MANDIANT

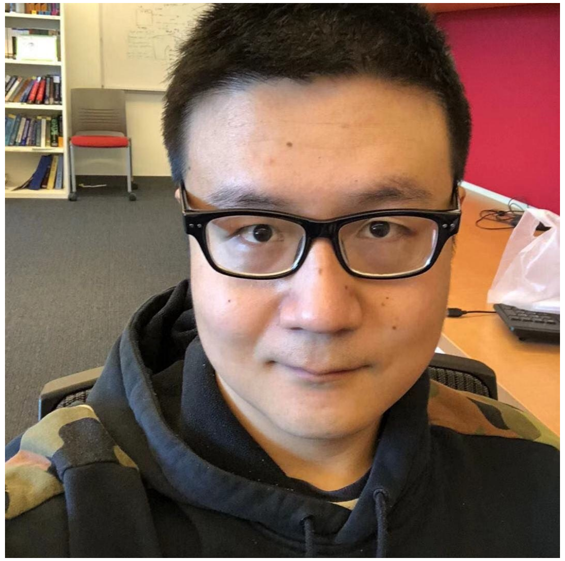
Lightweight, Emulation-Assisted Malware Classification

Xigao Li, David Krisiloff, Scott Coull

Stony Brook University

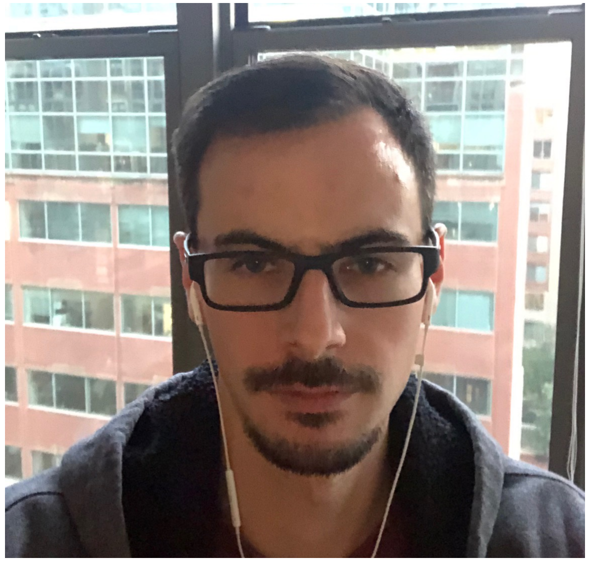
Mandiant Data Science

Who We Are



Graduate student at Stony Brook University

Mandiant Data Science Intern 2021

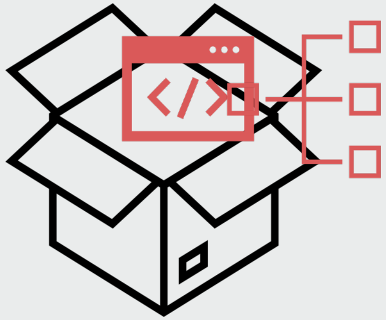


Manager, Data Science at Mandiant

Director, Data Science Research at Mandiant

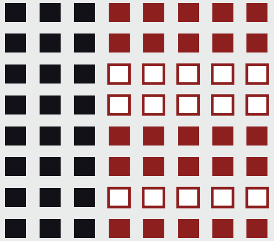
Outline

Emulation Intro



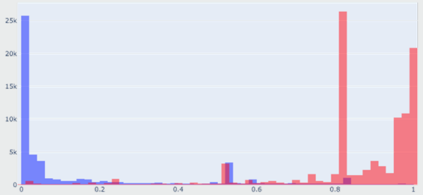
Background and questions we'll answer today

Experiment Design



Emulation features and data sets

First Attempt

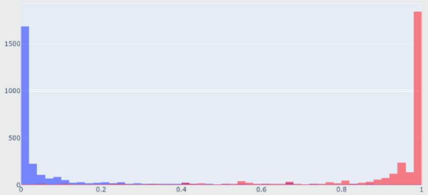


Things don't go quite as planned

Modifying the Emulator

Modifying things for ML purposes, not reverse engineering

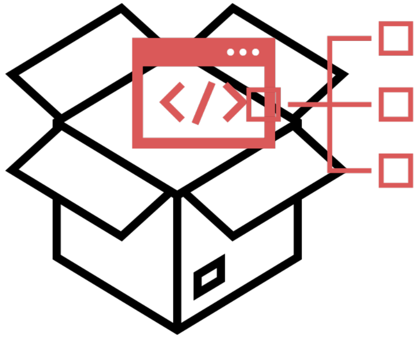
Results



Accuracy and speed results for goodware/malware and malware family tasks

Malware Analysis

- Static: Does this look like malware?
  - Not running program
  - Look for static features like strings, DLLs, etc.
  - May encounter difficulty on obfuscation or packing
  - Fast enough to block malware execution

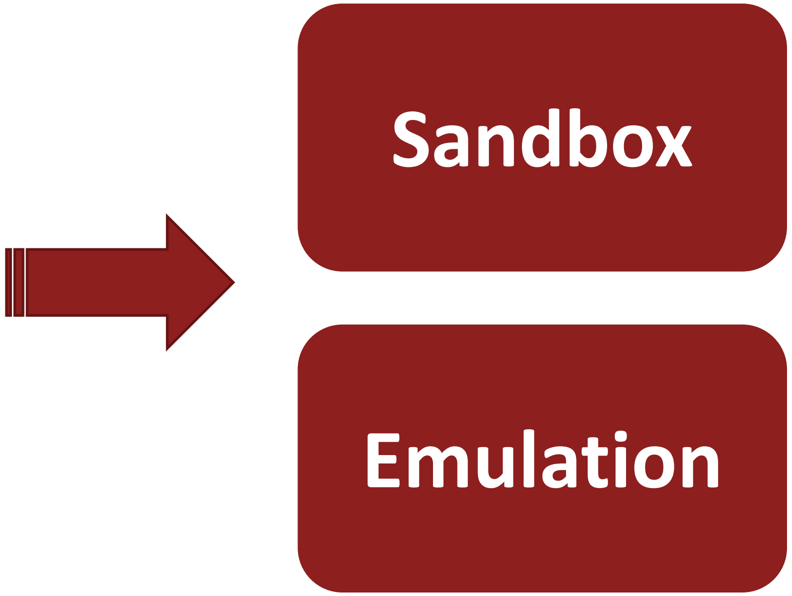


- Dynamic: Does this behave like malware?
  - Runs the program in specific environment
  - Record events logs during execution
  - More effective against obfuscation and packing
  - Not fast enough to block malware execution

Malware Analysis

- Static: Does this look like malware?
  - Not running program
  - Look for static features like strings, DLLs, etc.
  - May encounter difficulty on obfuscation or packing
  - Fast enough to block malware execution

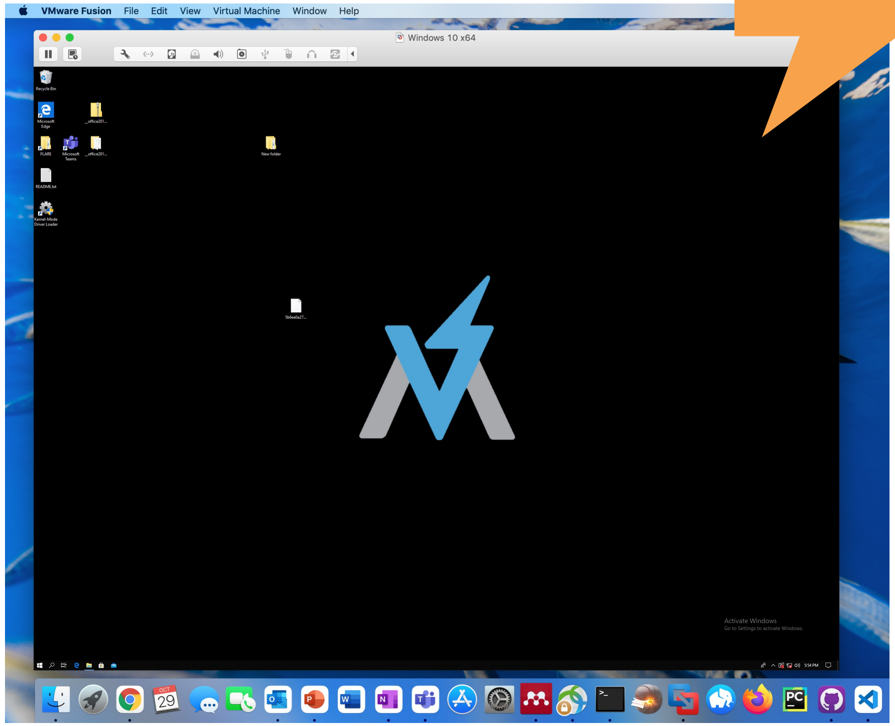
- Dynamic: Does this behave like malware?
  - Runs the program in specific environment
  - Record events logs during execution
  - More effective against obfuscation and packing
  - Not fast enough to block malware execution



Dynamic Analysis

Sandbox

Windows!



