MANDIANT

Annotating Malware Disassembly Functions Using Neural Machine Translation

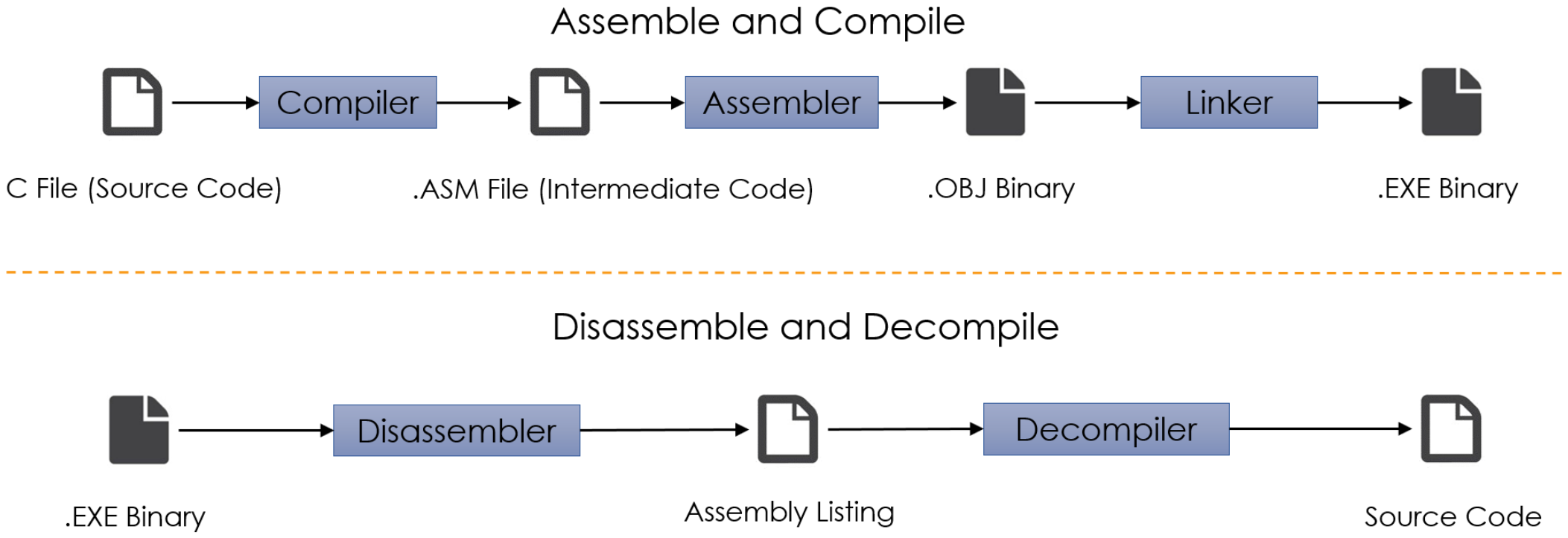
Conference on Applied Machine Learning for Information Security (CAMLIS) ‘21

Sunil Vasisht
Staff Data Scientist

Phil Tully, PhD
Manager, Data Science

Jay Gibble
Distinguished Research Engineer

Static Malware Analysis with Disassembly



Basic Static Techniques

- Initial triage/assessment

Basic Dynamic Techniques

- Also has shortcomings

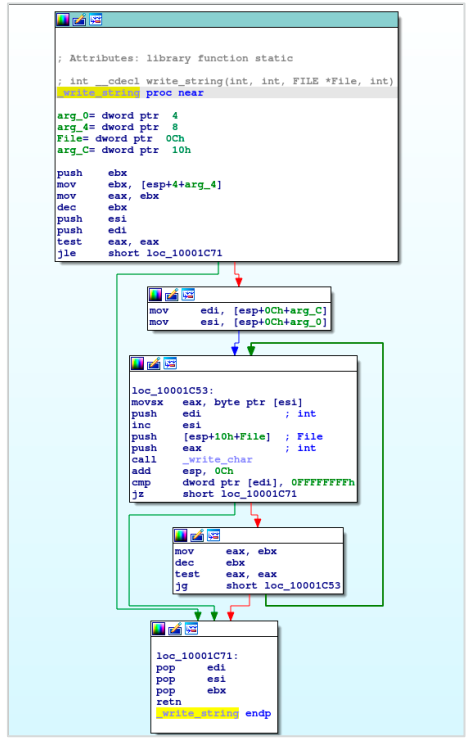
Disassembly/Decompilation

- Step closer to source

Interpreting Disassembly and Decompilation

Disassembly Control flow graph (CFG) view

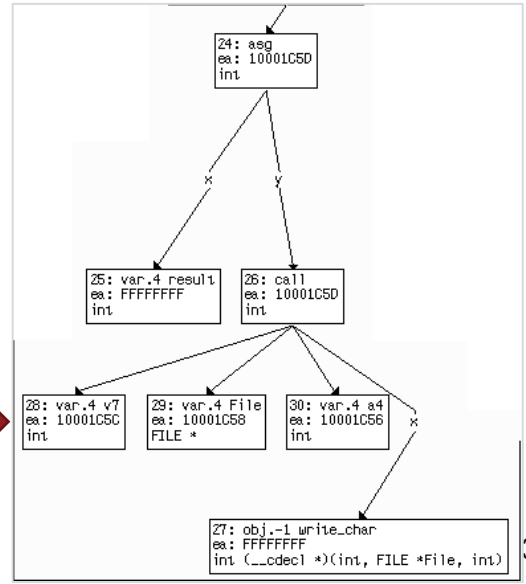
- Illustrates code execution path as a directed graph
  - ”Basic blocks" of sequential instructions, execution flow is top to bottom, up arrows = loops
  - Branches are shown as red/green pairs of outbound edges at bottom of block
- Data flow is challenging, keep track of register values and memory assignments



A decompiler renders functions in a high-level language such as C

- Compact, easier to understand representation
- Caveats: Unstable, lossy
- Abstract syntax tree (AST)

