INDEMAAS: AN AI-ENABLED KNOWLEDGE GUIDED FRAMEWORK FOR REALIZING IN-DEPTH MALWARE ANALYSIS AT SCALE

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

Tathagata Mukherjee
Department of Computer Science
The University of Alabama
Huntsville, AL

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
- Furthermore, we as the citizen of modern societies use and rely on services that run by servers and applications
  - Smart devices are ubiquitous and are used constantly to connect to all kinds of services
- Unfortunately, they create unprecedented opportunities for malicious actors

INTRODUCTION - BACKGROUND

- Perhaps the best way to measure the vulnerabilities in programs is the Common Vulnerabilities and Exposures (CVE) list
  - The list contains 202461 (on May 13, 2023)

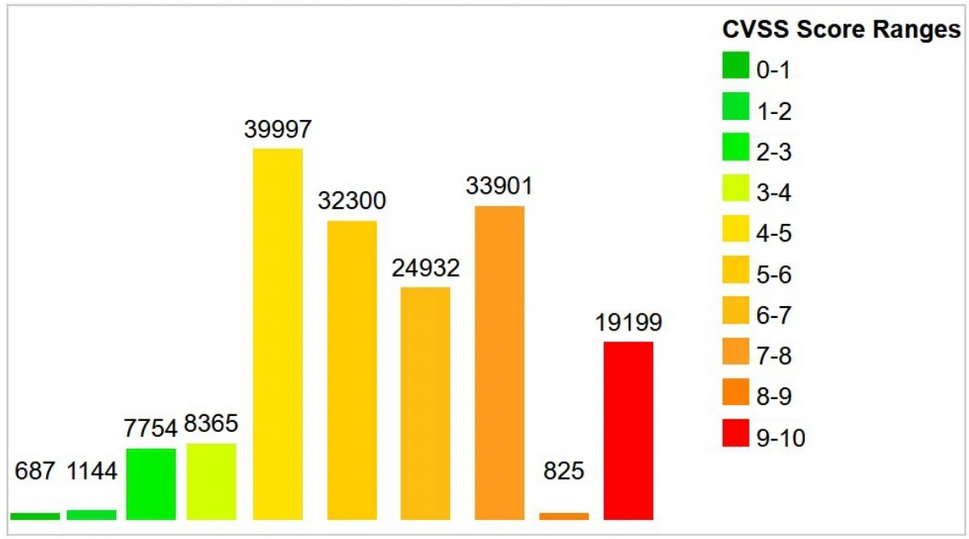
Current CVSS Score Distribution For All Vulnerabilities

Distribution of all vulnerabilities by CVSS Scores

CVSS Score	Number Of Vulnerabilities	Percentage
0-1	687	0.40
1-2	1144	0.70
2-3	7754	4.60
3-4	8365	4.90
4-5	39997	23.70
5-6	32300	19.10
6-7	24932	14.70
7-8	33901	20.00
8-9	825	0.50
9-10	19199	11.40
Total	169104	

