INCORPORATING AI INTO SOFTWARE REVERSE ENGINEERING COURSES

Xiuwen Liu and Mike Burmester

Department of Computer Science

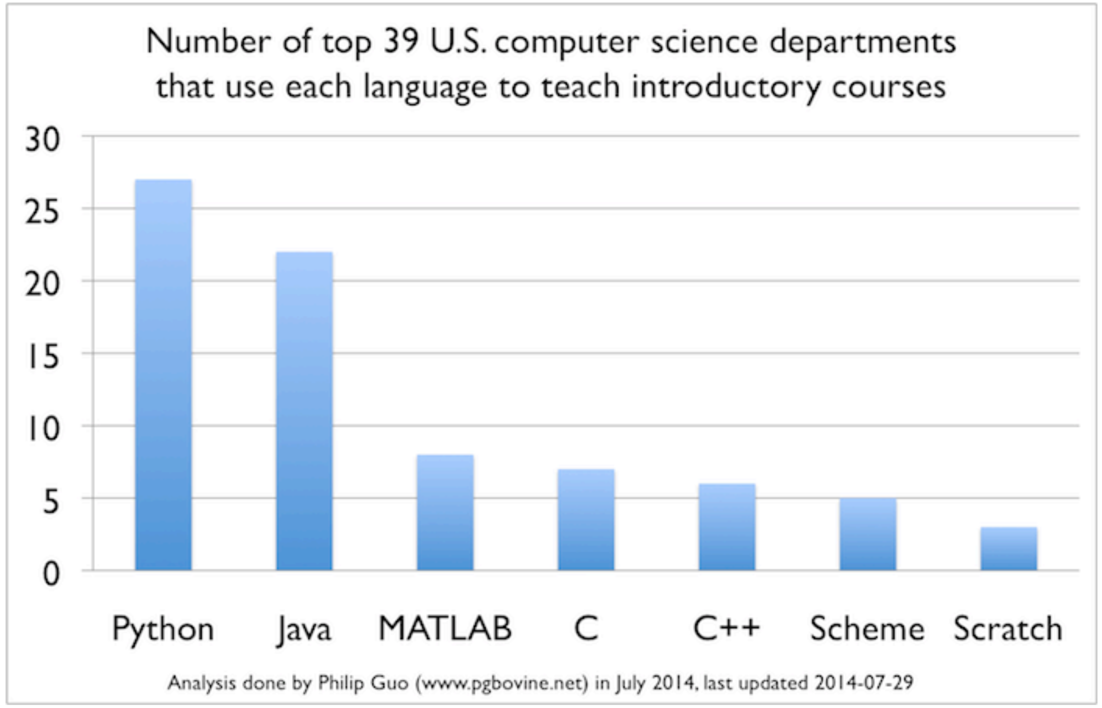
Florida State University, Tallahassee, FL

INTRODUCTION - BACKGROUND

- Software becomes the key components in many systems and its complexity has been increasing to meet context-dependent requirements
  - Unfortunately, as a result, there are many new vulnerabilities and new attack surfaces
  - As the least secure component of the digital ecosystem, user errors continue to contribute to the cyber incidences
- Consequently, reverse engineering of binaries is a fundamental skill
  - It is crucial for analyzing malware
  - Furthermore, it has a variety of other applications
- The challenge is that it is difficult to master reverse engineering skills
  - Programs are inherently flexible and complex at the binary level and analysts often spend long hours to figure out

INTRODUCTION – BACKGROUND – CONT.

