



### INCORPORATING AI INTO SOFTWARE REVERSE ENGINEERING COURSES

Xiuwen Liu and Mike Burmester Department of Computer Science Florida State University, Tallahassee, FL

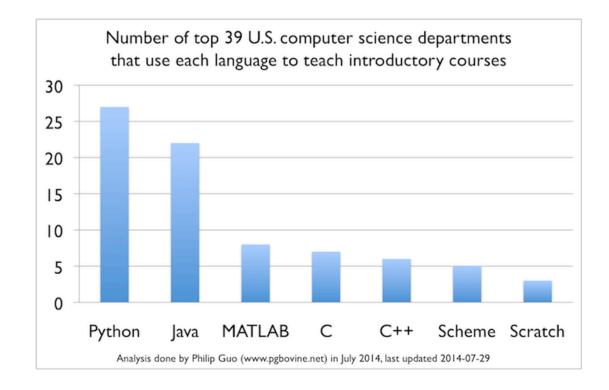
### **INTRODUCTION - BACKGROUND**

- 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

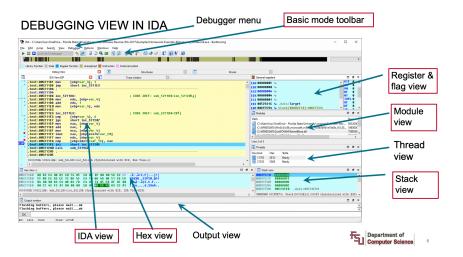
W Navigator	Instruction External symbol	
Functions window	© IDA View-A @ O Hex View-1 @ A Structures @ ⊞ Enums @ ∰ Imports @ [P Exports	Tabs
Function name		
7 main 7 sub_401010 7 sub_401100	■ m m = 1	
get_printf_count_output	This file has been generated by The Interactive Einsembler (TDA) Copyright(c) 2015 Har-Mary, comporther.ruy.com> License info 44-305-7134-CA Babas Eute Durenzhy	
ons Window	1 1 Japan 105 - 1 289498047982A08745968818571CA31 1 Japan CRC32 : 7D1734C6	
	2 File Rams : //Werr/j/Hortech/Deutop/Get/Forcesse.ess Fromat : Portal end : Porta	
7tmainCRTStartup 7 \$LN26 7 sub_401730	o offest to real data for section: 000000 relief of 000000 traits includes instability of type of type type increasing systems type increasin	
Jinitstdio	isclude uni.isc ; see unicode subdir of ida for info on unica	
endstdio    lock_file	-686p -MRK -model flat	
F _lock_file2	.model flat	
<u> </u>	j Segmant type: Pure code 1 Segmant permissions: Read/Execute	
	text segment para public 'CODE' use32	
Fftbuf	jorg 401000h assume essenthing, senothing, ds:_data, fs:mothing, gs:mothing	
📝 _write_char		
	) Attributes: bp-based frame	
	) intcdecl main(int argc, const char **argv, const char **envp) _main proc near	
A Graph overview 🛛 🕲 😋	arge- downd ptr 8 arge- downd ptr 0ch envy- downd ptr 10h	
Overview	2022 Aug 2022 2022 A	
100.00% (-291,-33) (1	54,440) 00000400 00401000: main (Synchronized with Hex View-1)	
Output window		0 0
Wiley Filts signature: Microsoft Visual2 2-11/net rustime Propagating type information Function argument information has been propagated The initial autoanalysis has been finished.		Output Wi