Weighted Average CVSS Score: **6.5**

Vulnerability Distribution By CVSS Scores



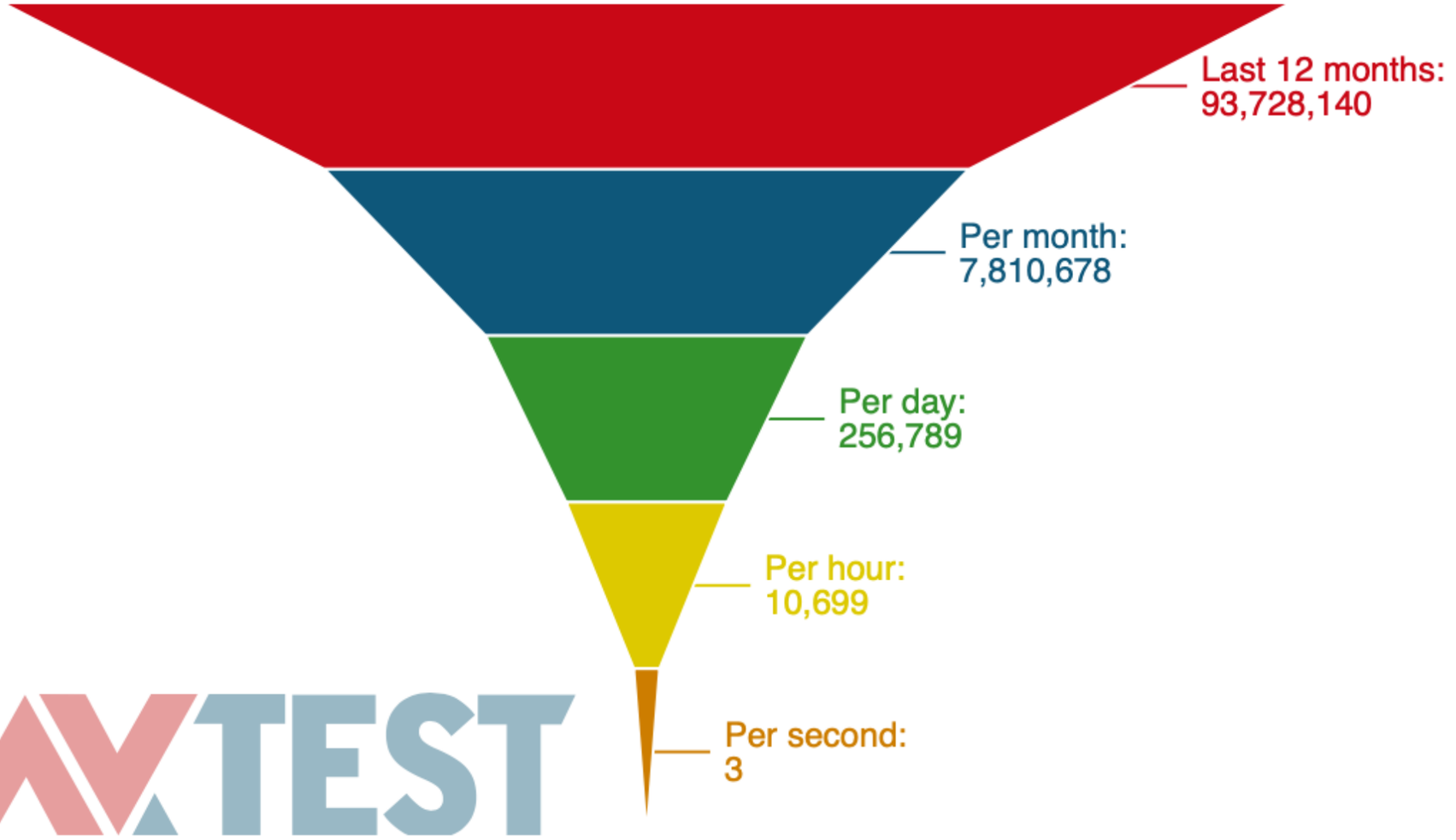
INTRODUCTION - BACKGROUND

- A hope to handle such large numbers of vulnerabilities is that the different types of vulnerabilities is much much smaller
  - The best source is the Common Weakness Enumeration (CWE), which has 933 of weaknesses

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

INTRODUCTION - BACKGROUND

- Malicious programs and applications is a serious issue at the global scale



INTRODUCTION - BACKGROUND

- There is a big market for security products

#2. The worldwide cybersecurity market is estimated to reach up to $300 billion by 2026

The value of the cybersecurity market is growing rapidly. Experts predict that it can easily grow by two or threefold in the next three years as the demand for cybersecurity services and solutions is growing.

  - We also install and use them on our computers and devices

- A lot was spent to deal with cybercrime and the cost will continue to increase

#1. Global expenditure on cybersecurity reached $6 trillion in 2021

According to Cybersecurity Ventures, Cybercrime is expected to cost the entire world $10.5 Trillion annually by 2025. This is up from $3 Trillion a decade ago, and $6 Trillion in 2021. It will be vital that cybersecurity spending continues to grow in proportion to this increase.