- Runs a full OS
- OS implements the system calls
- Heavy weight – need a system image

Emulator

Just the program

- Mocks execution – no OS
- Implement or fake system calls itself
- Lighter weight

Machine Learning + Emulation?

Windows PE in particular

- There are numerous emulators available
- Pre-existing work on ML classifiers based on emulation
  - A lot on Android
  - Microsoft has published work on PE emulation + ML
  - We're assuming a bunch of AV companies have something similar

Showcase

In our knowledge, Unicorn has been used by **123** following products (listed in no particular order).

- Qiling: Cross-platform & multi-architecture lightweight sandbox.
- UniDOS: Microsoft DOS emulator.
- Radare2: Unix-like reverse engineering framework and commandline tools.
- Usercorn: User-space system emulator.
- Unicorn-decoder: A shellcode decoder that can dump self-modifying-code.
- Univm: A plugin for x64dbg for x86 emulation.
- PyAna: Analyzing Windows shellcode.
- GEF: GDB Enhanced Features.
- Pwndbg: A Python plugin of GDB to assist exploit development.
- Eli.Decode: Decode obfuscated shellcodes.
- IdaEmu: an IDA Pro Plugin for code emulation.
- Roper: build ROP-chain attacks on a target binary using genetic algorithms.
- Sk3wlDbg: A plugin for IDA Pro for machine code emulation.
- Angr: A framework for static & dynamic concolic (symbolic) analysis.
- Cemu: Cheap EMUlator based on Keystone and Unicorn engines.
- ROPMEMU: Analyze ROP-based exploitation.
- BroIDS_Unicorn: Plugin to detect shellcode on Bro IDS with Unicorn.
- UniAna: Analysis PE file or Shellcode (Only Windows x86).
- ARMSCGen: ARM Shellcode Generator.
- TinyAntivirus: Open source Antivirus engine designed for detecting & disinfecting polymorphic virus.
- Patchkit: A powerful binary patching toolkit.
- Arpilnik: Very simple arithmetric expression compiler for x86_64 machines.
- Shellbug: Basic command-line, text-based, shellcode debugger.
- GCTF-Challenges: An assembly based puzzle at GryphonCTF 2016.
- Sibyl: A Miasm2 based function divination.
- Kadabra: A blanked execution framework.

Example emulation packages using the unicorn CPU emulator

https://www.unicorn-engine.org/showcase/

R. Agrawal, J. W. Stokes, M. Marinescu, and K. Selvaraj, *Proc. - IEEE Mil. Commun. Conf. MILCOM*, vol. 2019-Octob, pp. 571–578, 2019.
B. Amos, H. Turner, and J. White, *2013 9th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2013*, pp. 1666–1671, 2013.

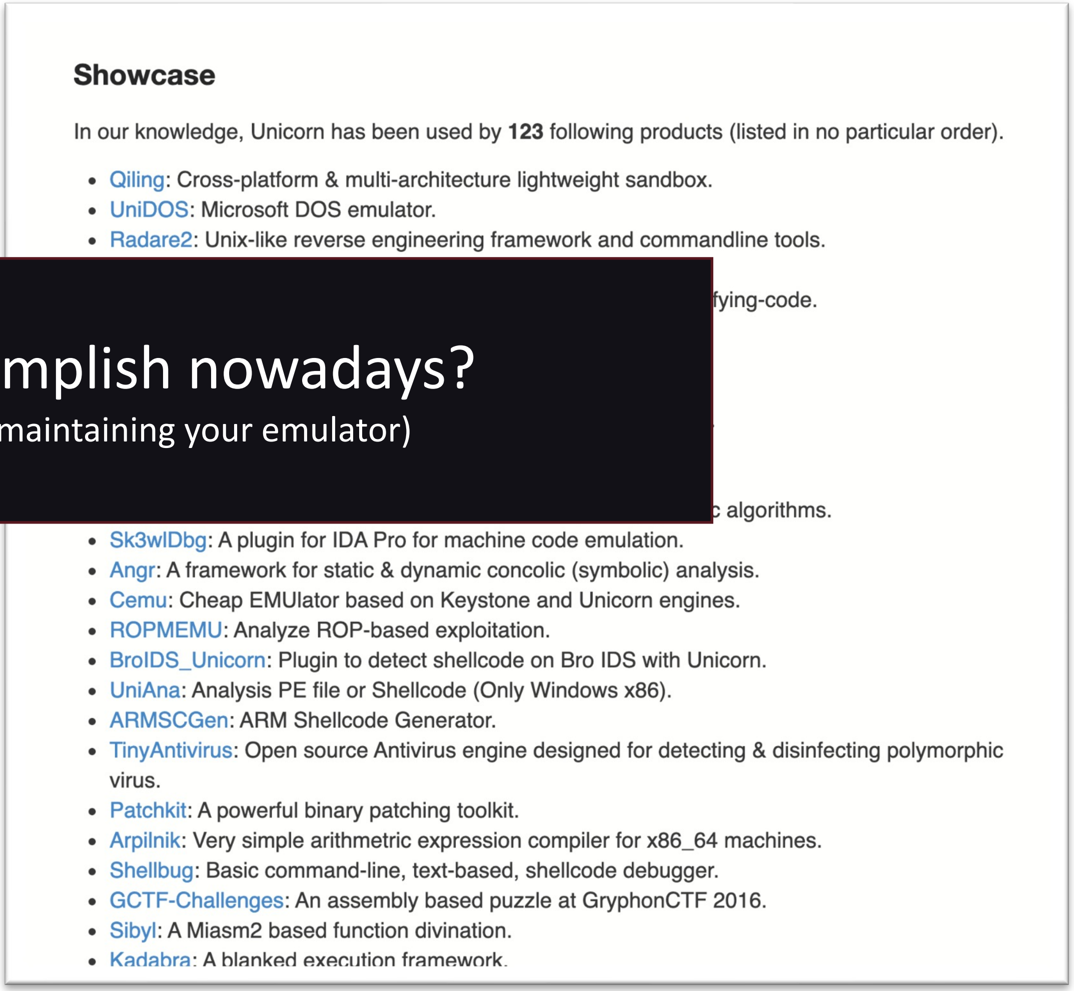
Machine Learning + Emulation?

Windows PE in particular

- There are numerous emulators available
- Pre-existing wor
emulation
  - A lot on Androic
  - Microsoft has published work on PE emulation + ML
  - We're assuming a bunch of AV companies have something similar

How easy is this to accomplish nowadays?

(Especially if you don't have a team maintaining your emulator)



Showcase

In our knowledge, Unicorn has been used by **123** following products (listed in no particular order).

- Qiling: Cross-platform & multi-architecture lightweight sandbox.
- UniDOS: Microsoft DOS emulator.
- Radare2: Unix-like reverse engineering framework and commandline tools.

fying-code.

c algorithms.

- Sk3wlDbg: A plugin for IDA Pro for machine code emulation.
- Angr: A framework for static & dynamic concolic (symbolic) analysis.
- Cemu: Cheap EMUlator based on Keystone and Unicorn engines.
- ROPMEMU: Analyze ROP-based exploitation.
- BroIDS_Unicorn: Plugin to detect shellcode on Bro IDS with Unicorn.
- UniAna: Analysis PE file or Shellcode (Only Windows x86).
- ARMSCGen: ARM Shellcode Generator.
- TinyAntivirus: Open source Antivirus engine designed for detecting & disinfecting polymorphic virus.
- Patchkit: A powerful binary patching toolkit.
- Arpilnik: Very simple arithmetric expression compiler for x86_64 machines.
- Shellbug: Basic command-line, text-based, shellcode debugger.
- GCTF-Challenges: An assembly based puzzle at GryphonCTF 2016.
- Sibyl: A Miasm2 based function divination.
- Kadabra: A blanked execution framework.

Example emulation packages using the unicorn CPU emulator

https://www.unicorn-engine.org/showcase/

R. Agrawal, J. W. Stokes, M. Marinescu, and K. Selvaraj, *Proc. - IEEE Mil. Commun. Conf. MILCOM*, vol. 2019-Octob, pp. 571–578, 2019.

B. Amos, H. Turner, and J. White, *2013 9th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2013*, pp. 1666–1671, 2013.

