Poster: Dissecting the Cryptographic Code Exchange

Jason Ly
The University of Adelaide
Australia

Minhui Xue
The University of Adelaide
Australia

***Abstract*—Code exchanges are the common choice of software documentation of developers regardless of their experience. These places form communities of programmers and professionals that exchange and offer solutions to a user's problems. This acts as a valuable resource to anyone seeking quick and practical solutions to any of their code related problems. However, with higher usability, users tend to sacrifice security. The copy and paste culture surrounding code exchanges has caused developers to implement potentially insecure code which may lead to security breaches in their applications. Due to its popularity, it is essential that this problem is minimised through a simplification of the code retrieval process to mitigate poor choices in code searching. This solution is presented via a system based off the 'nudge theory'.**

I. Introduction

Programmers of all levels have the tendency to rely on the work of others. Reuse of components and in-built libraries are several examples of the reliance and inactivity that takes place towards software development [2]. This, however, is not necessarily a negative trait as code-reuse is an important culture that supports lower development times and improves functionality [3]. In turn, it is understandable why communities such as Stack Overflow act as an essential tool through their example driven solutions. This access to a knowledge rich platform supported by a strong community forms an enticing resource for inexperienced to professional developers alike. It increases the functionality of software by reducing the effort of implementation which is ideal in all levels of software development.

However, when dealing with complex concepts such as cryptography, it is observed that its benefits may not counter weigh its potential risks. Specifically, the risks that arise from the implementation of insecure code into real-world applications are investigated. On one hand, a developer will likely reap the benefits of Stack Overflow's technical solutions [2]. On the other, they increase the likelihood of introducing the potential for malicious attacks. Oftentimes Stack Overflow users cannot rely on its community to properly identify vulnerabilities in solutions due to the specialised nature of cryptography [4]. This leads to shortcomings in the reliability of the mentioned examples and the overuse of compromised solutions in streamline applications. The repercussions of such dangers in working applications form the catalyst for the implementation of a deep learning nudge system.

To solve this, we approach the problem via supervised deep learning systems, which are superior in comparison to rule based filtration. This systems aims to make predictions through the analysis of user submitted code snippets to these code exchanges. However, rather than traditional networks, we use a Long short term memory (LSTM) network to predict security. The introduction of memory via its interaction gates adds a level of understanding to the system, to allow for more dynamic predictions. This method aids in accounting for the huge variance in potential input, due to the uniquely large and non-numerical inputs.

II. Implementations

For the implementation of our LSTM network, we utilise Python as the chosen language, attributing this choice to its extensive data manipulation libraries. To create a baseline for the network, we would first need to build a standard neural network, in our situation of 'shallow' depth, since LSTMs do not require many hidden layers. This is due to their introduction of the control gates, cell states and their weightings, meaning we do not require the network to be large as these interactions will introduce memory to the network through encoded states. Our forget gate, will help the network decide what features it forget based on the new input. The input and output gates dictate what information will be encoded into our new cell state and passed forward to the next layer. The cell state is a single vector that holds previously encoded data of features or patterns that have been 'remembered' by our network. These additional interactions throughout the network allow more control over the gradients, and prevents the vanishing gradient problem [1].

For the network to successfully predict a program's security, we must first have an appropriate data set that can be fed in as input. The task of converting an entire code program into a binary vector, while maintaining its crucial features to reflect the same program is extremely difficult. We generally need complete programs/source code, however stack overflow includes many incomplete systems. Hence, we need a Partial Program Analysis (PPA) tool. PPA was built specifically to be able to parse incomplete code to build abstract syntax trees (AST), which are the first intermediate form needed. The function of being able to parse programs independent from the completeness of the code allows for small code snippets to be analyzed and labelled. From here, we will have

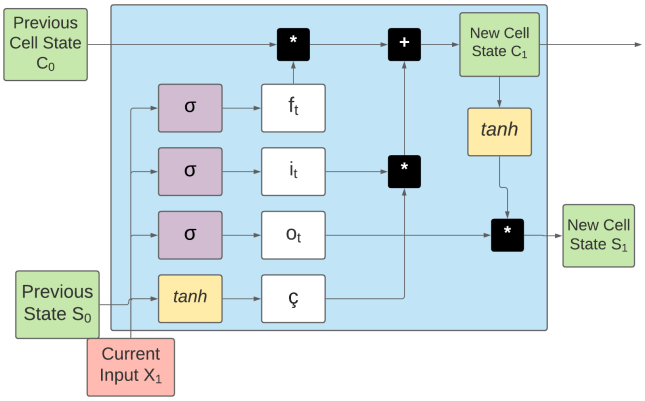


Fig. 1: Example of LSTM network, with added cell state input and outputs, and interaction gates.