Program Trees 🔚 🍪 🏷 🗙	E Usting: Arden	axKeylogger_E33AF9E602C88	74C3634C260815000	D18 🕥 💼 🔽 🐺 🔀	🔒 🗐 •	×	Decomplet makeTempFiles - (ArdamaxKeylogger_E33AF9660000) 🥵 🗈 🔀 🏫 -
ArdamaxKeylogger E33AP9E602C887AC36	1	IG II II				^ I12	
- B Headers		00403104 50	POSH	EAX		13	
.text		00401105 ff d3	CALL	EBS->RESNEL32.DLL:(CreateFileW			LOBO local 4cr
- [2] .rdeta		00401107 57	P058	EDI			1000 local 40;
-D .deta		00401108 57	POSH	Colo Colo		14	int local 30g
- D vac		00401109 68 10	PUSH	URIC		17	HANDLE local 34;
		00401100 00 45 50	PUGR MOV	deard ptr (ESF + Stack(-5x5)1.EAX		18	HANDLE local_30;
		0040110f ff 15 14	CALL	dword ptr [->HTRHEL32.DLL::SetFilePointer]		19	undefined4 local_2c;
		40 40 00				21	undefined *local_24;
		00401115 57	POSH	EDI		21	int local_los
		00401116 8d 45 f0	LEA	EAX=>Stack[-0x14],[EBP + -0x10]		22	DWORD local_14;
		00401119 50	PUSH	EAX		21	int local_10;
		0040111a 6a 1c	PUSH	Oxic		24	HANDLE hObject_00;
		0040111c 8d 45 b4	LEA	EAX=>Stack[-0x50],[ESP + -0x40]		25	HANDLE hFile;
>		0040111f 50	POSH	EAX		28	loguffer = (LSWITR) ==28YAPAXI82();
Program Tree ×		00401120 ff 75 fc	POSH	dword ptr [EBP + Stack[-5x8]]			param 1[1] = (int)lsBuffer;
		00401123 ff 15 2c	CALL	dword ptr [->RERNEL32.DLL::ReadFile]			local 10 = 0x40124dz
Symbol Tree 📑 🎦 🗙		40 40 00 00401129 81 7d b4	010				GetTempPathW(0x104, inBuffer);
imports ^		00401129 81 7d 54 53 72 58 4		dword ptr [ESP + Stack[-0x50]],0x46507253			nSize = 0x104;
KERNEL32.DLL     KERNEL32.DLL     KERNEL32.DLL		00401130 74 10	JZ	LAB 00401142		32	lofilename = local 258;
E f CreateFleW		00401132 ff 75 fc	DOWN	dword ptr [ESP + Stack[-0x0]]		33	hModule = GetModuleEandleW((LECWSTE)0x0);
B 1 DeletefileW		00401135 ff 15 18	CALL	dword ptr [->HERNEL32.DLL::CloseHandle]		<b>1</b> 24	GetModuleFileNameW(NModule, lpFilename, nSize);
8 £ Precibrary		40 40 00	10000			- 25	local_34 = CreateFileW(local_250,0x00100000,1,(LPSECURITY_ATTRIBU
£ GetModuleFieNameW		0040113b 6a 9d	POSH	-0x63		<b>a</b> 36	SetFilePointer(local_34,0x1c,(FLONG)0x0,0);
GetModuletiandleA     GetModuletiandleW	L : I	0040113d e9 e8 00	3952	185_0040122a		37	ReadFile(local_34, slocal_50, 0x1c, slocal_14, (LROVERLARPED) 0x0);
E GetProcAddress		00 00		-		1 St	if (local_50 == 0x46587253) {
getstartuptnfok						39	SetFilePointer(local_34,local_4c,(PLONG)0x0,0);
🕀 🖌 GetTempFileNaneW	- i+		LAB_00401142	XREF(1):	00401130(;	- 40	<pre>makeTempfile(param_1, local_258); pw/arl = _womdup(local_258);</pre>
- Po pPathName - Po pPrefixString		00401142 57	POSH	EDI		16	param 1[2] = (int)pWarl;
- P utrigue		00401143 57	POSH	EDI			local 30 = CreateFileW(local 258,0x40000000,1,(LPSECURITY ATTRI
Fo pTempFileName		00401144 ff 75 b8	PUSH	dword ptr [ESP + Stack(-0x4c])		44	local 2c = 0x1000r
E 1 GetTempFathill		00401147 ff 75 fc 0040114a ff 15 14	PUSH	dword ptr [ESP + Stack[-1x8]]		45	local_24 = 4LAB_00401065;
lter: 6		0040114a ff 15 14 40 40 00	CALL	dword ptr [->HIBNEL32.DLL::SetFilePointer]		44	hObject_00 = local_30;
		00401150 8d 85 ac	LEA	EAX->Stack[-0x258],[EBP + 0xfffffdac]		47	hFile = local 34;
Data Type Manager - X		14 ff ff	100	Enterprise [-origo]/[EEL + originated]		49	(**(code **)(*param_1 + 4))(slocal_34);
		00401156 50	P038	EXX		4.9	SetFilePointer(hFile, local_40, (PLOND) 0x0,0);
		00401157 tb ce	M077	parem 1,ESI		50	makeTempFile(param_1,local_258);
Cata Types .	÷	00401159 e8 49 ff	CALL	makeTempFile		51	<pre>pwVar1 = _wcsdup(local_250);</pre>
<ul> <li>BuildnTypes</li> <li>ØArdamaxKeylogger_E33AF9E602C887AC363</li> </ul>		tt tt				52	param_1[3] = (int)ps//ari;
<ul> <li>B GAY damaxxey/ogger_E33AF4E6020887AC363</li> <li>Windows_vs12_32</li> </ul>		0040115e 8d 85 ac	LEA	EAX=>Stack[-0x250],[EEP + 0xfffffdac]		· · · · · · · · · · · · · · · · · · ·	hObject = CreateFileW(local_250,6x40000000,1,(LPSECURITY_ATTRIE
B windows_vs12_52	1 ÷ 1	<b>FA FF FF</b>				v 54	local_30 = hObject;
		<			,		
	Console - Scrip	ang					8