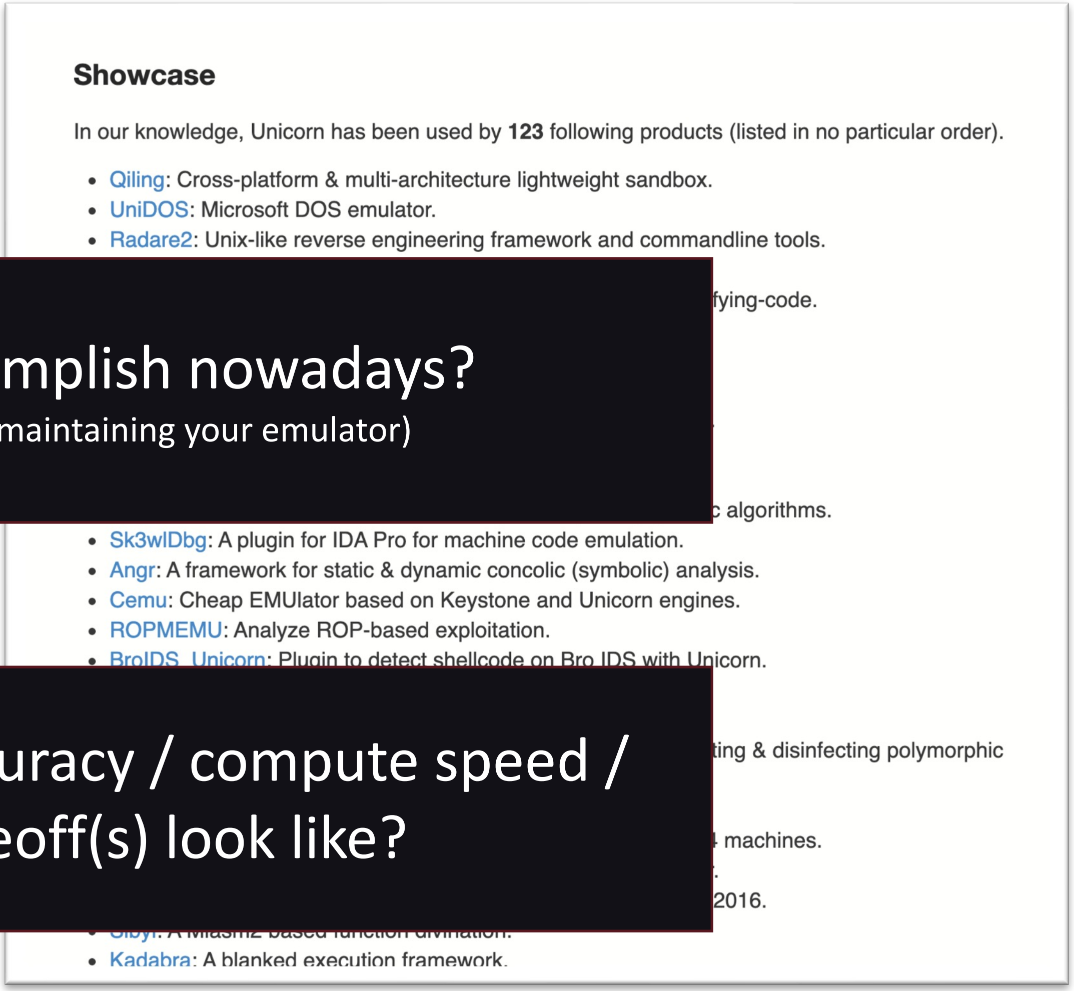
Machine Learning + Emulation?

Windows PE in particular

- There are numerous emulators available
- Pre-existing wor
  emulation
  - A lot on Androi
  - Microsoft has published work on PE emulation + ML
  - We're assuming a bunch of AV companies have something simil

How easy is this to accomplish nowadays?
(Especially if you don't have a team maintaining your emulator)

What does the emulation accuracy / compute speed / model accuracy tradeoff(s) look like?



Example emulation packages using the unicorn CPU emulator

https://www.unicorn-engine.org/showcase/

R. Agrawal, J. W. Stokes, M. Marinescu, and K. Selvaraj, *Proc. - IEEE Mil. Commun. Conf. MILCOM*, vol. 2019-Octob, pp. 571–578, 2019.
B. Amos, H. Turner, and J. White, *2013 9th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2013*, pp. 1666–1671, 2013.

Speakeasy

- A lightweight emulator aiming for acquiring the triage reports in automated way
- Open-Source package from Mandiant
- Designed for Windows malware
- Configurable environments
- Can add various limitations for partial running

THREAT RESEARCH

Emulation of Malicious Shellcode With Speakeasy

ANDREW DAVIS

AUG 26, 2020 | 15 MINS READ

#THREAT RESEARCH

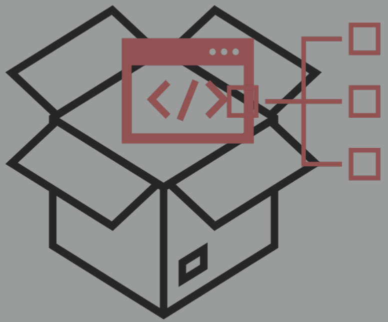
In order to enable emulation of malware samples at scale, we have developed the Speakeasy emulation framework. Speakeasy aims to make it as easy as possible for users who are not malware analysts to acquire triage reports in an automated way, as well as enabling reverse engineers to write custom plugins to triage difficult malware families.

Originally created to emulate Windows kernel mode malware, Speakeasy now also supports user mode samples. The project's main goal is high resolution emulation of the Windows operating system for dynamic malware analysis for the x86 and amd64 platforms. Similar emulation frameworks exist to emulate user mode binaries. Speakeasy attempts to differentiate from other emulation frameworks the following ways:

- Architected specifically around emulation of Windows malware
- Supports emulation of kernel mode binaries to analyze difficult to triage rootkits

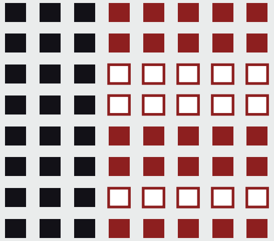
Experiment Design

Emulation Intro



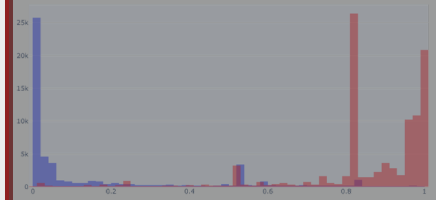
Background and questions we'll answer today

Experiment Design



Emulation features and data sets

First Attempt

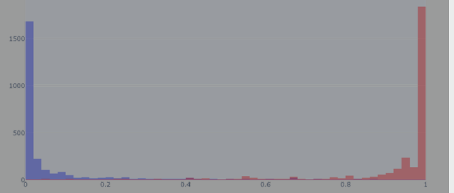


Things don't quite go as planned

Modifying the Emulator

Modifying things for ML purposes not reverse engineering

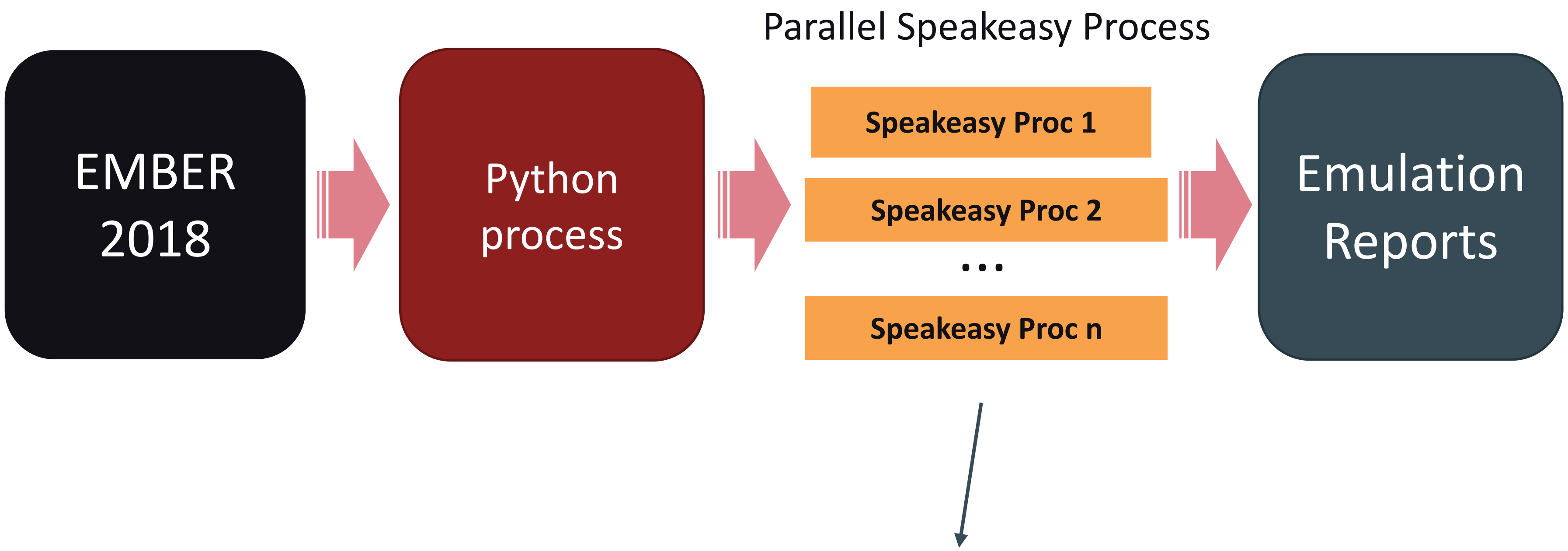
Results



Accuracy and speed results for goodware/malware and malware family tasks

Experimental Setup (1)

Emulation Pipeline



Controlled emulation process, limits on:

- Execution time
- RAM
- **Total # of instructions**

What Do We Get From Emulation?