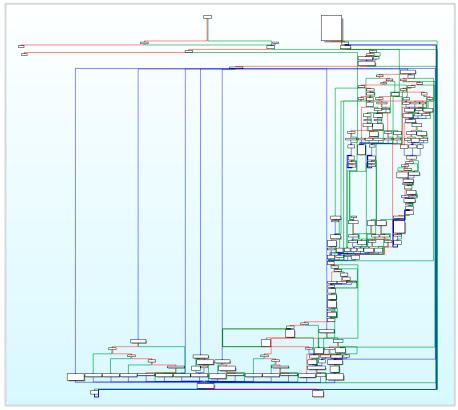
```
1  int __cdecl write_string(int a1, int a2, FILE *File, int a4)
2  {
3    int result; // eax
4    int i; // ebx
5    int v7; // eax
6
7    result = a2;
8    for ( i = a2 - 1; result > 0; result = i-- )
9    {
10     v7 = *(char *)a1++;
11     result = write_char(v7, File, a4);
12     if ( *(_DWORD *)a4 == -1 )
13       break;
14   }
15   return result;
16 }
```

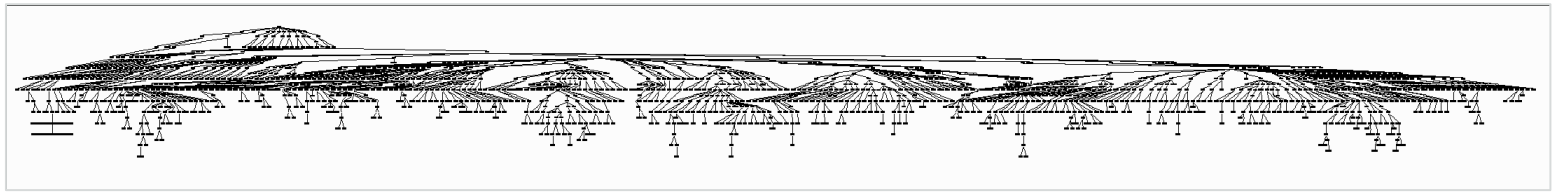


Levels of Abstraction and Reverse Engineering

Fast Library Identification and Recognition Technology (FLIRT) - sig database for function names

- Reversers can rename, IDA propagates signature changes, only need to change in one location
  - Helpful for situations like code reuse by malware authors, speeds up reversing process
- Custom signature development and persistence of .idb changes across analysts and samples
  - Fast Library Acquisition for Identification and Recognition (FLAIR), also FIRST, CrowdRE, collabREate, PSIDA, etc.





CFG and AST for *_output()* library function, a utility function used by the *printf()* family

- Real world, high complexity example not recognized by IDA Pro's FLIRT signatures
- Not relevant for most analyses (Don't want to spend time reverse engineering this function)

Reversing Workflow Pain Points

Instructions for an entire disassembled program can number in the thousands or even millions

- Even expert-level reversers can spend hours poring over disassembly to piece together code functionality for more complex malware samples

FLIRT signatures typically only account for library code added by a compiler (malware: 5-10%)

- Furthermore, while function names added previously by human analysts are highly precise, their correspondingly low recall means that most functions within a malware sample freshly pulled up within IDA tend to lack semantically meaningful names
- Manually-labeled function names vary significantly across human analysts

Reverse Engineering is an extremely difficult skill to master

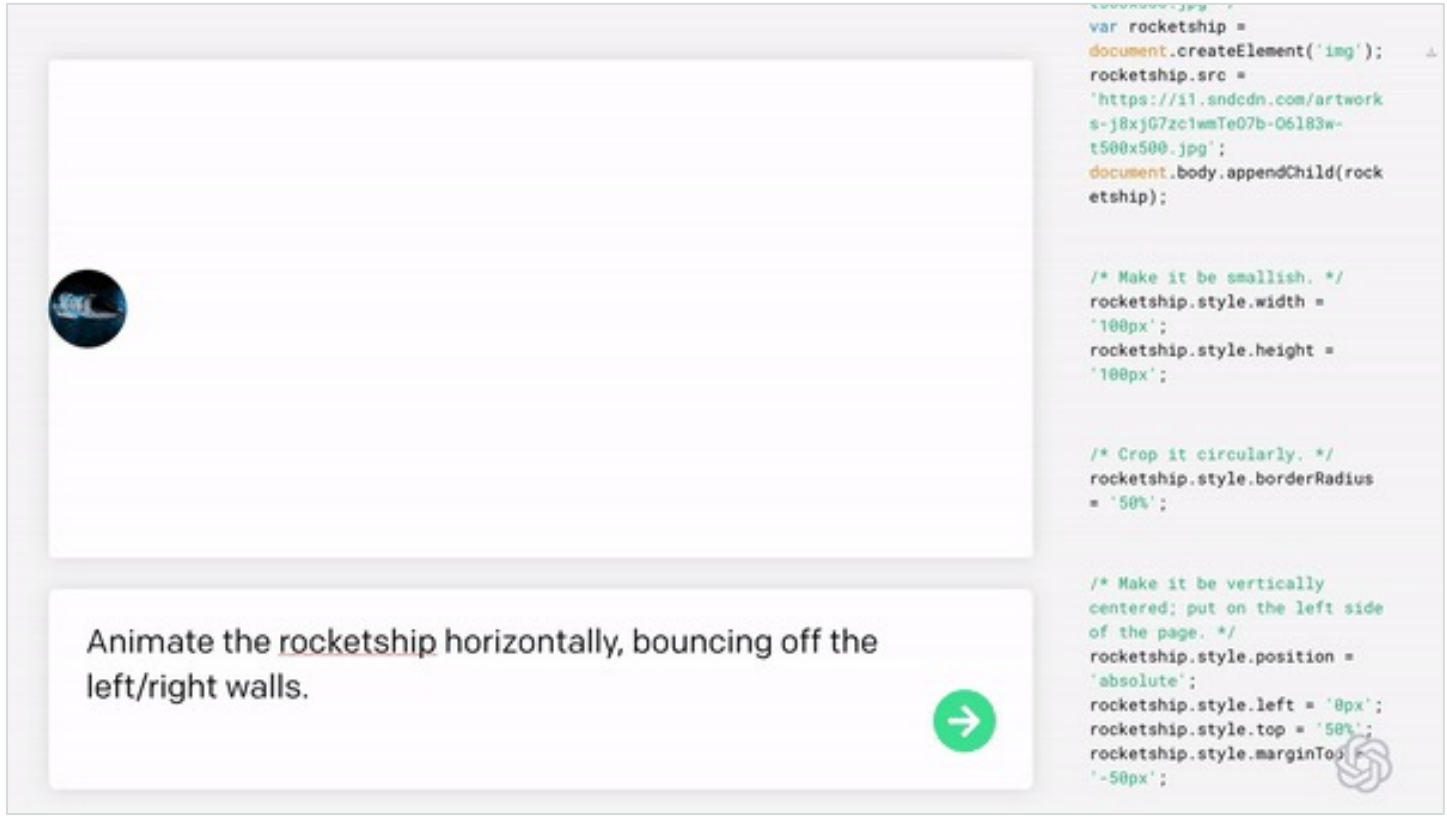
- Throwing more analyst bodies/worker-hours at the problem is not scalable

