Project Overview

- **PowerShell** - common target for cyberadversaries; can be obfuscated and executed from memory
- **Obfuscations** - different code but same functionality; defeat text-based approaches
- **Abstract Syntax Tree (AST)** - abstracts away code's specific details while retaining control flow and content-related information

Current progress

- Three types of obfuscations: AST, TOKEN and STRING; available from online tool Invoke-Obfuscation¹
- Dataset - obtained from Palo Alto Networks²; originally 4079 datapoints, 469 after preprocessing
- Train Random Forest $R_B$ with hand-engineered features
- Train Random Forest $R_E$ with learned representations from tree-structured VAE
- Compare performance on both natural and obfuscated dataset

Observations

- The learned representations are robust against AST and TOKEN but not STRING obfuscations
- Further investigation lead to the fact that STRING obfuscations transform the code in a very specific manner where the code is converted to a string and is passed to IEX command, similar to the `eval` procedure in most programming languages. This resulted in very similar ASTs of very few nodes, which explains the failure of the STRING obfuscations observed both qualitatively and quantitatively.

Relevant links


Open questions

- **Dataset** - need for larger, curated, labeled dataset that can be used for PowerShell malware detection and classification
- **AST engineering** - revealed shortcomings when applied to STRING obfuscations
- **De-obfuscation** - ML projects would require data preprocessing component where de-obfuscation might be essential

Future vision

- **Stronger baseline** - define a baseline that uses more complex features
- **Supervised learning** - try out supervised representation learning methods
- **Adversarial learning** - use obfuscated samples during training
- **Other languages** - explore languages other than PowerShell, (C, Python etc)