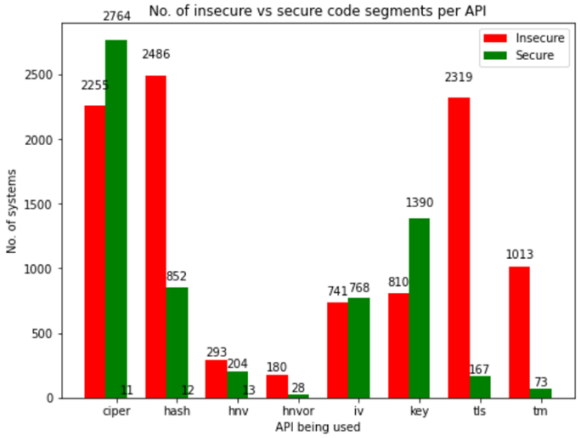


Fig. 2: Cost vs. Iterations (per hundreds)

our AST, where using a related tool e.g. Node2Vec. We can traverse our tree and build a one-hot binary vector. This form of data can then be fed into our network and be mathematically manipulated for analysis.

In terms of labelling, systems would be dedicated as secure or insecure based on their similarity to other secure or insecure programs. These would be attributed to similarities in their use-case patterns produced in the PDG, in other words if a program shares similar programming practices to a pre-determined secure program, we can deduce that new program is also secure.

A. *Programs were labelled based on:*

- Systems containing updated and strong, mainstream algorithms for symmetric cryptography.
- Systems containing obviously insecure code, e.g. using outdated, weak algorithms or static initialization vectors and keys for symmetric cryptography.
- Systems that contain code whose security required additional developer input.

Because of this, it is essential we properly teach the network with an extensive training set over many iterations. This gives the network the opportunity to build a strong baseline for its predictions, and gives room to learn dynamically based on successful predictions to become an unsupervised model.

III. Results

In our test setup, we ran our network over the training data set, feeding each transformed code snippet as input. After each iteration, the network would decide what key features of 'strong' cryptographic code would be encoded into its 'input gate' vector to be passed to the next iteration.

The network's final performance, over a 20000 iteration execution resulted in a final training accuracy of 99.04%. This was run with a pre-labeled data set provided by Fischer et al. [1] of a stack overflow code dump, where code snippets were filtered if they held keywords relating to cryptography. This dataset was manually labeled by security experts and laid a strong baseline for the training of our network. Expanding on this, over the run time, our network's cost undergoes a downward trend finishing at 0.005, that is to say, as our network iterates, its probability of predicting correctly increases, thus proving functionality.

IV. Conclusion

A deep learning-based system, that utilises both neural networks, and the concept of memory, is a quintessential tool in making up for the short-comings in the standard "rule-based filtration" approach. Its ability to learn and predict a code's safety based on its usage patterns means quick analyse times. By investigating the process of its recommendations to stronger cryptography, we can understand the system and seek to make improvements and tweaks to improve many output factors such as analyse times or prediction accuracy.

Acknowledgments

This work was supported in part by the Australian Research Council (ARC) Discovery Project (DP210102670) and the Adelaide Summer Research Scholarship.

References

[1] F. Fischer, H. Xiao, C.-Y. Kao, Y. Stachelscheid, B. Johnson, D. Razar, P. Fawkesley, N. Buckley, K. Böttinger, P. Muntean, et al. Stack overflow considered helpful! Deep learning security nudges towards stronger cryptography. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019.

[2] M. Sojer and J. Henkel. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems*, 2010.

[3] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and GitHub: Associations between software development and crowdsourced knowledge. In *International Conference on Social Computing*. IEEE, 2013.

[4] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In *NDSS*, 2012.

Poster: Dissecting the Cryptographic Code Exchange

Jason Ly
The University of Adelaide
Australia

Minhui Xue
The University of Adelaide
Australia

Introduction

Oftentimes Stack Overflow users cannot rely on its community to properly identify vulnerabilities in solutions due to the specialized nature of cryptography. Because of this, we propose an integrated learning system onto a code exchange such as stack overflow. Our network aims to learn key features of strong cryptography and predict their security based off its components.