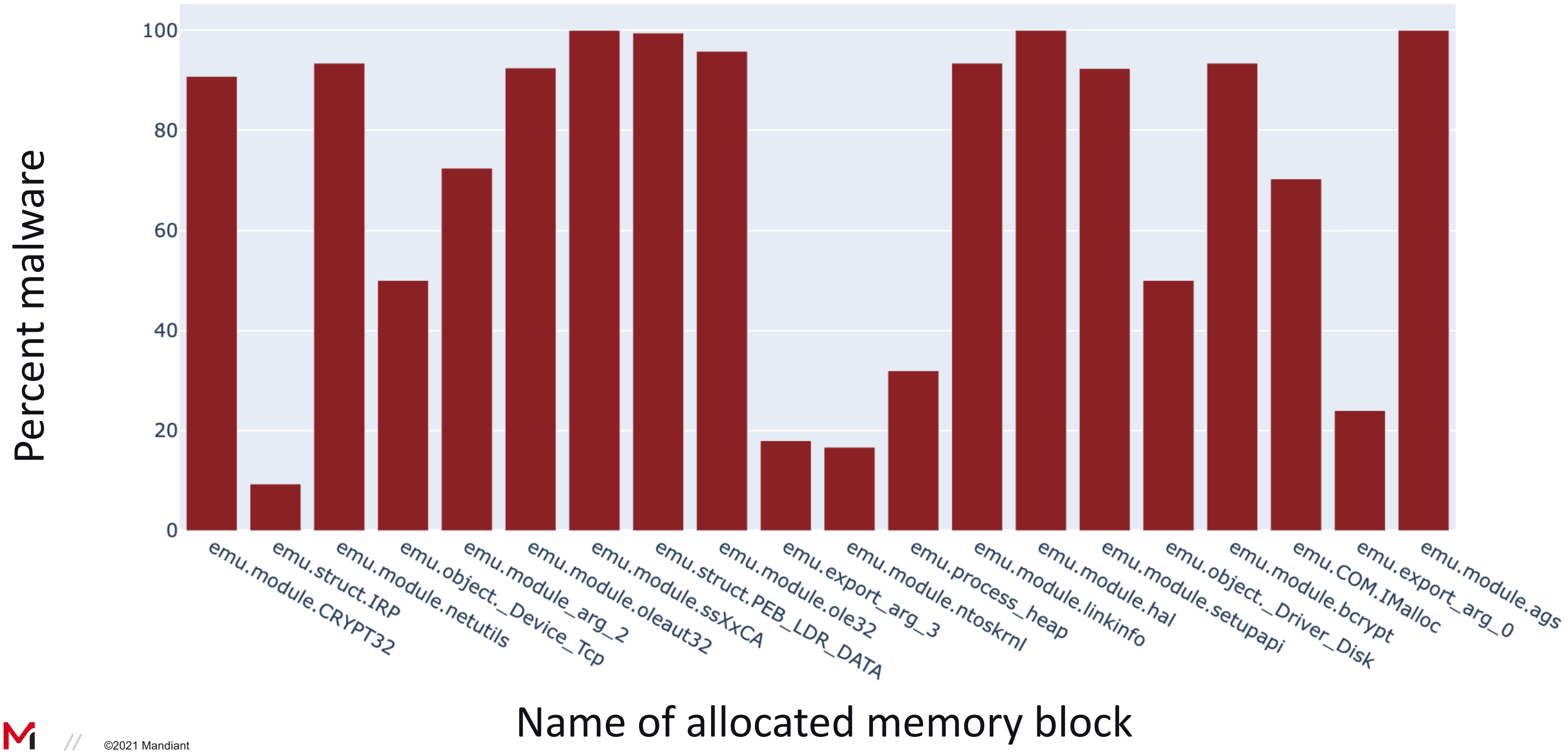
Instruction counter	Name	Returned	Arguments
0x401688	advapi32.CryptCreateHash	0x1	"0x680", "CALG_MD5", "0x0", …
0x4016a8	advapi32.CryptHashData	0x1	"0x2804","0x50000", …
0x401724	user32.wsprintfA	0x2	"00", "%02X"

Sequence of external APIs called

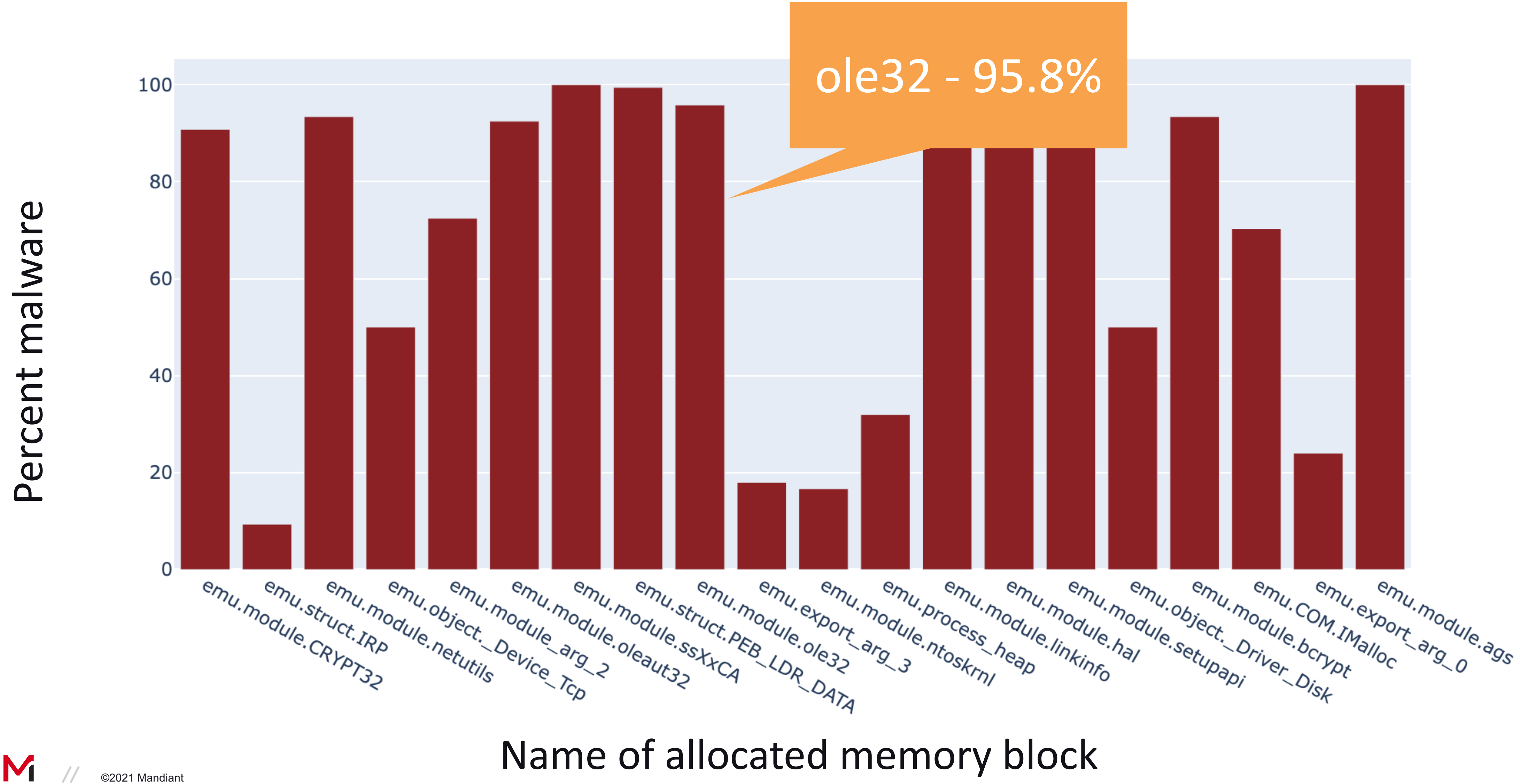
Name	Read	Write	Execute
Loaded binary file	61129	33587	492185
Program stack	106541	62419	0

