be verified upon reading/loading it. Generating and managing the secret key will be the responsibility of the user and falls outside the scope of this project.

3 Individual Components

This package replaces the "pickle.dump(obj, file)" function, which takes the object and the output file as required parameters, with a function that looks like "securedump(obj, file, SECRETKEY)" which takes the SECRETKEY as an additional required parameter to generate the secure hash to then be appended to the pickle file. Likewise, the "pickle.load(file)" function will be replaced by a function that looks like "secureload(file, SECRETKEY)" which will detach the secure hash attached to the end of the file from the securedump function and verify that it is the same hash which is generated with the current file contents and the passed SECRETKEY. If the hashes do not match, then the program throws an error which lets the user know that the file's integrity could not be verified, otherwise the object will simply be loaded into memory as with the normal pickle package.

4 Component Interactions

The secureload function takes into account that the pickle file is no longer a normal pickle file, but a pickle file with a hash appended to the end of it. Luckily, since the hashing algorithm used will always output a hash of fixed length, that length being 64 bytes, it is straightforward to peel off that fixed

amount from the end of the file. Then, that hash can be saved so it can be compared to the new hash generated from the rest of the file contents and the SECRETKEY parameter.

5 Non-Goals

This package only aims to ensure the integrity of a pickle file, not necessarily its confidentiality or availability. If an attacker had access to the file they could unpickle it without the need to verify the HMAC due to how the normal "pickle.loads(file)" function works, as it ignores the bytes past the pickled representation of the object. If a user wants to protect the file from attackers instead of simply protecting themselves from potentially malicious code inside pickle files, they can use any number of encryption libraries to help them do that. Also, it is not currently in the scope of this project to provide the user with a secure method of sharing or storing their secret key used for the HMAC.

6 Potential Threats

6.1 File Tampering

The most significant threat present to this application is the possibility of the pickle files being tampered with once they are created. If a file is in any way tampered with, then the HMAC integrity check will fail upon trying to load the pickle file. This effectively denies the user access to the file. This is by design because the contents of the file may be malicious. However, this means that an attacker could perform an attack that is kind of like a cross between a man-in-the-middle attack and a denial-of-service attack. More specifically, an attacker could intercept the pickle file with the HMAC and modify it in any way, then send the modified file to the actual recipient, essentially corrupting the file and preventing the user from being able to use it.

I would argue that this is a preferable alternative to potentially executing malicious code upon loading a pickle file with no HMAC integrity check though. Nonetheless, this is a trade-off that must be considered.

6.2 File Confidentiality

An attacker could potentially use the normal Python pickle package to load the pickle file themselves, even though the HMAC is appended to the end. This is because the way the normal "pickle.loads(file)" function works, as it ignores the bytes past the pickled representation of the object. That is to say, the confidentiality of the objects which users are sending to pickle files is not protected with this package.

6.3 Brute Force Attack

If the HMAC key is weak then an attacker could write a brute-force algorithm to guess the key, given that they also know which hashing algorithm is used in the background.

6.4 Hashing Collisions

However unlikely, it is still theoretically possible that an attacker could find a hash collision which could compromise the HMAC verification process. If I were to use a weak hashing algorithm, this would become more likely. Also, as quantum computing advances so to does the ability to break less sophisticated hashing algorithms.

7 Threat Mitigation

7.1 File Tampering and File Confidentiality

Being able to mitigate the tampering of pickle files falls outside the scope of this package, because this package does not aim to supply a means of secure file storage or secure file transfer. Nor does this package seek to remedy any issues of protecting the confidentiality of pickle files. Those responsibilities

fall onto the user should they choose to need that level of security, as there are other existing ways to solve those issues.

7.2 Brute Force Attack

I employ a check which makes sure that users supply a minimum length key to the "securedump(obj, file, SECRETKEY)" function when creating the pickle file. If the user-supplied key is too short, then I will display an error message which will inform the user that they need a key that has a length of at least 64 bytes.

7.3 Hashing Collisions

I will make sure to use a modern, uncompromised hashing algorithm to generate the HMACs. Namely the SHA3 hashing algorithm, which is the newest and most collision resistant hashing algorithm which NIST recommends according to this site.