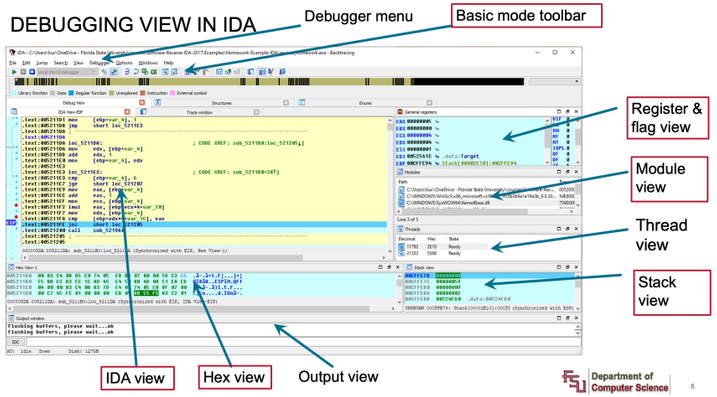
- Why are there so many malicious programs and applications still?
  - Could we cope the malicious programs more effectively?

CURRENT SITUATION AND CHALLENGES

- Most of the malware samples are analyzed individually
  - The relationships via shared functions are not explored effectively
- Three commonly used approaches, static analysis, dynamic analysis, symbolic execution, and combinations of them are not scalable
  - Static analysis methods are not precise, labor-intensive, and not scalable
  - Dynamic analysis methods are precise, can be done automatically, but not scalable with limited coverage often
  - Symbolic execution techniques can handle small programs only due to the state explosion problem
    - For example, a for statement such as “for (i=0; i<1000; i++) sum+=i;” can have 2^{1000} different execution branches

CURRENT SITUATION AND CHALLENGES – 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



CURRENT SITUATION AND CHALLENGES – CONT.

- However, malware analysis 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 large language models such as chatGPT and burpGPT have been to be very helpful
  - Especially through the API provided by openAI, some of the tasks can be done more quickly and there are tools available for some purposes as well

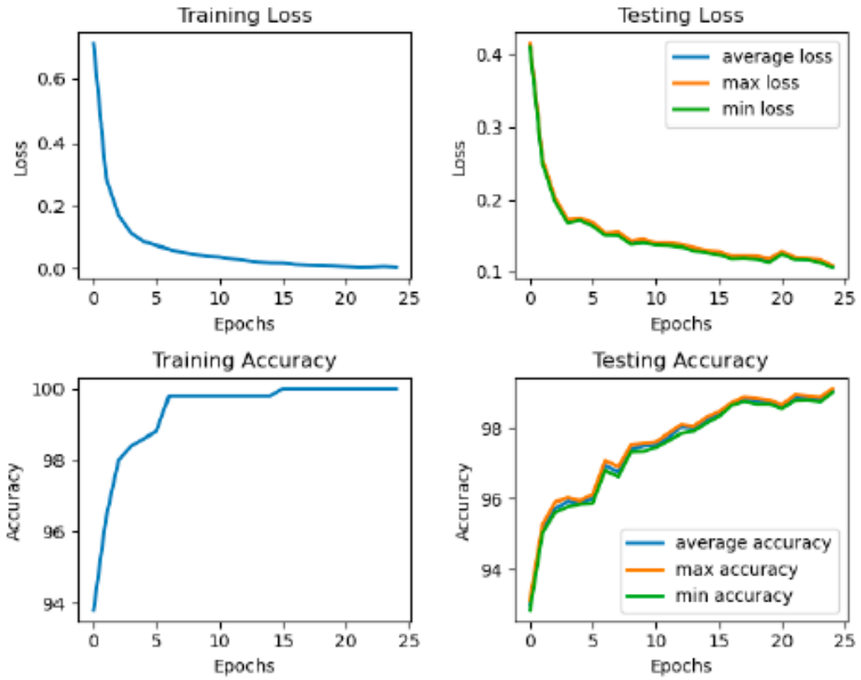
```python
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")

completion = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": "Tell the world about the ChatGPT API in the s
  ]
)

print(completion.choices[0].message.content)
```