How can we increase function name coverage within binary disassembly in order to accelerate malware triage?

PROBLEM STATEMENT

Generating Natural Language from Source Code (& Vice-Versa)

- NMT Approaches
  - Seq2Seq, LLMs
- Use Cases
  - Code summarization
  - Code documentation
  - Inferring variable names and types
  - Bug detection
  - Auto-completion of code (e.g. Codex)
- Higher-Level Programming Languages
  - shorter in length
  - easier to read
  - more linearly ordered
  - syntactically richer



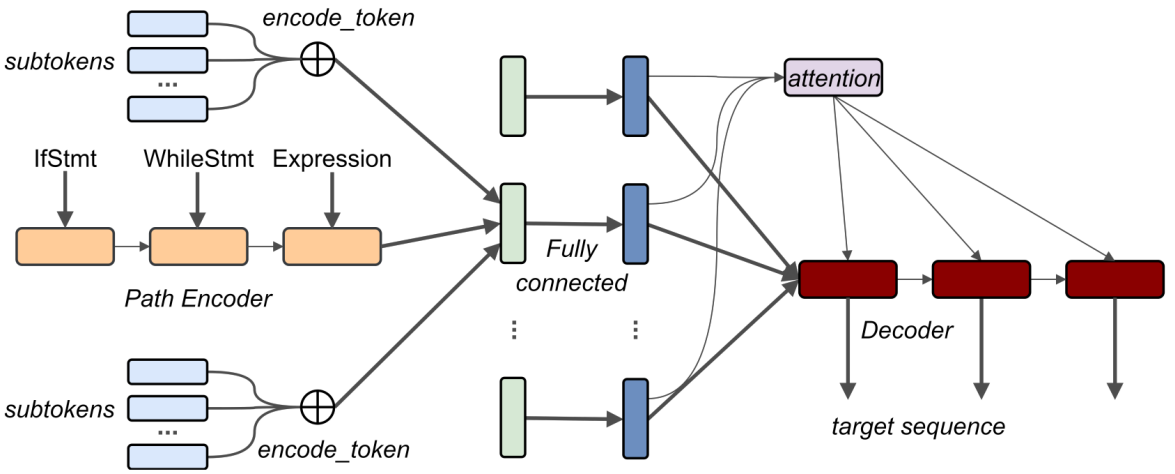
https://openai.com/blog/openai-codex/ | https://copilot.github.com/

Chen, Mark, *et al.* "Evaluating large language models trained on code." arXiv preprint arXiv:2107.03374 (2021).

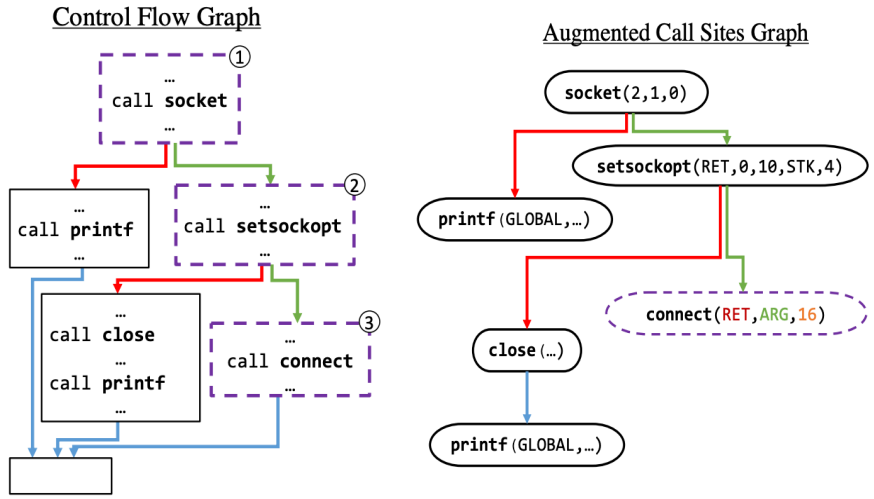
Previous Work: code2seq and Nero

MODEL	DATASET	PRECISION	RECALL	F1
code2seq	Java-large	64.03	55.02	59.19
Nero - GCN	GNU ELF	48.61	42.82	45.53

- code2seq (https://code2seq.org/)
  - Input Representation: Decompiler AST
    - AST leaves encode user-defined identifiers and names
    - Internal AST nodes encode structures like loops, expressions, and variable declarations
  - Model Architecture: seq2seq w/ Attention
    - Random AST paths encoded as vectors using a BiLSTM
    - Concatenate path embeddings w/ leaf token embeddings
    - Attend to relevant AST paths during decoding



- Nero (https://github.com/tech-srl/Nero/)
  - Input Representation: Disassembly CFG
    - Nodes = basic blocks, Edges = control flow instructions
  - Model Architecture: GCN
    - Augmented API call sites (+args), pointer-aware slicing
    - Sequences of encoded call site subtoken embeddings
    - LSTM decoder attends to node representations



Alon, Brody, Levy, and Yahav. "code2seq: Generating sequences from structured representations of code." ICML (2019).
David, Alon, and Yahav. "Neural reverse engineering of stripped binaries using augmented control flow graphs." Proc. ACM Prog. Lang. 4.OOPSLA (2020): 1-28.