Memory access statistics

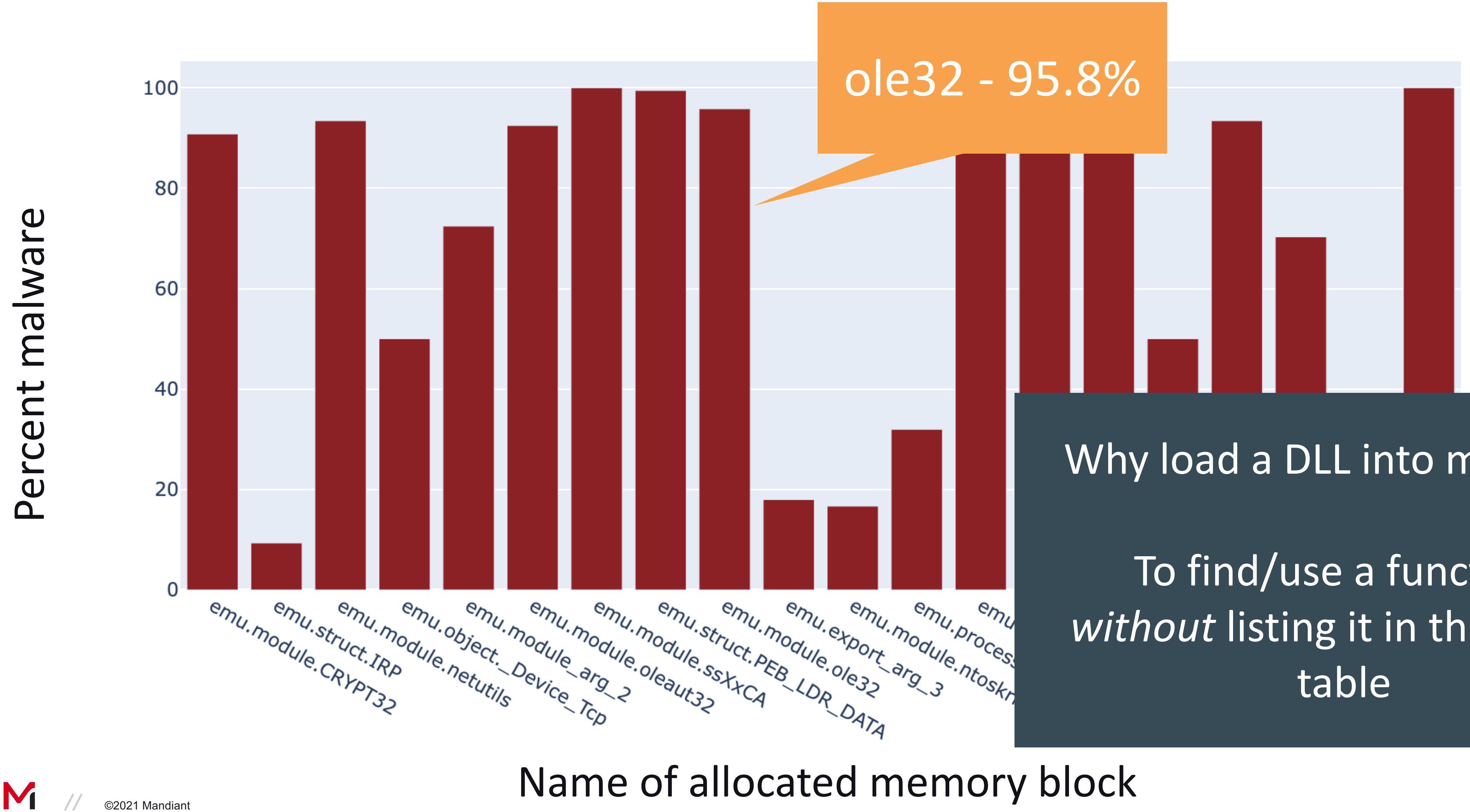
Is This Useful for Classification?



Is This Useful for Classification?



Is This Useful for Classification?



Why load a DLL into memory?

To find/use a function, *without* listing it in the import table

Feature Engineering

APIs	hash trick
	bag of words
	n-grams?
Memory section names	hash trick
	bag of words
Memory access	read/write/execute counts as integers
	$X[h(name + "-read")] += reads$

Feature Engineering

APIs	hash trick
	bag of words
	n-grams?
Memory section names	hash trick
	bag of words
Memory access	read/write/execute counts as integers
	$X[h(name + "-read")] += reads$

← For this talk we're sticking to bag of *individual* words

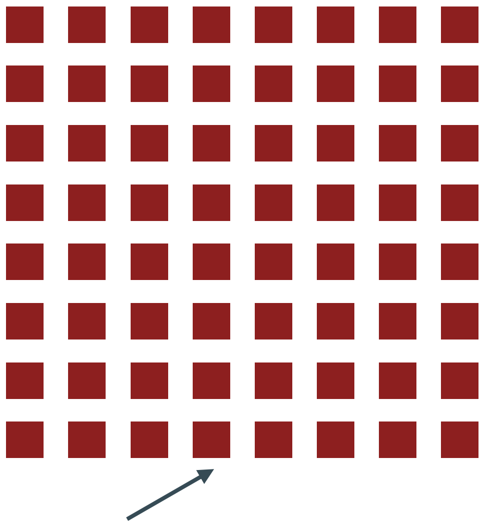
Feature Engineering

APIs	hash trick	
	bag of words	
	n-grams?	← For this talk we're sticking to bag of *individual* words
Memory section names	hash trick	
	bag of words	
Memory access	read/write/execute counts as integers	
	$X[h(name + "-read")] += reads$	← Provides potentially interesting evidence of unpacking (write + execute)

Experimental Setup (2)

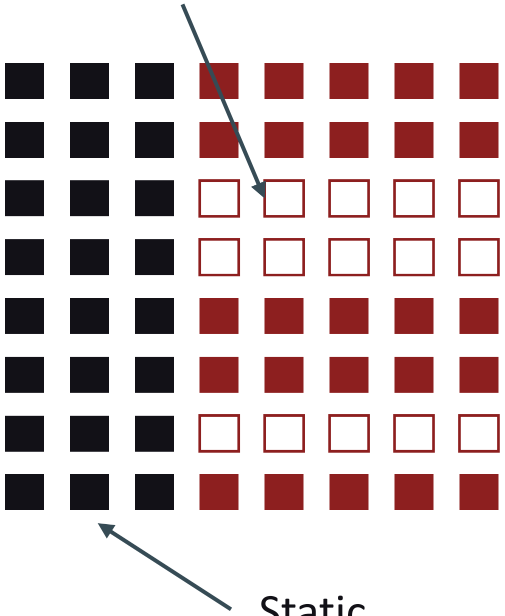
How we're modeling

Just the files that emulated



- Using only emulation features
- Makes measuring changes to the emulator easy
- Ignores a bunch of files (.NET)

All of EMBER

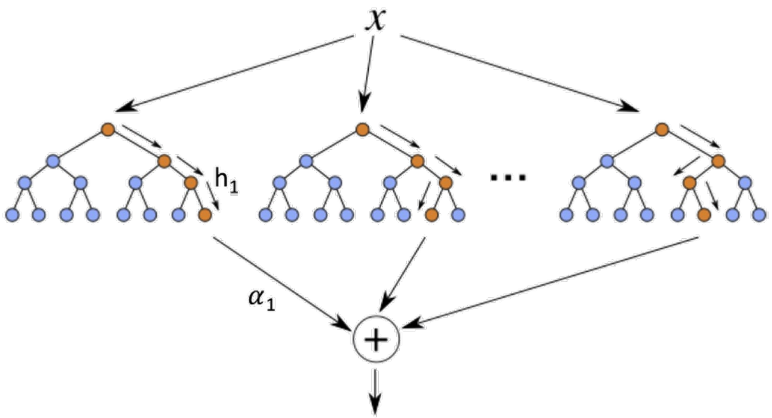


- Using static and emulation features
- More production realistic
- Missing emulation features are encoded as -1

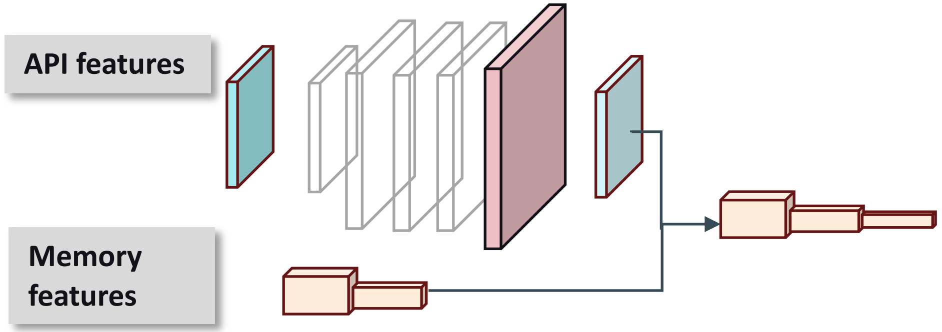
We'll look primarily at goodware/malware classification, but we also experiment with malware family classification

A Note on Model Choice

LightGBM

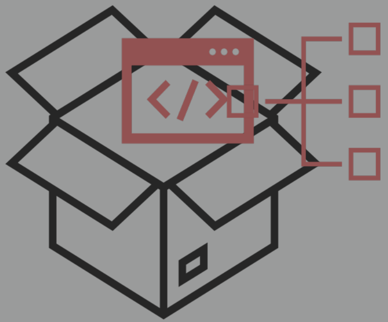
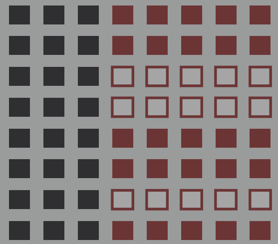
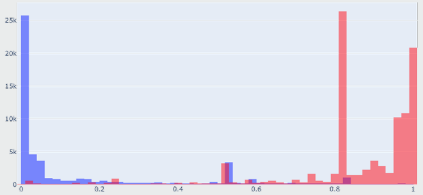
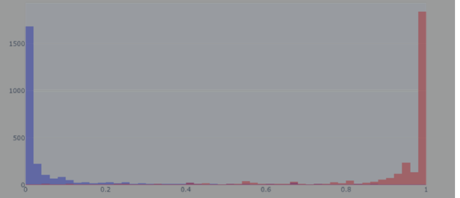