- Curricular in typical computer science programs make teaching software engineering skill more challenging
  - To meet the demand of software developers, how to improve software development productivity becomes the top priority of computer science programs
    - By enabling and encouraging code reuse using libraries and higher-level programming languages

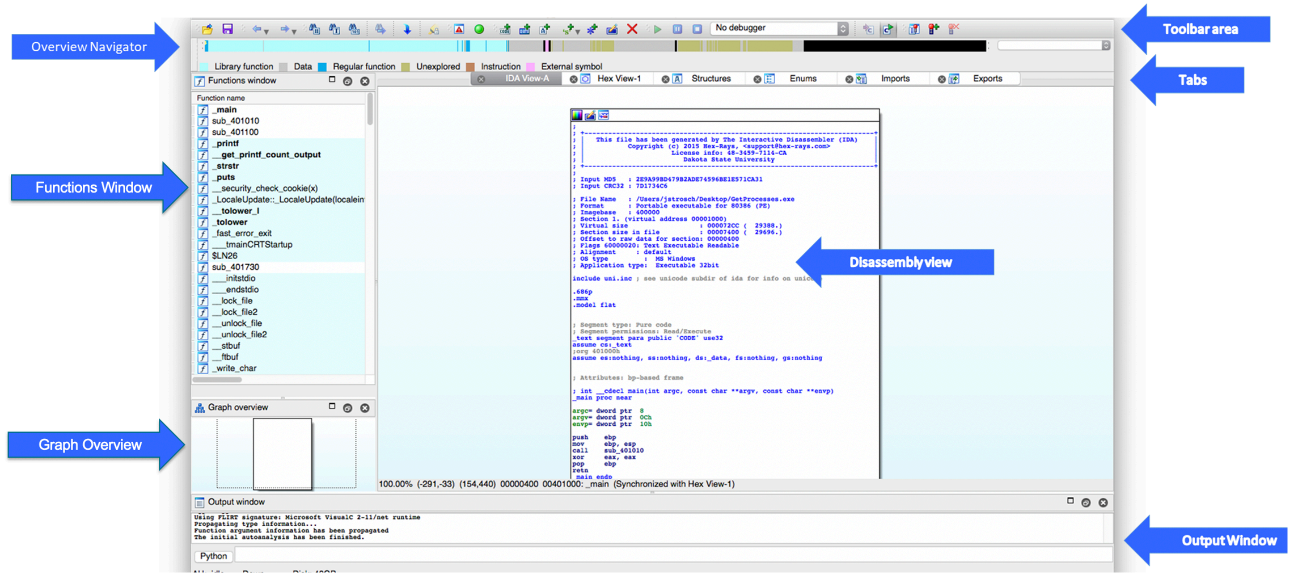


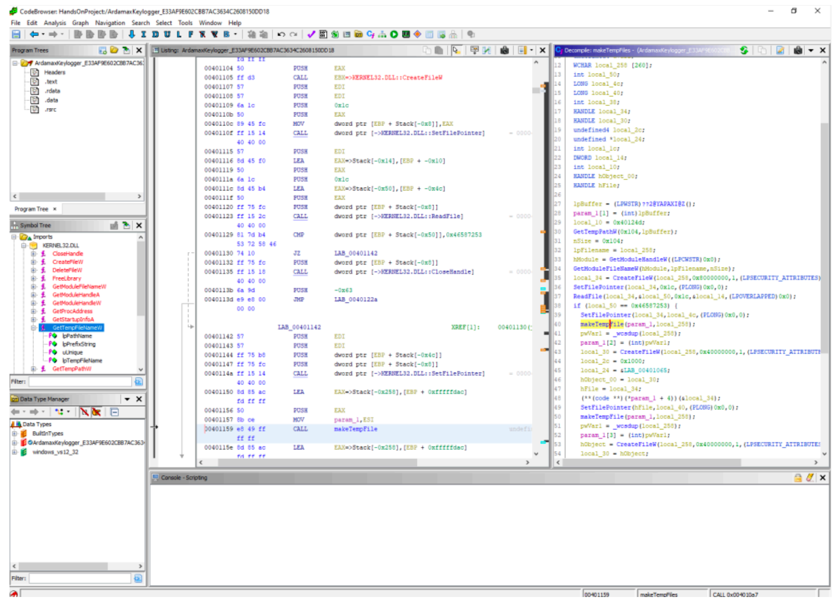
INTRODUCTION – BACKGROUND – CONT.