8 5010 Addendum

The "securedump(obj, file, SECRETKEY)" function is able to take any python object as its 'obj' parameter, the 'file' parameter must be a file object that is open with the mode 'wb' (write/byte), and the 'SECRETKEY' parameter must be a string or bytearray that is at least 64 bytes long. This function verifies the input and then writes the bytedata of obj along with the HMAC generated by that bytedata and SECRETKEY to 'file'. So as long as the input parameters are properly formed, the securedump function will output a pickle file with an attached HMAC at the end.

The "secureload(file, SECRETKEY)" function takes another file object as the 'file' parameter, but this time it must be in the mode 'rb' (read/byte), and the 'SECRETKEY' parameter which does not have the same size-check as before because it would be redundant. It then uses the bytedata from the file and SECRETKEY to try and reconstruct the HMAC which is also in the file. If the reconstructed HMAC matches, then the file has passed the integrity check the the function returns the bytedata in the form of a python object to the user.

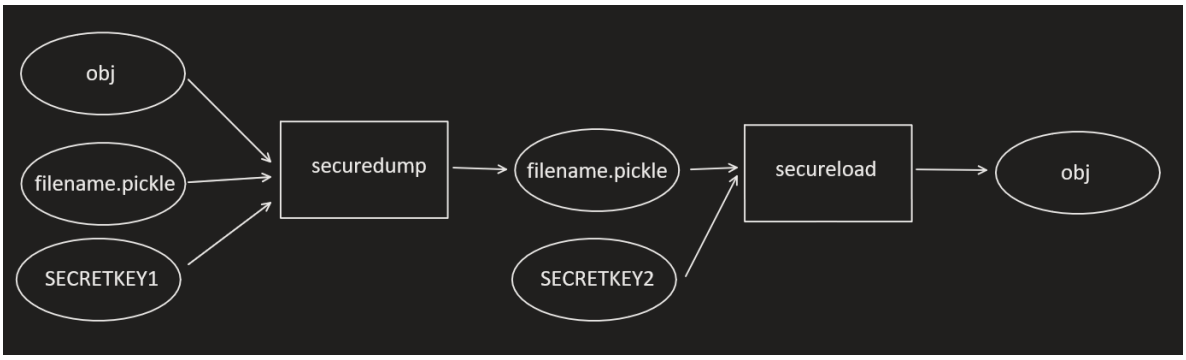

Figure 1: My crude sketch of the rough flow of how this package is designed. Note that SECRETKEY1 must equal SECRETKEY2 AND filename.pickle must not have been edited in any way in order for the object to be returned from secureload.

8.1 External Threats

An external threat this package faces is an attacker who attempts to tamper with the pickle file to modify or inject malicious code. This is obviously mitigated because the software generates an HMAC with the object byte data and SECRETKEY using sha3 and appends it to the file. Upon loading the file, the software checks the HMAC for verification, and if it does not match, it raises an error. This ensures that the integrity of the pickle file is maintained, and any attempt to modify or inject malicious code into the file will be detected.

Another threat faced is an attacker who attempts to intercept the SECRETKEY to gain unauthorized access to the file. This is mitigated because the software checks the length of the SECRETKEY to ensure it is at least 64 bytes long, which makes it difficult to guess or brute-force. Additionally, the SECRETKEY is not stored in the file, and therefore, it cannot be intercepted in that way.

8.2 System Threats

One system threat lies with the file objects, such as incorrect modes or file sizes. This issue is mitigated because the software checks the mode of the file object to ensure it is in the correct mode ('wb' for securedump and 'rb' for secureload). If the file object is not in the correct mode, it raises an error. Additionally, the software checks the size of the file to ensure it contains at least 64 bytes for the HMAC. If the file size is insufficient, the software raises an error.

Another issue involves running into errors during the pickling or unpickling process. This is mitigated because the software uses Python's built-in pickle module to serialize and deserialize the object, which ensures that the object can be pickled and unpickled correctly. Additionally, the software does not modify the pickled data, which ensures that the data remains intact during the process.