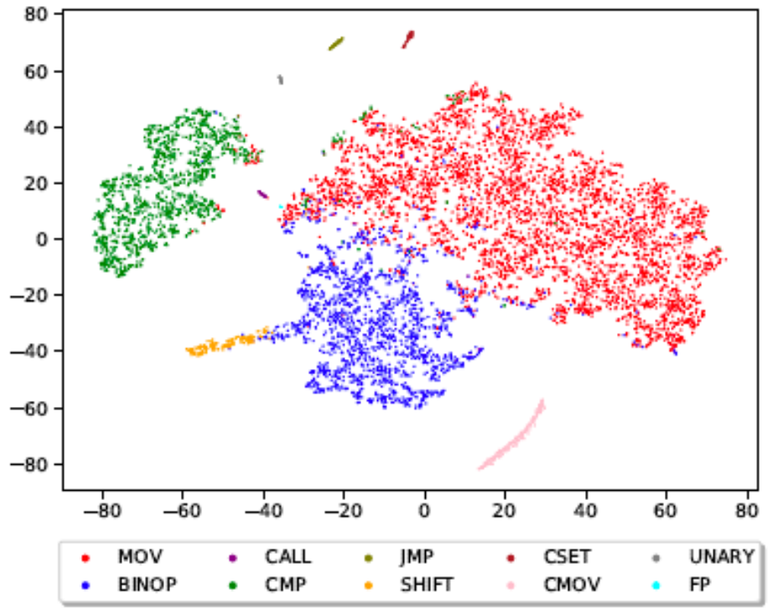
Like many modern tools, ChatGPT has an API that allows third-party applications to query the AI and receive replies using a script instead of the online user interface. Some individuals have already used this API to create impressive open-source analysis tools that can make cybersecurity researcher's jobs a lot easier.

Notable examples of such tools are Gepetto and GPT-WPRE, which add meaningful comments to code decompiled using IDA and Ghidra, respectively. Another helpful tool is IATelligence, a script that extracts the content of the Import Address Table (IAT) from PE files and adds information about MITRE ATT&CK techniques that can be implemented using the imported libraries. These are mere examples of the potential ChatGPT has as an analysis tool, and with some thought and effort this AI might even be able to integrate into SIEM systems and do much of the work currently done by tier-1 analysts.