Neural Network

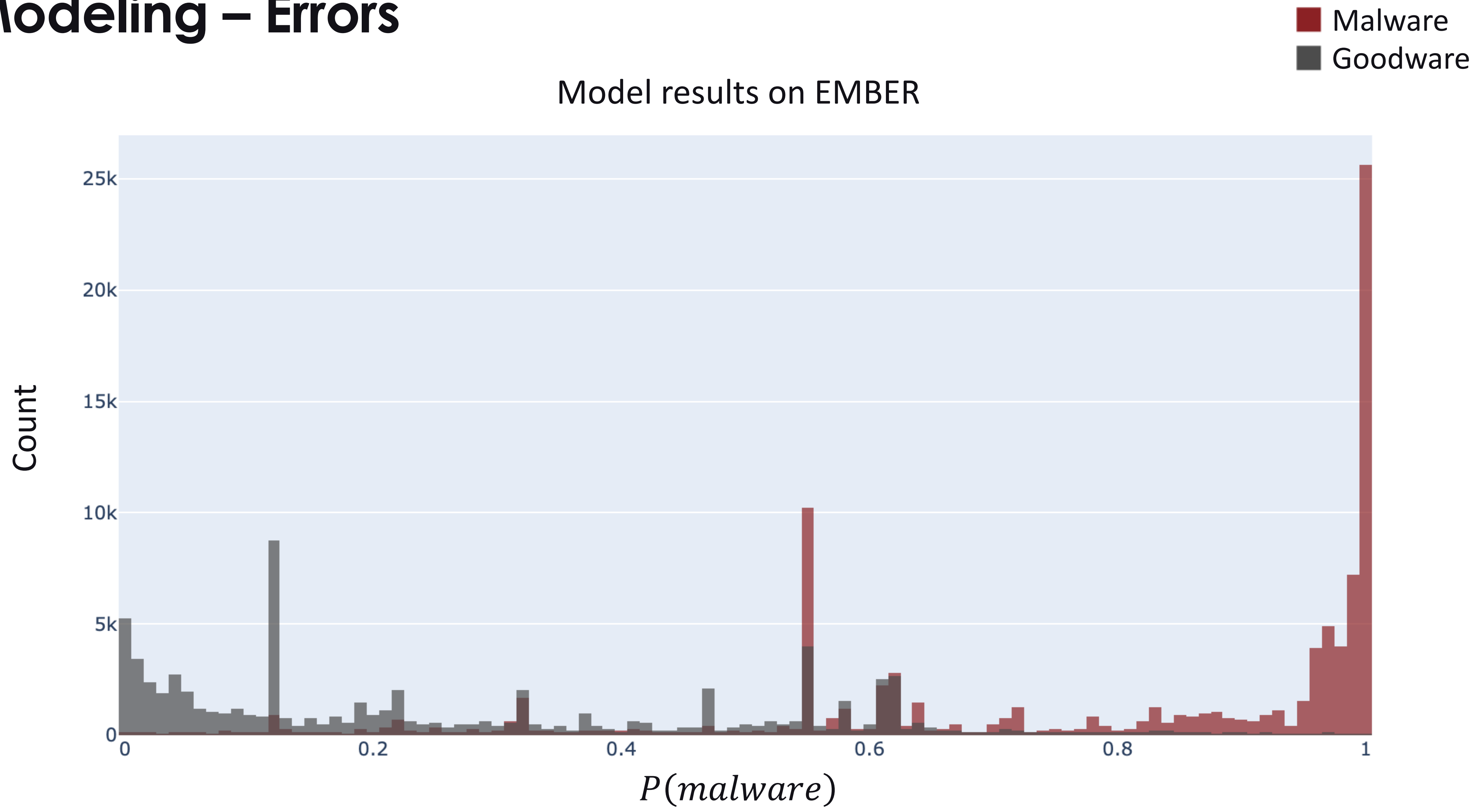


We explored both LightGBM (gradient boosted trees) and various neural network architectures. We got the best performance from LightGBM, but our search was hardly exhaustive.

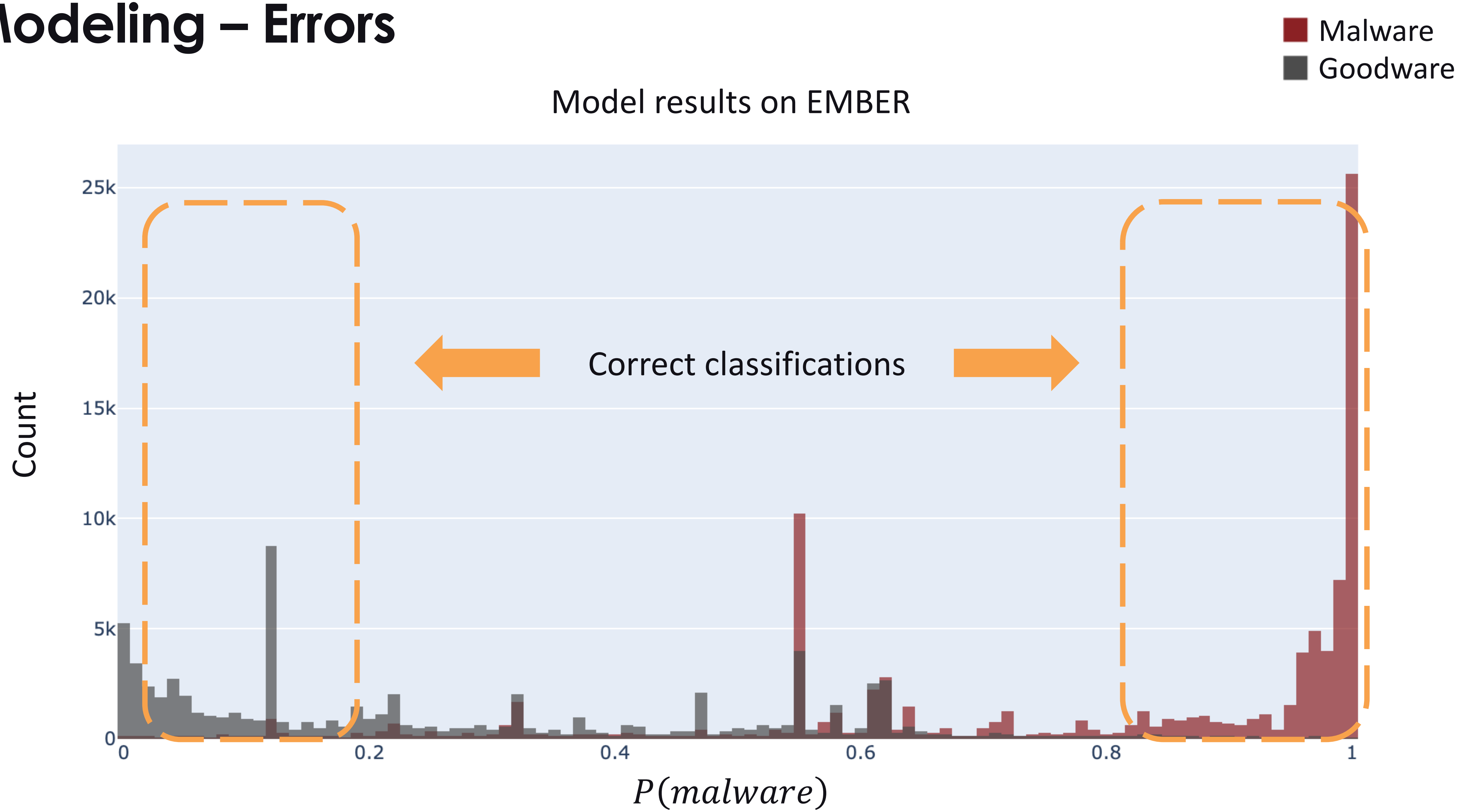
First Attempt

Emulation Intro	Experiment Design	**First Attempt**	Modifying the Emulator	Results
				
Background and questions we'll answer today	Emulation features and data sets	Things don't quite go as planned	Modifying things for ML purposes not reverse engineering	Accuracy and speed results for goodware/malware and malware family tasks