KEY HYPOTHESIS

This will work for PE Malware Disassembly

ASTs and CFGs should normalize differences at the source code level so that commonalities emerge between functions, despite variation within individual variables/control structures

Malware Disassembly Dataset

3.2m annotated functions from 4.6k malicious PEs

- $\sim\frac{1}{3}$ auto-generated IDA function names from FLIRT
- $\sim\frac{2}{3}$ a decade's worth of Mandiant reverser annotations

AST de-duplication

- Remove duplicated (md5, func va) pairs, remove identical ASTs, duplicate ASTs ignoring leaf tokens
- Resulting training dataset
  - 420k named functions from 4k binaries

fn_name	fn_name_processed	fn_tokens
RtlDecodePointer_4	rtl decode ptr	[rtl, decode, ptr]
QObject::Sock5Server::constructor_1	qobject sock server constructor	[qobject, sock, server, constructor]
_handle_errorf	handle errorf	[handle, errorf]
runtime.flushmcache	runtime flush mcache	[runtime, flush, mcache]
??1AFX_AUTOHIDE_DOCKSITE_SAVE_INFO@@QAE@XZ	afx autohide docksite save info	[afx, autohide, docksite, save, info]
_wwincmdln	wwincmdln	[wwincmdln]
@Graphics@TBitmap@GetMonochrome$qqrv	graphics tbitmap get monochrome	[graphics, tbitmap, get, monochrome]
_CallSETranslator	call se translator	[call, se, translator]
terminate	terminate	[terminate]
ASN1_STRING_dup_64_openssl_1.1.0f_vs2008_x64	asn str dup 64 openssl vs2008 x64	[asn, str, dup, 64, openssl, vs2008, x64]

Label token consolidation

- Tokenization (punctuation, snake_case, camelCase)
- Lemmatization, synonym & acronym replacement
- Single char edit merging, high entropy noise removal

Encoding Malware Function Decompilation ASTs

Sampled AST sections

- Label tokens
- Leaf value tokens
- Node path
- Type tokens (tried with and without)

code2seq parameters

- cross entropy loss using AdamW optimizer
- Output label max sub tokens: 7
- Max AST contexts: 200
- Max path length: 11, max leaf sub tokens: 5
- Vocabulary sizes:
  - Labels: 16K, Node Types: 83, Leaf Tokens: 43K
- Encoder embedding size: 128; Decoder size: 320

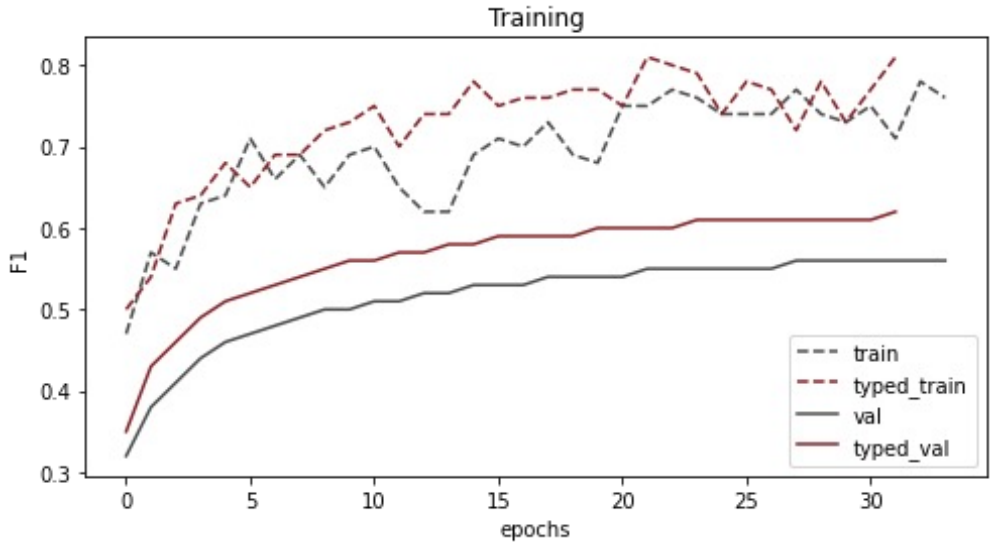
remote|shell|exec struct|spc|startupinfoa,startup|info,var|ref|call|expr|block|expr|call|var,string,char|spc|*
bool|spc,create|process|a,obj|call|lor|if|block|if|block|return|var,localv,int|obj,close|handle,bool|spc