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

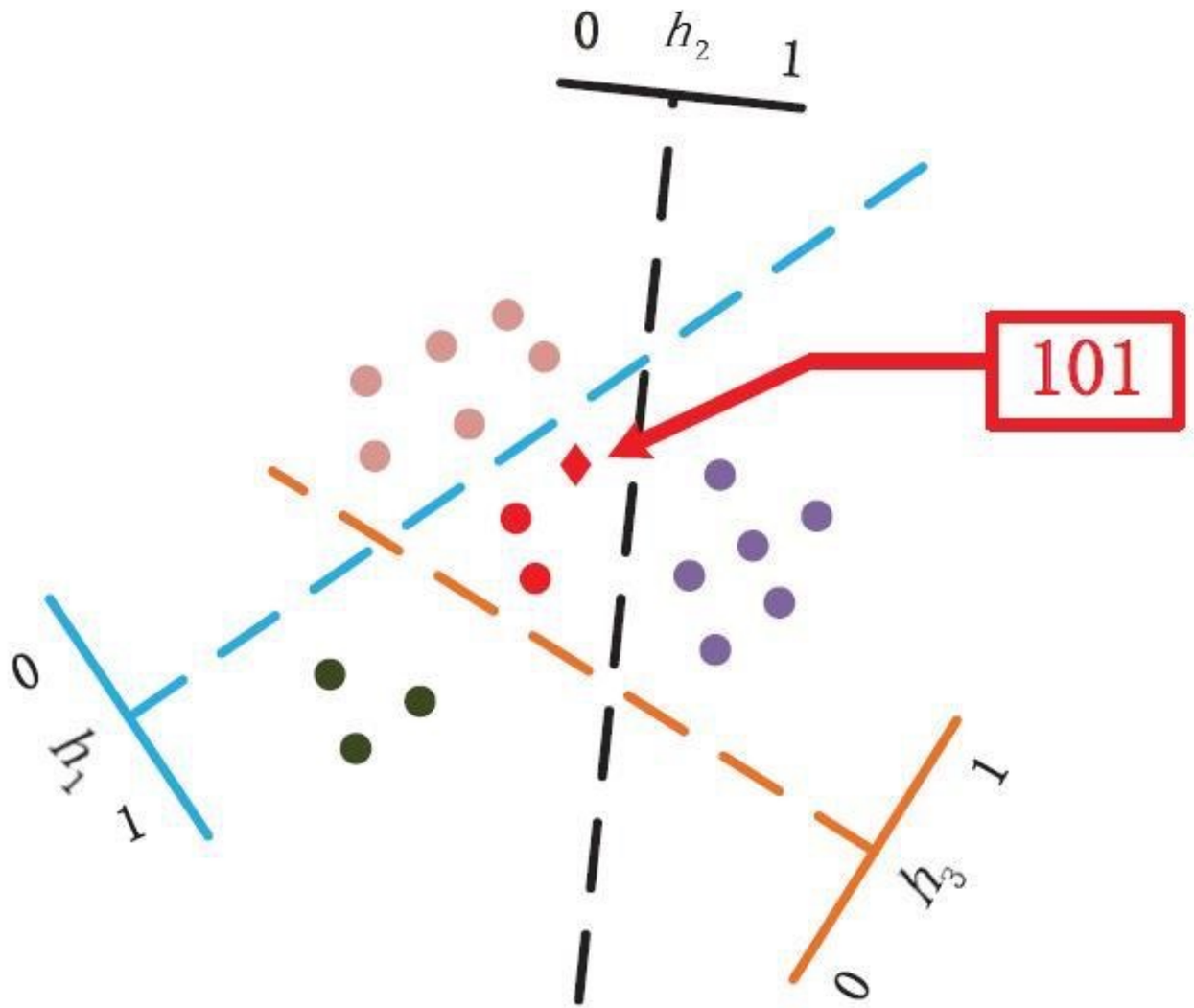
- As analysts and programmers understand programs using meaningful variable and function names much better, it seems a good idea to utilize the available source code to learn patterns so that lost semantics in stripped binaries can be recovered
  - With the recent techniques of graph neural networks 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

SCALABLE IN-DEPTH MALWARE ANALYSIS

- Clearly in-depth malware analysis is time consuming
  - The system has to understand the functions and how the programs work
- However, as malware samples reuse and share functions with other samples, the key to achieve scalability is to reuse the analyses effectively
  - In our proposed system, it will be done using a knowledge base
  - To find similar functions in the database, we use locality sensitive hashing
  - We are developing a system, called INDEMAAS, in-depth malware analysis at scale