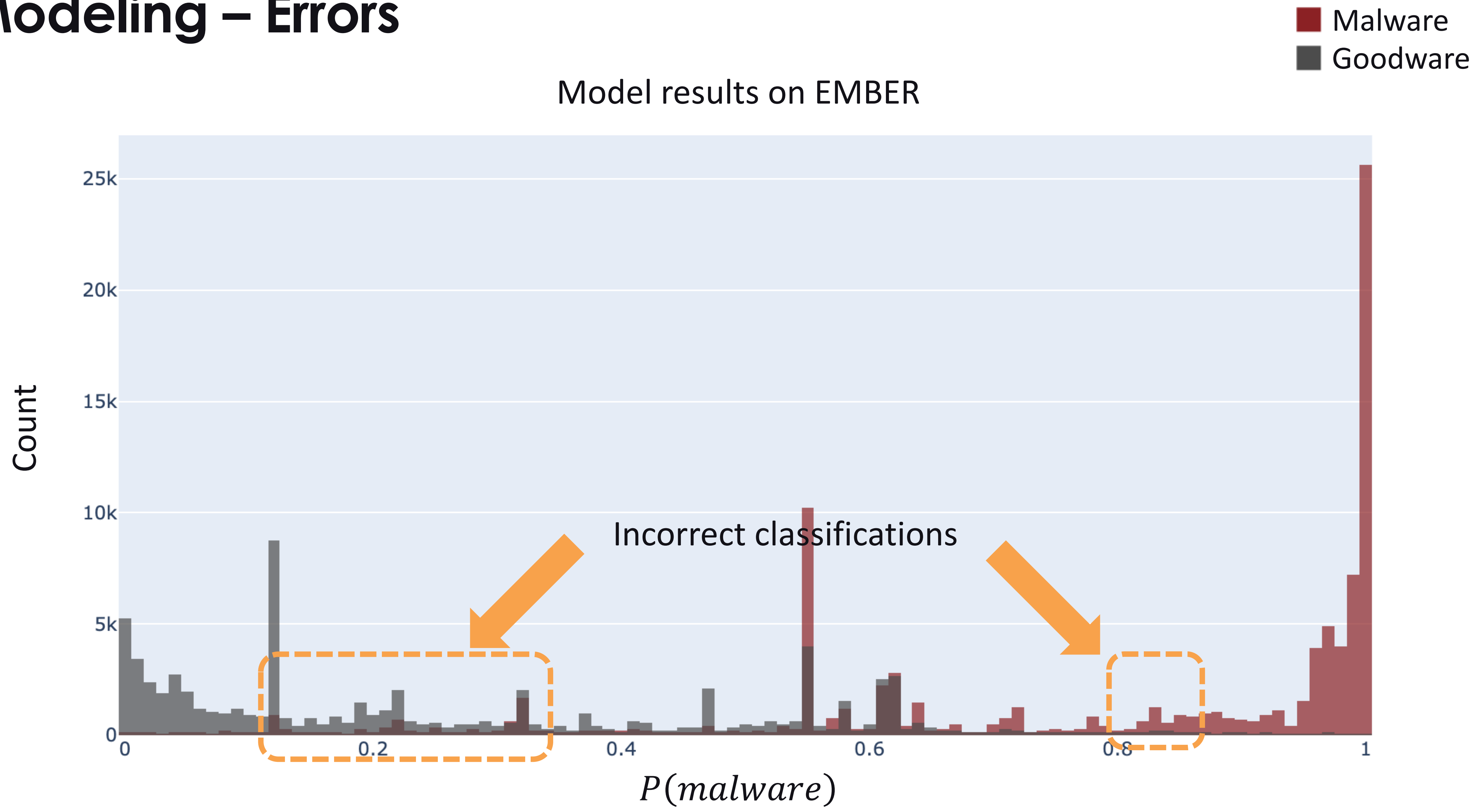
Modeling – Errors



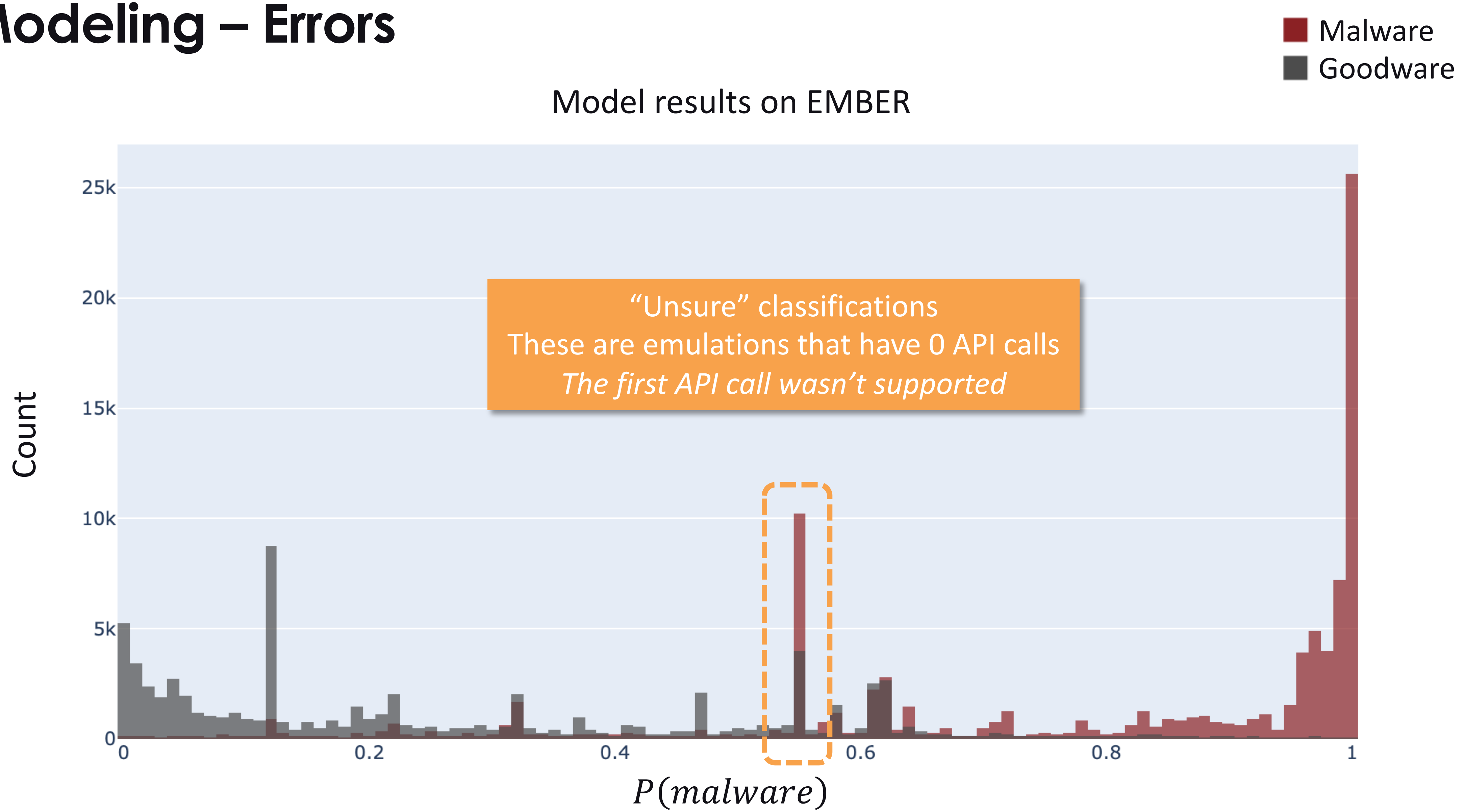
Modeling – Errors



Modeling – Errors



Modeling – Errors



Handling External APIs

The emulator needs to mock the API call

- The return value
- Occasionally shuffling value into/out of memory registers
- Side effects
  - Opening files
  - Editing registry keys

There are more than 2,000 functions in kernel32.dll alone

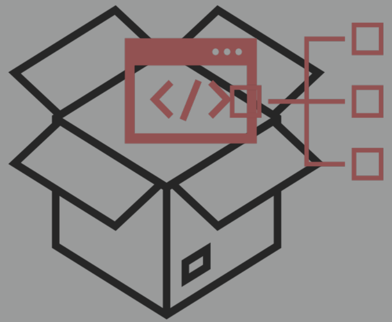
Unsurprisingly, a common anti-emulation technique is to call an obscure API that an emulator is unlikely to mock.

“We must think more carefully about the assumptions and beliefs that we bring to a problem.”