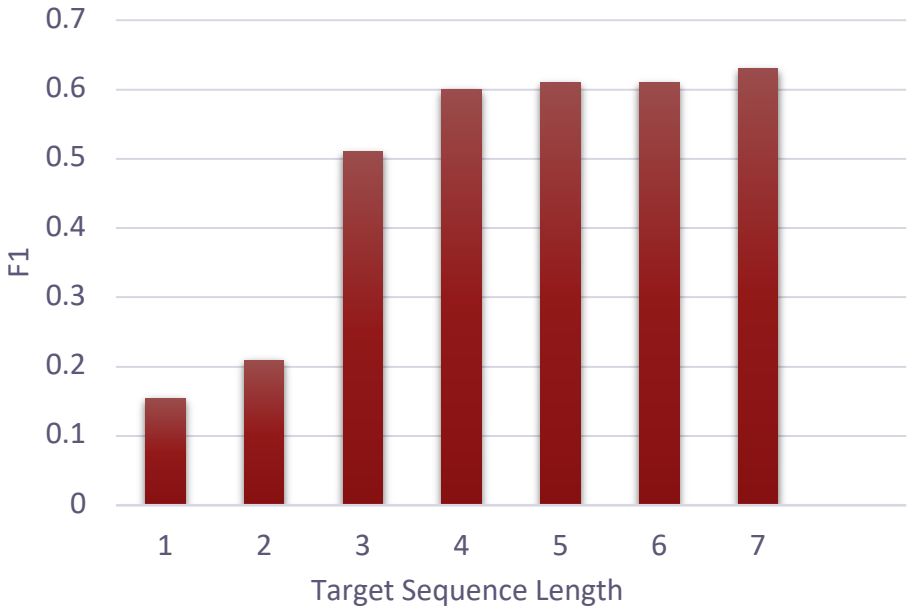
```
['64: block\\nea: 00401CCA',
   ['65: expr\\nea: 00401CCA',
    ['66: call\\nea: 00401CCA\\nvoid *',
     ['67: helper memset\\nea: FFFFFFFF\\nvoid *(__fastcall *)(void *, int, '
      'int)'],
     ['68: ref\\nea: 00401CCA\\nstruct _STARTUPINFOA *',
      ['69: var.-1 StartupInfo\\nea: 00401CCA\\nstruct _STARTUPINFOA']],
     ['70: num 0\\nea: 00401CCA\\nint'],
     ['71: sizeof\\nea: 00401CCA\\nint',
      ['72: var.-1 StartupInfo\\nea: FFFFFFFF\\nstruct _STARTUPINFOA']]]],
   ['73: expr\\nea: 00401CD9',
    ['74: call\\nea: 00401CD9\\nvoid',
     ['75: obj.-1 GetStartupInfoA\\nea: FFFFFFFF\\nvoid (__stdcall '
      '*)(LPSTARTUPINFOA lpStartupInfo)'],
     ['76: ref\\nea: 00401CD8\\nstruct _STARTUPINFOA *',
      ['77: var.-1 StartupInfo\\nea: 00401CD8\\nstruct _STARTUPINFOA']]]]
   ['78: expr\\nea: 00401CE2',
    ['79: asg\\nea: 00401CE2\\nDWORD',
     ['80: memref (m=0)\\nea: FFFFFFFF\\nDWORD',
      ['81: var.4 StartupInfo\\nea: FFFFFFFF\\nstruct _STARTUPINFOA']],
     ['82: num 68\\nea: 00401CE2\\nint']]],
 ........
    ['108: expr\\nea: 00401D0A',
    ['109: call\\nea: 00401D0A\\nchar *',
     ['110: helper strcpy\\nea: FFFFFFFF\\nchar *(__fastcall *)(char *, '
      'const char *)'],
     ['111: var.-1 String2\\nea: 00401D0A\\nCHAR *']
     ['112: string \\"WinSta0\\\\\\\\\\Default\\"\\nea: 00401D0A\\nchar[17]']]],
```

Decompilation AST Results

- Enhance ASTs with identifier types by embedding them into numerical vectors and concatenating them with vectors for both tokens and AST paths
- F1 - learned correlations between label tokens
- Noticed similar improvements when embedding types into AST code2seq-based model:

DATASET	MODEL	PRECISION	RECALL	F1	LOSS
Java-med	Original	**51.38**	39.08	44.39	-
Java-med	Type-enhanced	49.15	**42.35**	**45.5**	-
PE Test (Ours)	Original	55.68	55.70	55.69	28.99
PE Test (Ours)	Type-enhanced	**62.12**	**61.10**	**61.60**	**26.38**





https://github.com/JetBrains-Research/code2seq

Spirin, Bogomolov, Kovalenko, and Bryskin. "PSIMiner: A Tool for Mining Rich Abstract Syntax Trees from Code." 18th IEEE/ACM International Conference on Mining Software Repositories, MSR (2021): 13-17. arXiv:2103.12778.

Qualitative Evaluation

- Bakeoff on a PMA sample, analyst has no knowledge of predictions
- Observations: analyst variability, selective naming (5-20 per sample)
- More near hits than misses. Envisioning a “guided” analyst setting

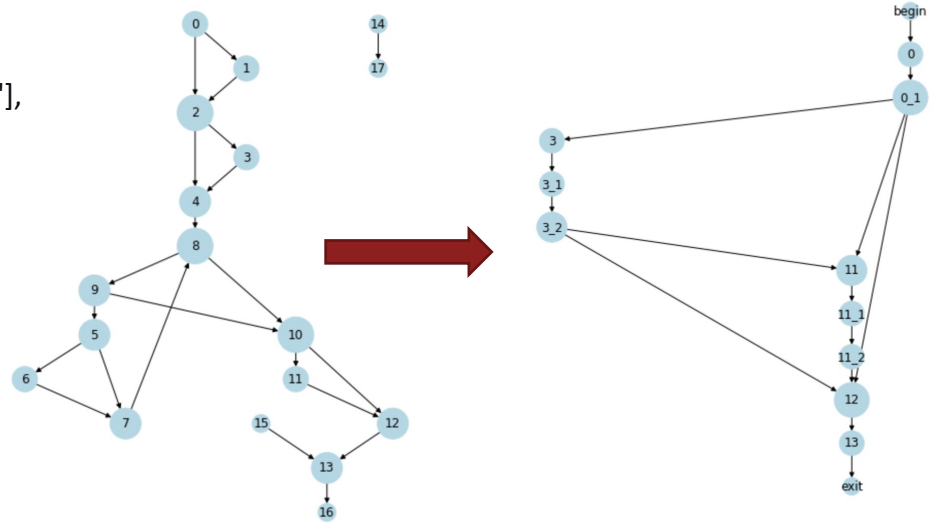
PREDICTION	ANALYST A	ANALYST B
['execute', 'self', 'delete', 'self']	_delete_myself	selfdel
['get', 'sedebug', 'priv']	has_priv	grant_priv
['get', 'string', 'name']	_parsecmd	
['get', 'string', 'by', 'hash']	_decode_arg	
[‘aes’, ‘decrypt’]		likely_rijndael_routine
['tolower']		tolower_wrapper
['get', 'reg', 'value', 'keys']		vmcheck_reg_devices
['inject', 'into', 'process']		process_replacement
['get', 'system', 'directorya']		form_path
['zb64', 'decode']	_base64_decode_probably	b64_decode
['create', 'shell', 'thread', 'shell']		RevShell
['start', 'main', 'thread']		DllMainThreadStart
['dobase64']	_base64_encode	b64encode3
['xor', 'data']	_xor_cipher	decode
['do', 'toupper', 'ctype', 'std']	_all_toupper	ucase
['get', 'system', 'directorya']	_get_system_directory	
['get', 'module', 'file', 'name']	_get_module_file_name	getModuleNameWrapper