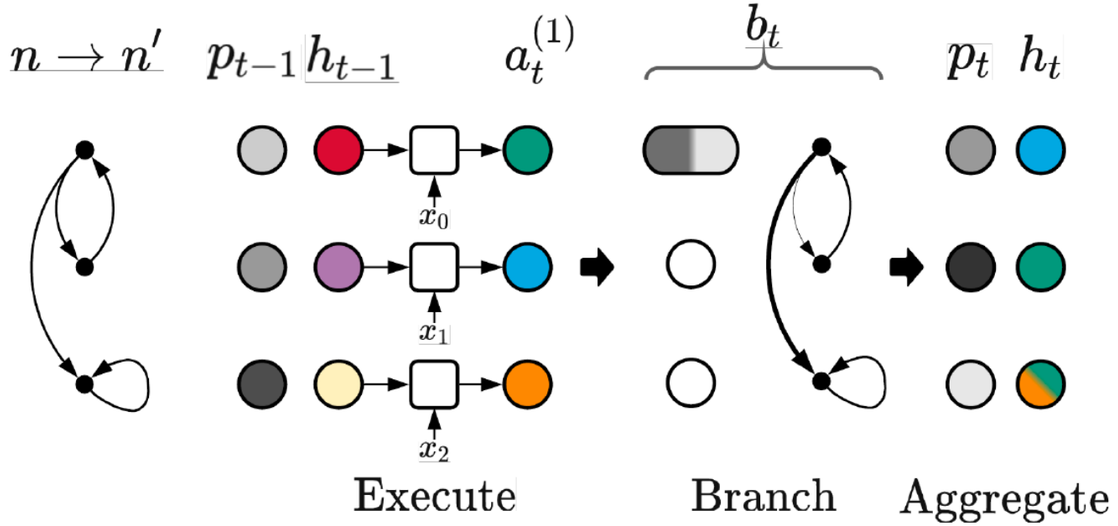
LOCALITY-SENSITIVE HASHING

- Instead of searching for similar functions in the semantic embedding space, we use multiple linear projections to produce binary patterns that can be used for finding similar functions very quickly
  - In low dimensional spaces, we also use hierarchical lookup tables for fast mapping to classes



SCALABLE IN-DEPTH MALWARE ANALYSIS – CONT.

- We use a deep learning model to learn to summarize functions
  - In a way, very similar to what analysts do, the model summarizes the functions in a higher level so that the complexities of the programs can be handled effectively
    - The key advantage is that we can summarize the functions in parallel from bottom-up
  - We use an architecture very similar to IPA-GNN [1]



[1] D. Bieber, et al., "Learning to Execute Programs with Instruction Pointer Attention Graph Neural Networks," NeurIPS, 2020.

SCALABLE IN-DEPTH MALWARE ANALYSIS – CONT.

- We use a deep learning model to learn to summarize and execute programs
  - In a way, very similar to what analysts do, the model summarizes the programs in a higher level so that the complexities of the programs can be handled effectively
  - For each function, we also have the different cases using simulations

```
int if_else(int a, int b, int c) {
    int result;
    if (a > b) {
        result = a + b;
    }
    else {
        result = c - a;
    }
    return result;
}
```

```
int switch_large(int a, int b, int c, int d) {
    int result = 0;
    switch (a) {
    case 1:
        result = b;
        break;
    case 2:
        result = c;
        break;
    case 3:
        result = d;
        break;
    case 4:
        result = b - c;
        break;
    case 5:
        result = c - d;
        break;
    case 6:
        result = d - b;
        break;
    case 7:
        result = b + c;
        break;
    case 8:
        result = c + d;
        break;
    case 9:
        result = d + b;
        break;
    case 10:
        result = b * c;
        break;
    case 11:
        result = c * d;
        break;
    case 12:
        result = d * b;
        break;
    }
        return result;
}
```

SCALABLE IN-DEPTH MALWARE ANALYSIS – CONT.

- In INDEMAAS, each function will have a semantic representation learned by the deep learning model
  - For each function, we also have the different cases using simulations
    - Since there are 933 common weaknesses, we map each function to these weaknesses as Intezer does but using the MITRE ATT&CK patterns

MITRE ATT&CK	Technique	Severity	Details
Execution::Exploitation for Client Execution [T1203]	The EQNEDT32 equation process created a child process likely indicativ...	High	created_process:C:\Users\Public\vbc.exe
Execution::Exploitation for Client Execution [T1203]	The EQNEDT32 process established a network connection downloading...	High	ip address:103.140.250.22,url:http://103.140.250.22/Explorer/vbc.exe,fi...
-	A process created a hidden window	Medium	Process:vbc.exe -> C:\Users\Public\vbc.exe
-	The office file contains a macro	Medium	-
-	Attempts to connect to a dead IP:Port (1 unique times)	Low	IP:103.140.250.22:80 (Vietnam)