- One side effect is that computer science students have little exposure to low-level concepts and may not appreciate vulnerabilities and their potential impacts
  - For example, the students graduated with a computer science degree now may not know that functions are implemented using calling conventions
    - They may know the instruction set architecture concept, but may not have written any programs in assembly
    - We are aware the efforts of requiring the students in CAE to have such exposure so that the students could be able to code securely
- The challenge is how to educate capable analysts that can analyze new malware efficiently and effectively

CURRENT SITUATION

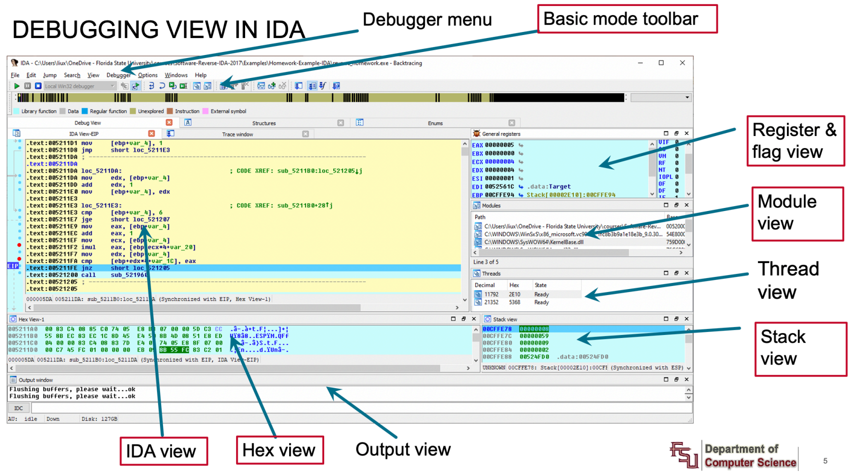
- There are very useful and powerful tools to help analysts analyze malicious and binary programs and identify vulnerabilities
  - Such as Ghidra and IDA Pro
  - We found that most students use both IDA Pro and Ghidra jointly
    - Signature matching capabilities in IDA Pro enable it to identify many functions
    - Code review in Ghidra makes easier to understand the logics in the binaries





CURRENT SITUATION – CONT.

- Students often create C or C++ programs for functions so that they can test their ideas and generate running cases
  - For dynamic analysis, the debugging capabilities in IDA are very helpful as well
  - Even though Ghidra offers emulation capabilities, they are available only via APIs and most students found them cumbersome to use
- Overall, we have observed that the current tool set allows our students to learn static and dynamic analyses and use them for malware analyses effectively
  - The scripting and programming capabilities are very important as well



