Malicious Pickles: Keep My Pickles Edible

Problem:

We intend to explore vulnerabilities in Pickle, Python's serialization module. Pickle is used extensively in machine learning frameworks such as PyTorch and TensorFlow when saving and reconstructing model weights. These weights can later be uploaded to public machine learning forums and model sharing hubs, making pickle files a soft target for opening backdoors into the high-performance computing resources of an unsuspecting research team. And, unfortunately for researchers, Pickle's architecture and "unpickling" process opens the door for arbitrary code execution.

Pickle uses a virtual machine, unlike other serializing/deserializing tools that use a series of marshaling hints, in order to "de-pickle" and reconstruct Python objects while keeping these objects' structure ambiguous and shareable. Marco Slaviero finds that the presence of Pickle's virtual machine and the special opcodes REDUCE and GLOBAL leaves pickle files open for abuse, allowing for arbitrary code execution through a variety of opcode injection attacks [1]. Hugging Face has recently posted warnings to unsuspecting researchers and hobbyists of Pickle's insecurity, but has abstained from direct action against model binaries containing malicious weight pickles. We aim to produce a **pickle sanitizer** for commonly used **ML model binaries** (i.e. models built using PyTorch or Tensorflow). Although a general-purpose sanitizer is likely infeasible, limiting our scope to ML model binaries containing malicious weight pickles allows a sanitizer to leverage possible structure when reconstructing a model's weights. Thus, a sanitizer can be produced, which will produce a sanitized model binary that can be run safely and efficiently on any machine.

Context:

Marco Slaviero's Black Hat 2011 presentation *Sour Pickles* brought notoriety to Pickle as a insecure serialization tool due to the presence of the GLOBAL and REDUCE opcodes in the Pickle Virtual Machine (PVM). Assembly-like, human-unreadable code is run within the PVM, with GLOBAL used to load arbitrary python objects onto the stack and REDUCE used to call a stack-bound python object's __reduce__ dunder method for reconstruction. Keeping these opcodes in mind, we identify three possible exploits when constructing a malicious pickle: arbitrary pickle reconstruction, __setstate__ class injection, and eval/exec abuse.

Arbitrary pickle reconstruction is simply modifying a pickle file freely as demoed in [3]. New GLOBAL/REDUCE opcode calls can be introduced by a malicious entity, with the modified pickle file executing code of a hacker's choice. However, all is not lost against a malicious entity, as executable code must be tied to a python class imported alongside the "unpickling" code; that being said, this limitation can be circumvented by both a malicious python author and a more creative malicious entity. Evan Sangaline identifies a vulnerability tied to the __setstate__ dunder method used when reconstructing a pickled object, where code in this method will be

run freely during reconstruction [2]. A malicious python author could inject their own code into these methods, executing arbitrary code in the process. However, a malicious entity with no knowledge or access to a pickle file's "unpickling" environment is not out of options. Yanic Kilcher demos an eval/exec exploit that abuses the REDUCE method alongside the eval() and exec() methods in python to execute python code encoded as a string within a pickle file silently, giving malicious entities an attack avenue with little prior knowledge of the "unpickling" environment [7]. Thankfully, evaluation tools for insecure pickles exist to detect possible attacks and mitigate their effects. Fickling comes to mind as an effective pickle attack mitigation tool, providing a Fickling-specific PVM for running pickle code safely and warning messages for potentially harmful pickle files [4]. However, no sanitation is performed on pickle files run through fickling, relying on its proprietary, python-based PVM and warning messages to keep "unpickling" environments safe.

Ben Frederikson [5] and Ned Batchelder [6] separately condemn the use of Pickle for serialization and persistence, pointing out its speed, space efficiency and security issues and recommend using other serialization methods instead.

Approach:

Since it is clear that the problem of unpickling files is a real issue, the question becomes, what can be done about it? While the obvious solution is not to unpickle any untrusted file (which is also a disclaimer on both the pickle module documentation and PyTorch documentation), it is often not followed in practice. Downloading and unpickling pre-trained models from platforms such as GitHub or HuggingFace is a ubiquitous practice today, which essentially translates into running arbitrary, untrusted and insecure code on one's device.

Brunson et al. [4] developed a python library called fickling that detects malicious pickle files. Fickling can accept pickled data streams and decompile them into human-readable Python code that, when executed, will deserialize to the original serialized object. While their approach is similar to what we intend on implementing, their library is implemented in Python, which comes at a considerable performance trade-off. We plan on implementing our sanitizer in C++. Since C++ is a compiled language, we expect our compiler to be much more efficient. Slaviero et al. [1] introduces ananpickle, another sanitizer similar to the one described above. Anapickle is capable of accepting, examining and manipulating pickle files. It suffers from the same drawbacks as Fickling, i.e., performance issues. Furthermore, both Fickling and Ananpickle can be utilized exclusively for the detection of malicious pickle files. The sanitizer we plan on implementing will go a step further to not just detect malicious pickle files but attempt to fix them. Due to the frequent corruption of pickle files when modifications are performed, this is a challenging process.