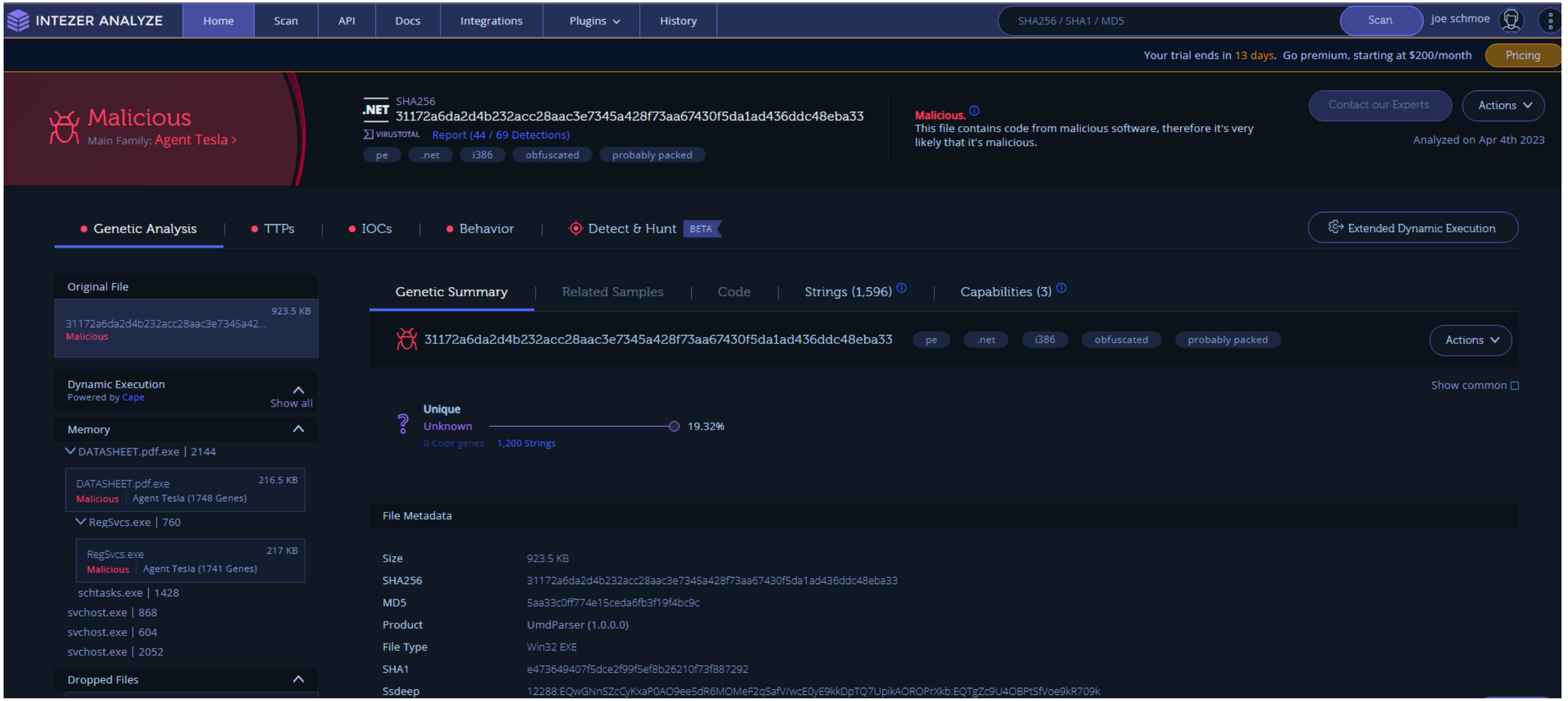
SCALABLE IN-DEPTH MALWARE ANALYSIS – CONT.

- We use a deep learning model to learn to summarize and execute programs
  - In a way, very similar to what analysts do, the model summarizes the programs in a higher level so that the complexities of the programs can be handled effectively
  - For each function, we also have the different cases using simulations

```
int while_loop(int a, int b) {
    int result = 0;
    while (a < b) {
        b--;
        a++;
        result++;
    }
    return result;
}
```

INTEGRATING WITH INTEZER

- We are in the process of incorporating our ideas into the Intezer platform



SUMMARY

- Achieving in-depth malware analysis at scale is very challenging
  - Based on the key observation that malware samples reuse functions, we build an AI-approach to overcome the challenges
    - We use deep learning models to learn a semantic representation for each of the functions
      - We build a knowledge for relationships among different functions
      - To index a huge base, we use locality-sensitive hashing
      - We train a deep learning model to learn how to summarize functions at a higher level to cope with complexities and the state explosion problem associated with symbolic execution
    - Integrating all the components leads an efficient and effective solution for the problem