BEYOND MANUAL ANALYSIS

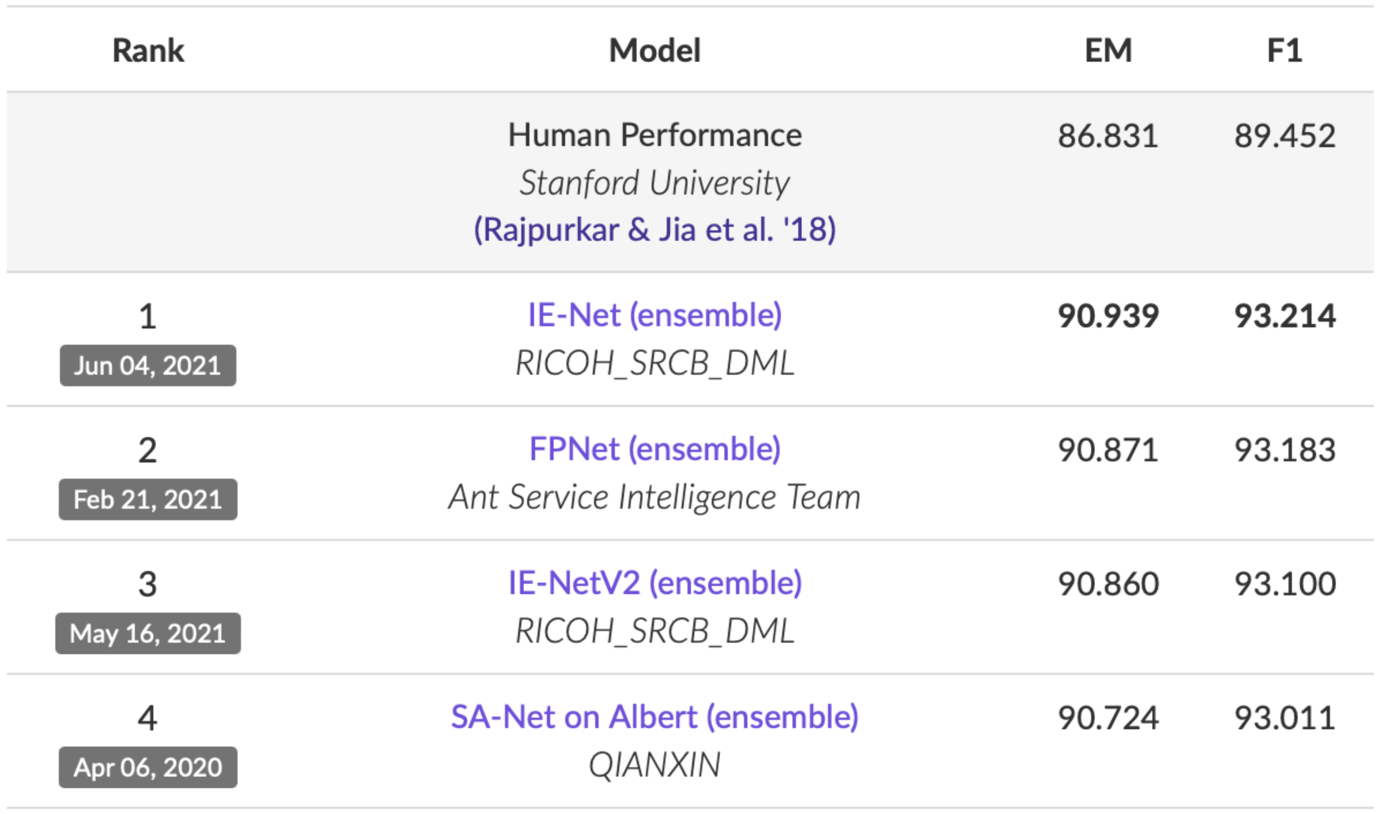
- However, software reverse engineering is still dominantly a manual process
- Given the rapid developments in artificial intelligence and in particular machine learning, it is highly desirable to be able to incorporate such advanced capabilities into reverse engineering routines
  - Some techniques such as symbolic execution have been around for some time and recent developments make them more applicable
  - Machine learning and deep learning are improving natural language processing, object detection, and speech recognition significantly
    - Applying these techniques to software reverse engineering could be fruitful
  - Graph neural networks are particularly suitable for representing multiple relationships in programs
    - They could lead to better software reverse engineering tools

SYMBOLIC EXECUTION

- Symbolic execution techniques are very appealing conceptually
  - The precision and coverage tradeoff for binary program analysis is well understood
  - In order to systematically understand a malware sample, symbolically enumerating all execution possibilities can be helpful
    - However, there is a mismatch between cases analysts expect and cases by symbolic execution techniques
      - For example, a for loop can generate many different branches by unrolling the loop
      - But analysts would like to treat that as a loop unless it is vulnerable
  - There are also practical issues when using them to analyze whole binary programs
    - For examples, system and library functions cause issues for typical symbolic execution routines

REPRESENTATION LEARNING USING MACHINE LEARNING

- In recent years, transformer-based natural language processing models such as BERT have surpassed humans in a number of challenging tasks
  - For example, see the Squad leaderboard at https://rajpurkar.github.io/SQuAD-explorer/
  - It is highly desirable to apply these techniques to binary program analyses