PREDICTION	ANALYST A	ANALYST B
['get', 'http', 'post']		readFromUrl
['get', 'resource']		ExtractAndDecodeUrl
['write', 'file', 'to', 'disk']		maybe_write_screenshot?
['create', 'dib', 'window']	_generate_bitmap	CaptureScreenshot
['add', 'hash']		initEncodingConstants
['des', 'encrypt', 'openssl', 'i386', 'libeay32']	_doobf	encode
['ecp', 'nistz256', 'mul']	_obfuscate_bitmap	
['do', 'write', 'string']		call_maybe_write_scrshot
['get', 'http', 'data']	_docmd	
['get', 'next', 'sub', 'command', 'data']	_unobf	s_strcmp
['get', 'rand', 'file', 'name']		boolean_instr_func
['load', 'resource', 'by', 'name']		checkmac
['get', 'random', 'type']	_vmchk	vmcheck_in_vx
['read', 'file']	_responder	ThreadStart2
['rl', 'stream', 'load', 'from', 'file']	_encrypt_aes_maybe	cryptoRoutine
['add', 'hash']		initEncodingConstants

Encoding Malware Function Disassembly CFGs

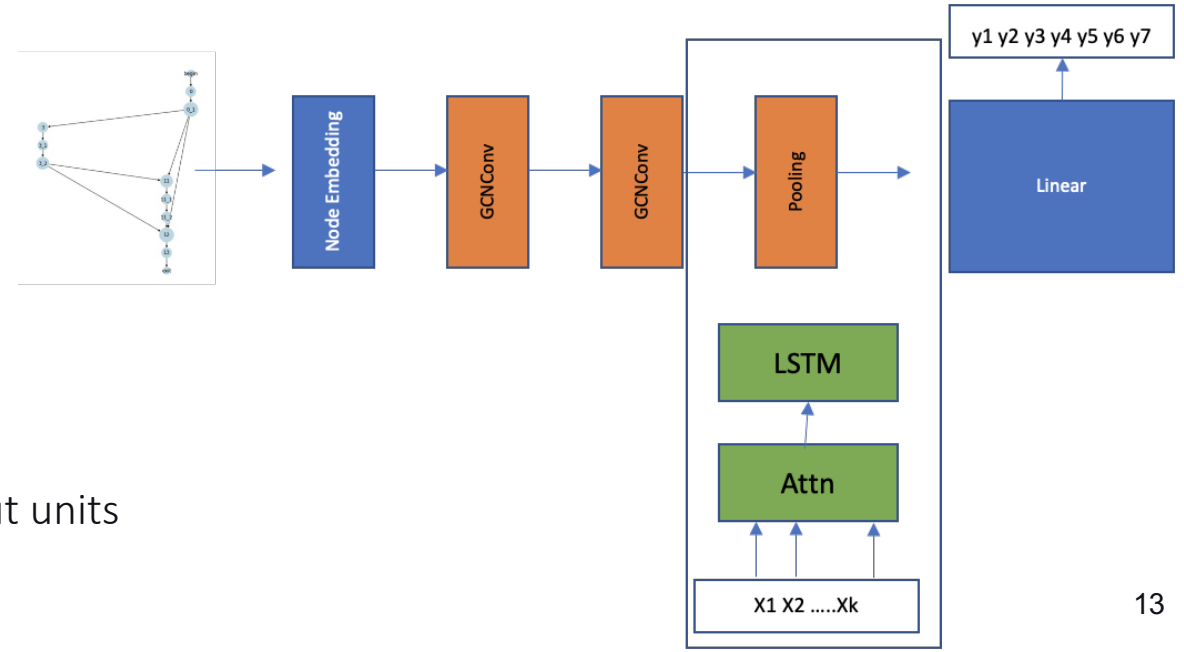
{'blk_end_va': 5390201,
'blk_id': 0,
'blk_pred': [],
'blk_start_va': 5390156,
'blk_succ': [{'blk_end_va': 5390207,
'blk_id': 1,
'blk_start_va': 5390201},
{'blk_end_va': 5390211,
'blk_id': 2,
'blk_start_va': 5390207}]},
{'blk_end_va': 5390207,
'blk_id': 1,

['31: expr\\nea: 5390191',
['32: asg\\nea: 5390191\\nunsigned int',
['33: var.4 v16\\nea: 4294967295 \\nunsigned int'],
['34: call\\nea: 5390191\\nunsigned int',
['35: helper __readfsdword\\nea: 4294967295
\\nunsigned int (__fastcall '
'*)(unsigned int)'],
['36: num 0\\nea: 5390183\\nint']]]],
['37: expr\\nea: 5390194',
['38: call\\nea: 5390194\\nvoid',
['39: helper __writefsdword\\nea: 4294967295
\\nvoid (__fastcall *)(unsigned '
'int, unsigned int)'],



Converting CFGs to call site graphs

- We map the API addresses from AST to block addresses in IDA generated CFGs and retain only the call site nodes