# 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**

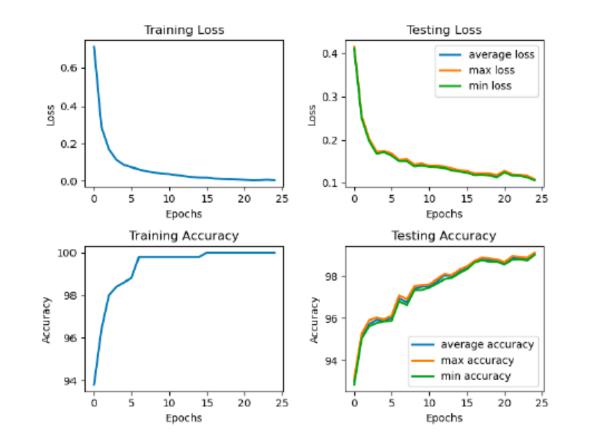
- In recent years, transformer-based natural language processing models such as BERT have surpassed humans in a number of challenging tasks
  - For example, see the Squad leaderboard at https://rajpurkar.github.io/SQuAD-explorer/
  - It is highly desirable to apply these techniques to binary program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

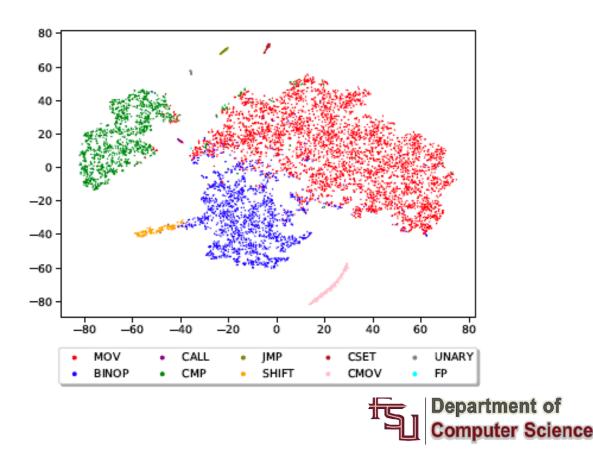
Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
<b>1</b> Jun 04, 2021	IE-Net (ensemble) RICOH_SRCB_DML	90.939	93.214
<b>2</b> Feb 21, 2021	<b>FPNet (ensemble)</b> Ant Service Intelligence Team	90.871	93.183
<b>3</b> May 16, 2021	IE-NetV2 (ensemble) RICOH_SRCB_DML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



### 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





# NG – CONT. tokens, instructions,

# he embeddings everse engineering

### 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