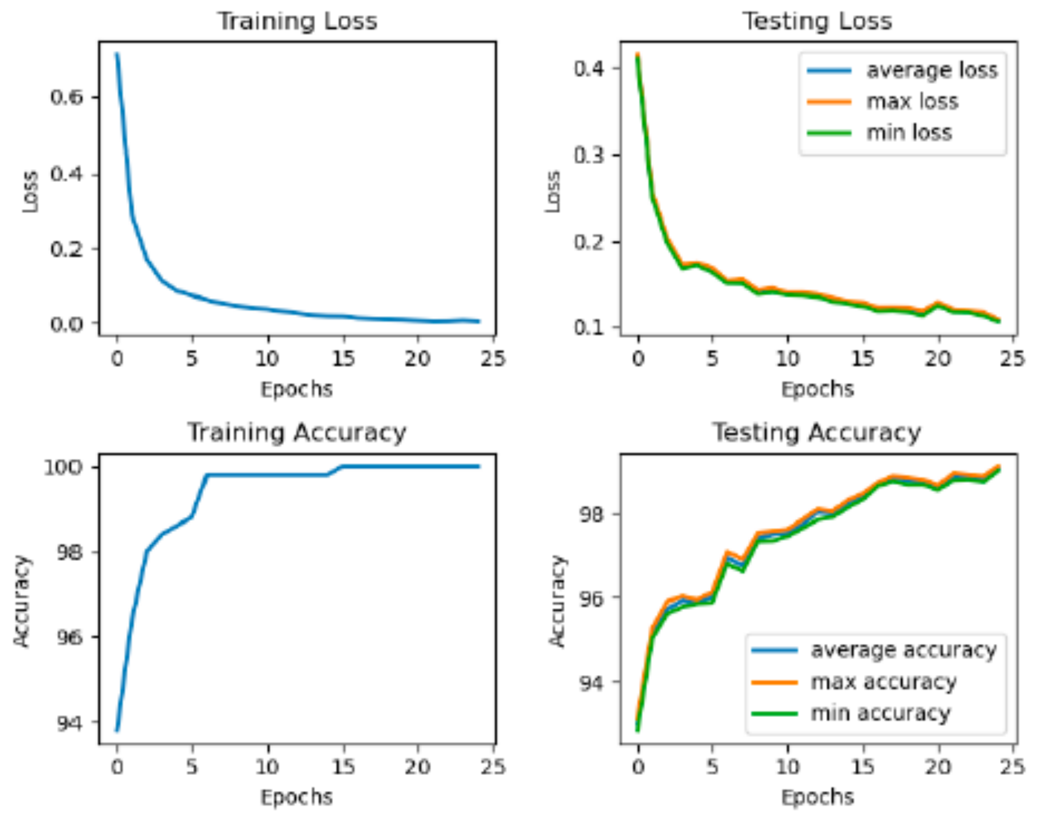


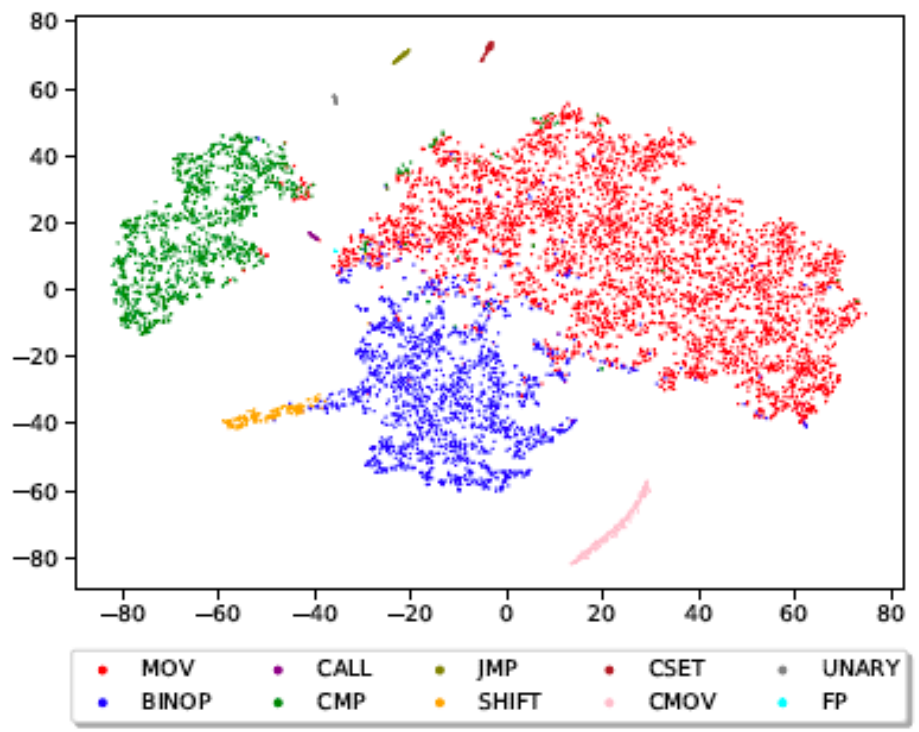
Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Jun 04, 2021	IE-Net (ensemble) *RICOH_SRCB_DML*	**90.939**	**93.214**
2 Feb 21, 2021	FPNet (ensemble) *Ant Service Intelligence Team*	90.871	93.183
3 May 16, 2021	IE-NetV2 (ensemble) *RICOH_SRCB_DML*	90.860	93.100
4 Apr 06, 2020	SA-Net on Albert (ensemble) *QIANXIN*	90.724	93.011

Department of Computer Science

REPRESENTATION LEARNING USING MACHINE LEARNING – CONT.

- Fundamentally, these techniques learn a vector representation for tokens, instructions, basic blocks, and functions to facilitate downstream tasks
  - Recently we have used metric learning to successfully improve the embeddings
  - However, it may not be realistic to expect students in software reverse engineering to know such techniques and be able to apply them effectively





BRIDGING SEMANTIC GAPS AUTOMATICALLY

- As analysts and programmers understand programs using meaningful variable and function names much better, it seems a good idea to utilize the available source code to learn patterns so that lost semantics in stripped binaries can be recovered
  - With the recent techniques of graph neural networks to model multiple syntactic and semantic relationships between instructions, learned instruction embeddings and graph representations will likely lead to substantial improvements in automatically recovering human readable variables and function names
    - Recent work has demonstrated that function names can be recovered from stripped binaries with an F1 score of 0.45
    - More interpretable functions could be synthesized based on available input-output relationships
- However, they are still not readily useful to analysts

INCORPORATING AI INTO REVERSE ENGINEERING COURSES

- While AI and machine learning techniques have shown promise for a number of software reverse engineering tasks, how to train and equip analysts to be able to use such techniques to analyze a new unknown binary sample is a challenge
  - It seems that it is not realistic to incorporate such advanced topics to a software engineering course
    - Even though elements of such topics can be covered
    - Encouraging students to explore them in a term project seems a good strategy after they are introduced
  - Developing separate courses that focus on AI and machine learning techniques for reverse engineering can be an option
    - However, it may be necessary to have an advanced machine learning course as a prerequisite, which, however, will limit the student pool significantly

SUMMARY

- In the last few years, we have tried to incorporate AI and machine learning techniques into our software reverse engineering course
  - Symbolic execution techniques are useful but they must be used via available libraries and APIs
  - For instruction and other embeddings based on natural language processing models, the analysts need to learn and be familiar the techniques in order to incorporate them into their analysis routines
  - Graph neural networks offer great potential to learn better and more effective models by modeling multiple syntactic and semantic relationships among instructions
    - However, learning and using such models is challenging
- We hope we can explore the issues collectively and figure out an effective way to take advantage of the recent and exciting development in AI and machine learning