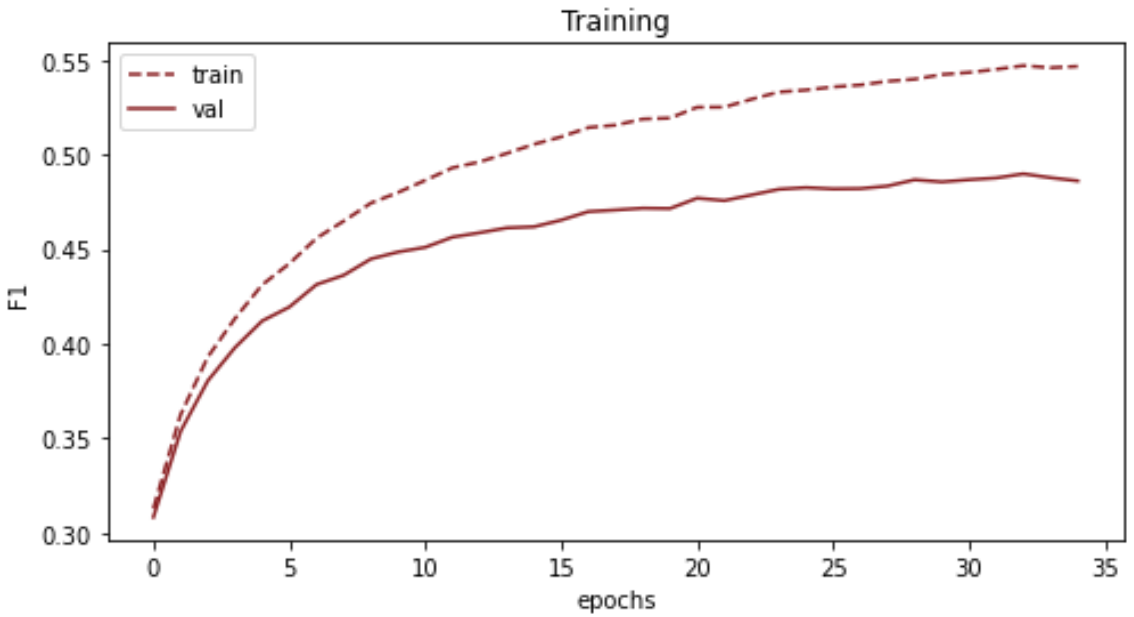


CFG2seq parameters

- Use API name and arg type subtokens (not args)
- Max API sub tokens: 50, Output Vocabulary size: 16K
- Node Embedding: 128, Set2Set Pooling: 256, 7 Linear output units

Disassembly CFG Results

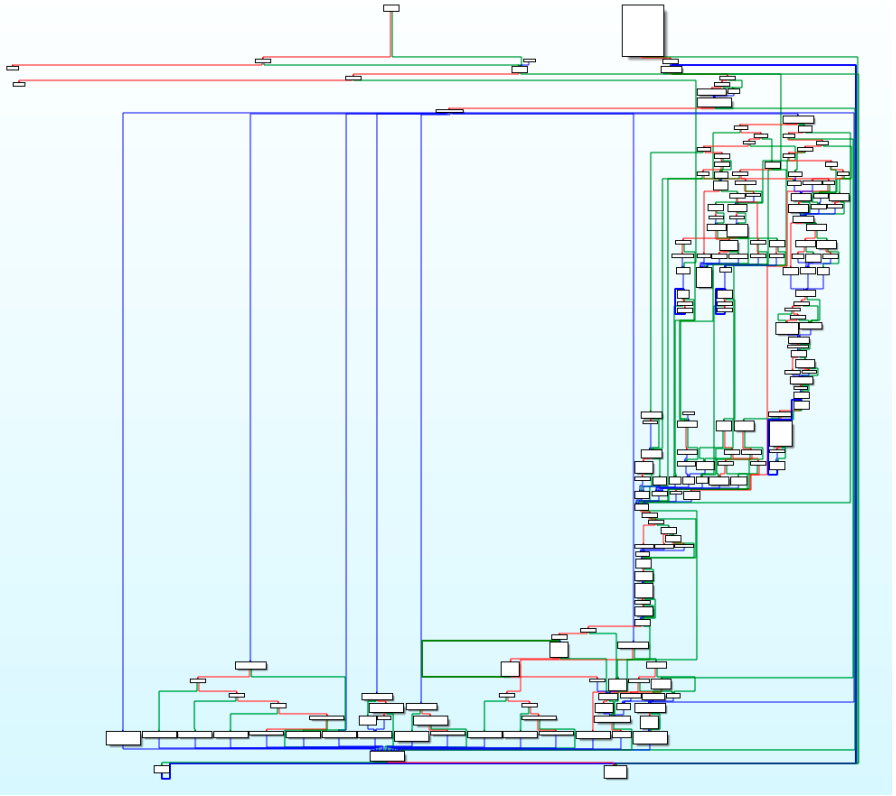
- While Nero performs binary analysis of assembly to find abstract/concrete values of API args, we are using the type information for args present in the AST
- F1 scores for the CFG model are lower than the code2seq AST-based model
- CFG based representation using only API and arg types is a more compact representation
- Future work: qualitative comparison, combined model leveraging both ASTs and CFGs



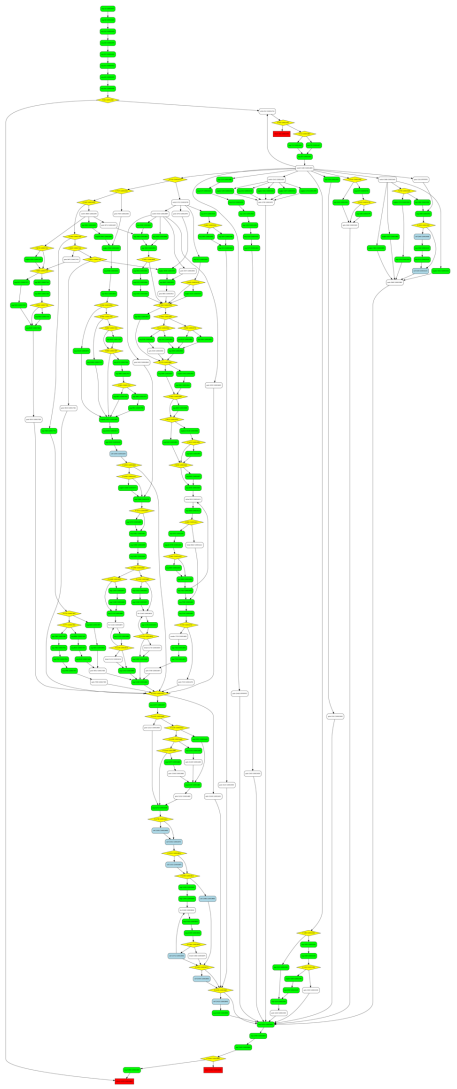
DATASET	MODEL	PRECISION	RECALL	F1
GNU ELF	Nero	48.61	42.82	45.53
PE Test (Ours)	Type-enhanced	**62.12**	**61.10**	**61.60**
PE Test (Ours)	CFG2Seq	55.07	43.49	48.60