To conclude, the advantages our sanitizer offers over previous works are two-fold; firstly, we attempt to improve on performance over the previous studies by implementing our sanitizer in C++. Secondly, we go a step further to not just detect malicious pickle files but also attempt to

fix them. There is no tool in our knowledge available today that can offer this two-fold advantage.

Evaluation:

Being unable to easily find malicious pickle files online makes it difficult to evaluate our sanitizer. Even if we were to come across such malicious pickles, trying to download and test our sanitizer on these models would be a very dangerous endeavor. On the other hand, if we attempt to inject malicious code into our own pickle files, it introduces a bias into the evaluation process.

To counteract this problem we have decided to evaluate our sanitizer using the red team - blue team approach [8]. The red team will consist of one team member who will not be familiar with the working of our sanitizer. This member would solely focus on creating malicious pickle files of different classes that would simulate attacks on host machines. The rest of our team members (blue team) would focus on implementing the sanitizer that can detect any class of malicious pickles and protect the host machine.

Hugging Face hosts a variety of model binaries in both Pytorch and Tensorflow that can be easily downloaded. We intend to recreate all three arbitrary pickle reconstruction, __setstate__ class injection, and eval/exec attacks on the top 50 most downloaded PyTorch and top 50 most downloaded Tensorflow models and evaluate the compatibility of our sanitizer. As explained above, our recreated attacks will be constructed by a sanitizer-blind subset of our group. If a model can be constructed and successfully run once with 1-5 sample input(s), we will consider that model as compatible.

Scope:

Pickle sees mass adoption across various Python applications, all of which have Pickle files with differing contents and structures. We choose to limit our project's scope to sanitizing pickle files used in popular machine learning applications, specifically those used in PyTorch. We focus strictly on sanitization for this project, choosing to leave potential security changes in Python, Pytorch, Tensorflow, and Pickle to their respective developers.

Checkpoints:

(Week 6) Sep 26th - Oct 3rd:

- Everyone
  - Split members into sanitizers/exploiters
  - Get familiar with Pickle PVM and Opcode Execution

(Week 7) Oct 3rd - Oct 10th:

- Everyone:
  - Create code to locate and extract weights pickle from model binary
  - Study structure of PyTorch/Tensorflow pickles

(Week 8) Oct 10th - Oct 17th (Midterm Week):

- N/A

(Week 9) Oct 17th - Oct 24th:

- Sanitizers:
  - Detect and remove arbitrary pickle reconstruction attacks
  - Detect and remove eval/exec attacks
- Desanitizers:
  - Create automated script that creates randomized pickle reconstruction attacks
  - Create automated script that creates eval/exec attacks

(Week 10) Oct 14th - Oct 31st:

- Sanitizers:
  - Detect and remove __setstate__ attacks
- Desanitizers:
  - Create automated script for __setstate__ attacks

(Week 11) Oct 31st - Nov 7th:

- Sanitizers:
  - Continue detecting and removing __setstate__ attacks
- Desanitizers:
  - Start creating automated hugging face evaluation tool

(Week 12) Nov 7th - Nov 14th:

- Everyone:
  - Continue creating automated hugging face evaluation tool

(Week 13) Nov 7th - Nov 14th:

- Everyone:
  - Continue creating automated hugging face evaluation tool
  - Gather preliminary results

References:

[1] Marco Slaviero. 2011. Sour Pickles: Shellcoding in Python's serialization format. Retrieved from https://media.blackhat.com/bh-us-11/Slaviero/BH_US_11_Slaviero_Sour_Pickles_WP.pdf

[2] Evan Sangaline. 2017. Dangerous Pickles - Malicious Python Serialization. Retrieved from https://intoli.com/blog/dangerous-pickles/

[3] Nelson Elhage. 2011. Exploiting misuse of Python's "pickle". Retrieved from https://blog.nelhage.com/2011/03/exploiting-pickle/

[4] Evan Sultanik. 2021. Never a dill moment: Exploiting machine learning pickle files. Retrieved from https://securityboulevard.com/2021/03/never-a-dill-moment-exploiting-machine-learning-pickle-files/

[5] Ben Frederikson. 2014. Don't pickle your data. Retrieved from https://www.benfrederickson.com/dont-pickle-your-data/

[6] Ned Batchelder. 2020. Pickle's nine flaws. Retrieved from https://nedbatchelder.com/blog/202006/pickles_nine_flaws.html

[7] Yannic Kilcher. The hidden dangers of loading open-source AI models. Retrieved from https://www.youtube.com/watch?v=2ethDz9KnLk&t=562s

[8] Jon Boyens. Notional Supply Chain Risk Management Practices for Federal Information Systems. Retrieved from https://csrc.nist.gov/glossary/term/red_team_blue_team_approach