Data labelling

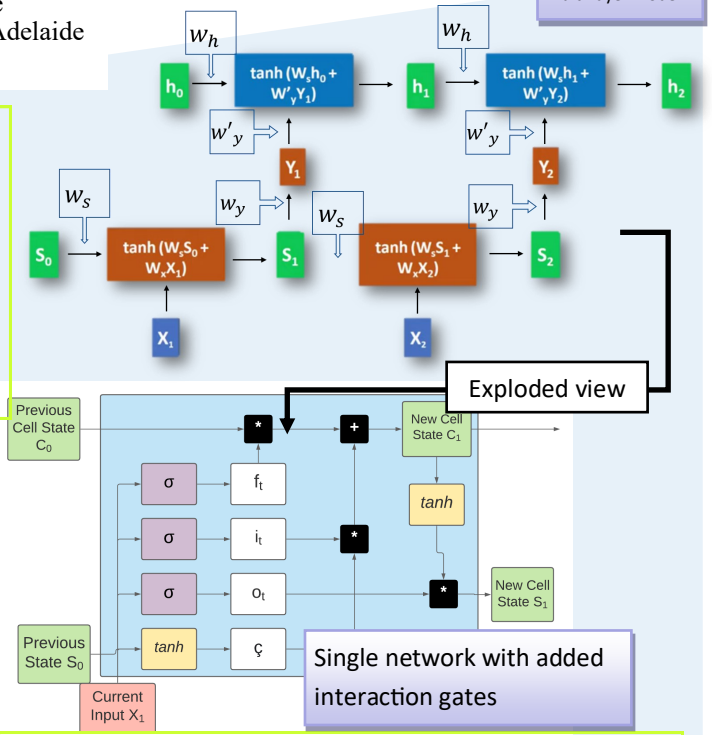
- Systems containing a combination of updated and strong, mainstream algorithms for symmetric cryptography (AES, CBC,).
- Systems containing obviously insecure code, e.g. using outdated, weak algorithms or static initialization vectors and keys for symmetric cryptography (DES, ECB, SHA-1).
- Systems that contain code whose security required additional developer input.

Motivations

- Programmers have the tendency to confide in the work of others through the reuse of proven and tested implementations.
- Code exchanges such as Stack Overflow do not have enough security experts to properly verify most cryptography-based solutions
- Attempts to guide users are observed to be more effective when their choices can be influenced rather than restrict their access.

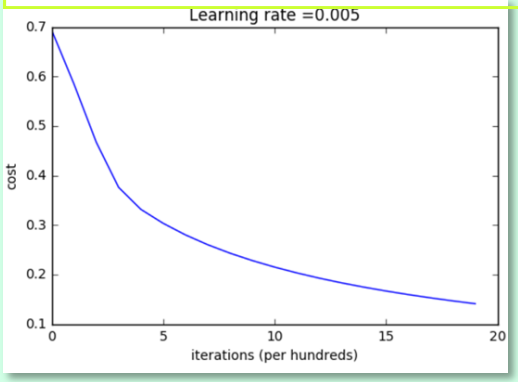
We define our state (S_T) at time (T) , as our additional input. We can view it as a function of the previous state (S_{t-1}) at time T-1 and current input X and time T.

- $S_T = F_w(S_{T-1}, x_T)$
- $S_T = tanh(w_s S_{T-1}, w_x x_T)$ ←tanh activation
- $y_T = w_y S_T$ ←Weight * activation



Proof of concept

Successful training of RNN to recognize and predict program accuracy with a 99.04% prediction rate



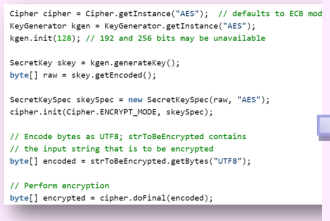
Key takeaways

- Properly safe guarding code exchanges is essential to beginners and professionals alike.
- Identifying the specific shortcomings in code snippets opens way for a huge amount of variance.
- It is essential we aim for dynamic learning compared to rule-

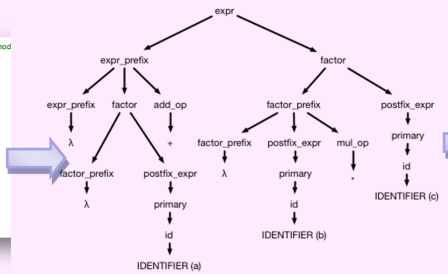
Data Conversions

To train our network on code snippets, we must undergo some form of data conversion to feed them into our network.

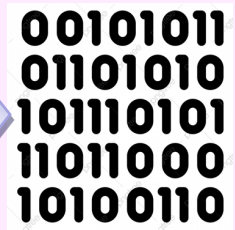
Converting entire code programs into binary vectors, while maintaining the crucial, defining features is extremely difficult. To solve this, we utilize a multi-step transformation method. As shown:



Code snippet



Abstract Syntax Tree



Binary Vector

References

[1] Prole, K. (2018). '*Code reuse: How to reap the benefits and avoid the dangers*.' [online] Code Dx, Available at: https://codedx.com/blog/code-reuse-how-to-reap-the-benefits-and-avoid-the-dangers/

[2] Fischer F, Xiao H, Kai C, Stachelscheid Y, '*Stack Overflow Considered Helpful! Deep Learning Security Nudges Towards Stronger Cryptography*.' Security Symposium (USENIX Security 2019)

[3] Bogdan V, Vladimir F, Alexander S, '*StackOverflow and GitHub: Associations Between Software Development and Crowdsourced Knowledge.*' Socialcom 2013

[4] James, A. (2019). *Java SE | Oracle Technology Network | Oracle*. [online] Oracle.com. Available at: http://www.oracle.com/technetwork/java/javase/