_output()

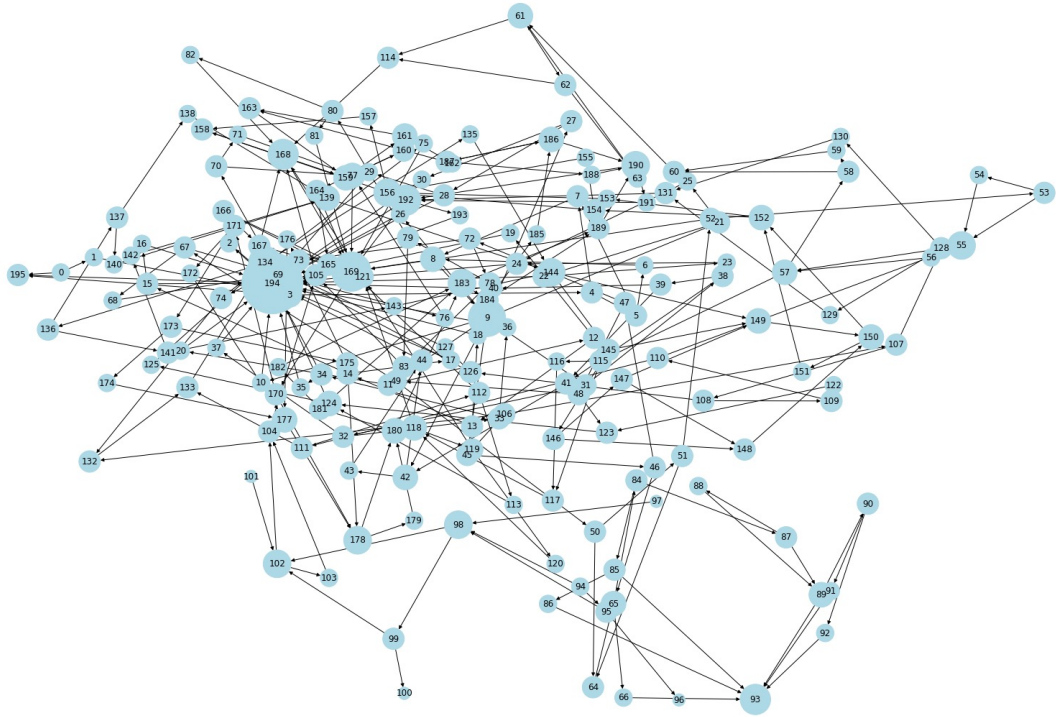
['str', 'conv', 'format', 'uint']



IDA block diagram



FIDL AST



NetworkX CFG

https://www.mandiant.com/resources/fidl-flares-ida-decompiler-library

CAMLIS ‘21 Wrap-Up

Deployment

- Standalone IDA Pro/Ghidra plug-in
- Mandiant’s automated, scalable malware analysis pipeline
- Feedback-driven development from reversers

Machine Language Processing

In the Works

- Further labels/feature consolidation, .NET/ELF binaries, splits
- LM benchmarks, compare to flat assembly code, label hierarchies
- Function dependencies within a binary (e.g. message passing)

MANDIANT

YOUR CYBERSECURITY ADVANTAGE

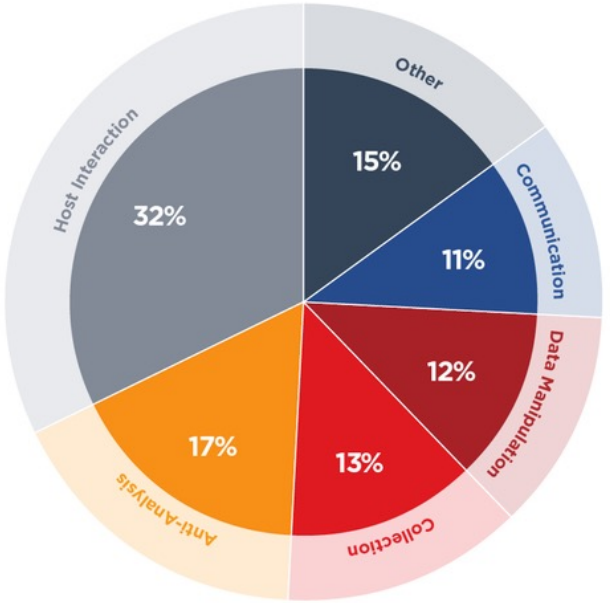
Appendix

CAPA

- capa is an open-source tool that detects capabilities within executables, it tells you what it thinks the program can do
  - Over 600 capability detection rules, covering a wide range of different techniques
  - By passing the -vv flag (very verbose), capa reports capability evidence at virtual function address offsets

- Augmenting our annotation prediction model w/ capa data
  - We ran capa v3 over our dataset of malicious PE files
  - Downselected based on scope: ‘function’ and ‘basic_block’

- capa labels matched only 5% of existing labeled dataset
  - We utilized capa data as an auxiliary task in our CFG2Seq model and saw only a minor improvement in F1 scores (~0.01)




```
$ capa.exe suspicious.exe -vv
...
execute shell command and capture output
namespace   c2/shell
author      matthew.williams@mandiant.com
scope       function
att&ck      Execution::Command and Scripting Interpreter::Windows Command Shell [T1059.003]
references  https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/ns-processthreac
examples    Practical Malware Analysis Lab 14-02.exe_:0x4011C0
function @ 0x10003A13
  and:
    match: create a process with modified I/O handles and window @ 0x10003A13
      and:
        or:
          api: kernel32.CreateProcess @ 0x10003D6D
        number: 0x101 @ 0x10003B03
        or:
          number: 0x44 @ 0x10003ADC
        optional:
          api: kernel32.GetStartupInfo @ 0x10003AE4
    match: create pipe @ 0x10003A13
      or:
        api: kernel32.CreatePipe @ 0x10003ACB
    or:
      string: cmd.exe /c  @ 0x10003AED
...
```