-NATE SILVER

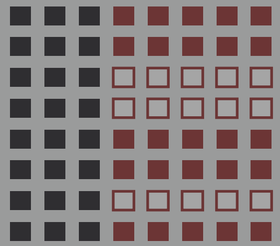
Modifying the Emulator

Emulation Intro



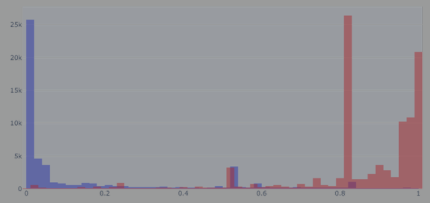
Background and questions we'll answer today

Experiment Design



Emulation features and data sets

First Attempt

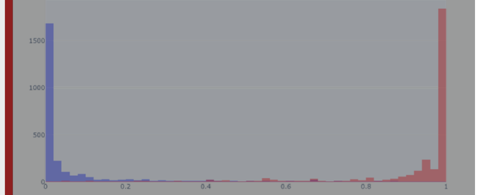


Things don't quite go as planned

Modifying the Emulator

Modifying things for ML purposes not reverse engineering

Results



Accuracy and speed results for goodware/malware and malware family tasks

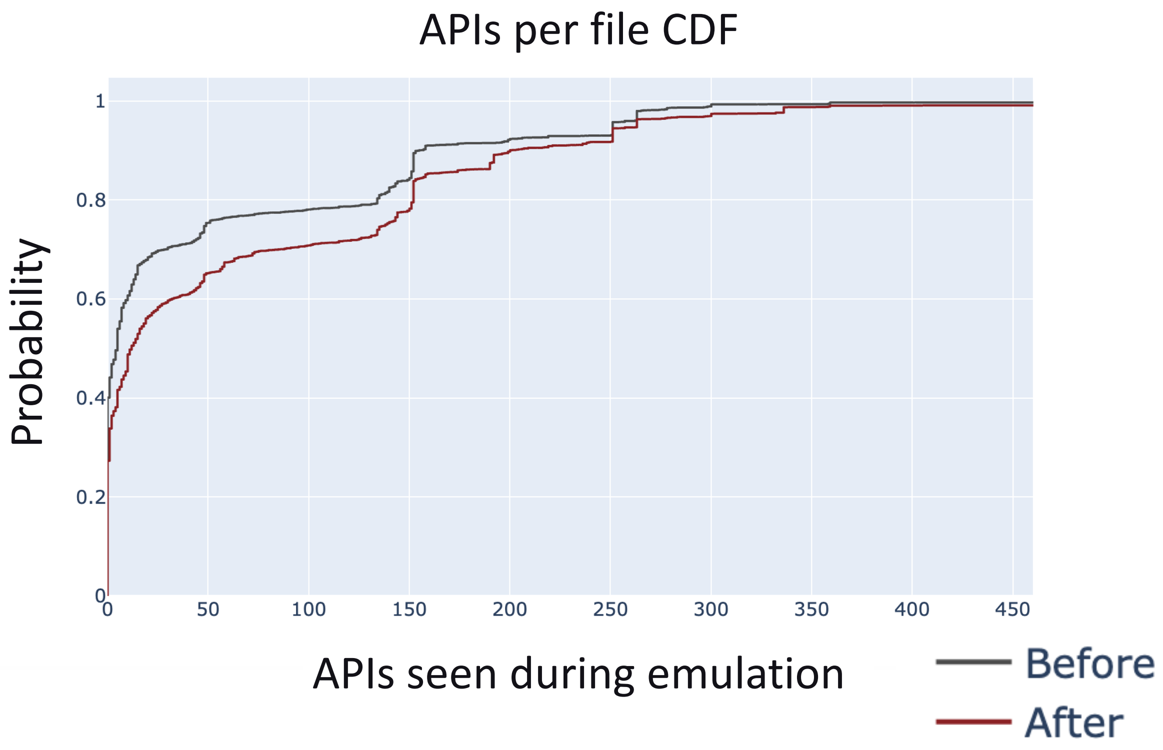
Modifying the Emulator

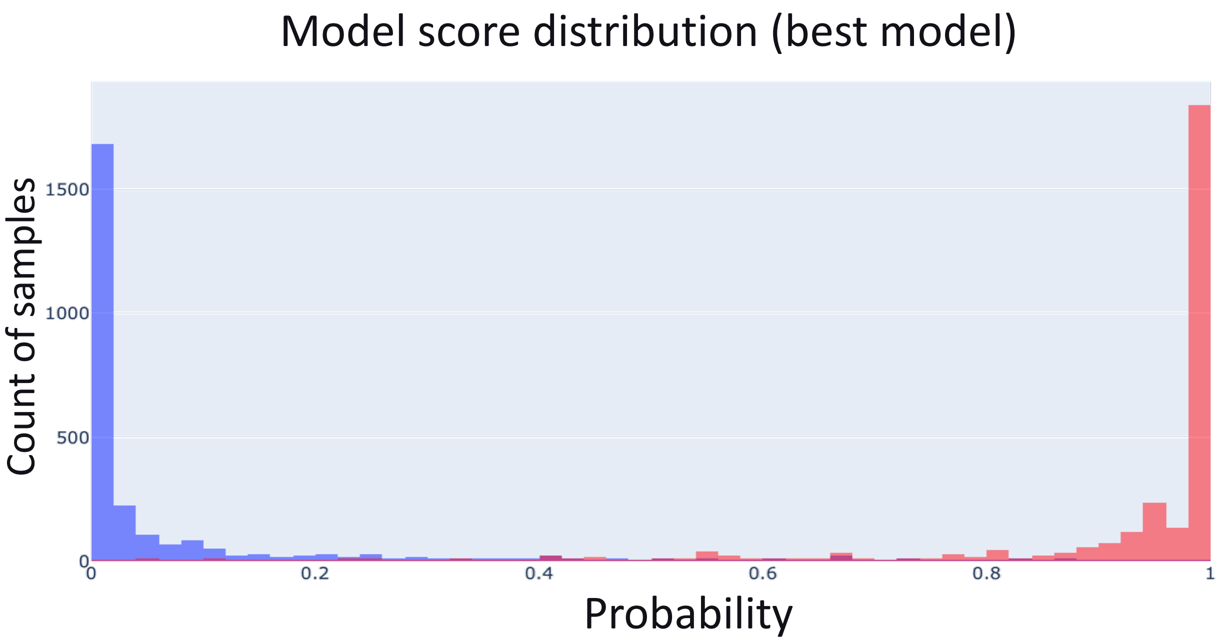
- How accurate does the emulation need to be to be useful for an ML model?
  - We do not really need the program to run "*correctly*", we just need them "*running*".
- If we faked the API, what would happen?
  - The emulation would continue
  - At some point it'll probably *segfault*
  - Overall, we'll get more information but with increased noise

```
def unknown_api():
        return 0
```

How we're "supporting" unknown APIs

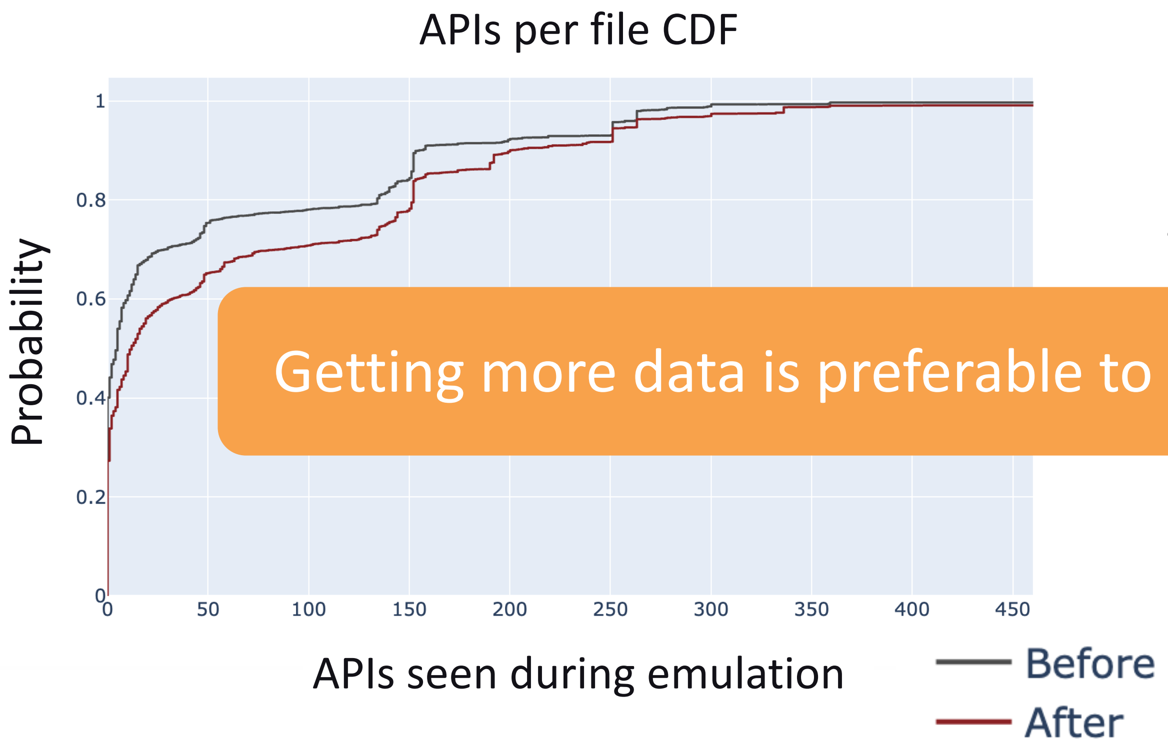
Improvements on Speakeasy

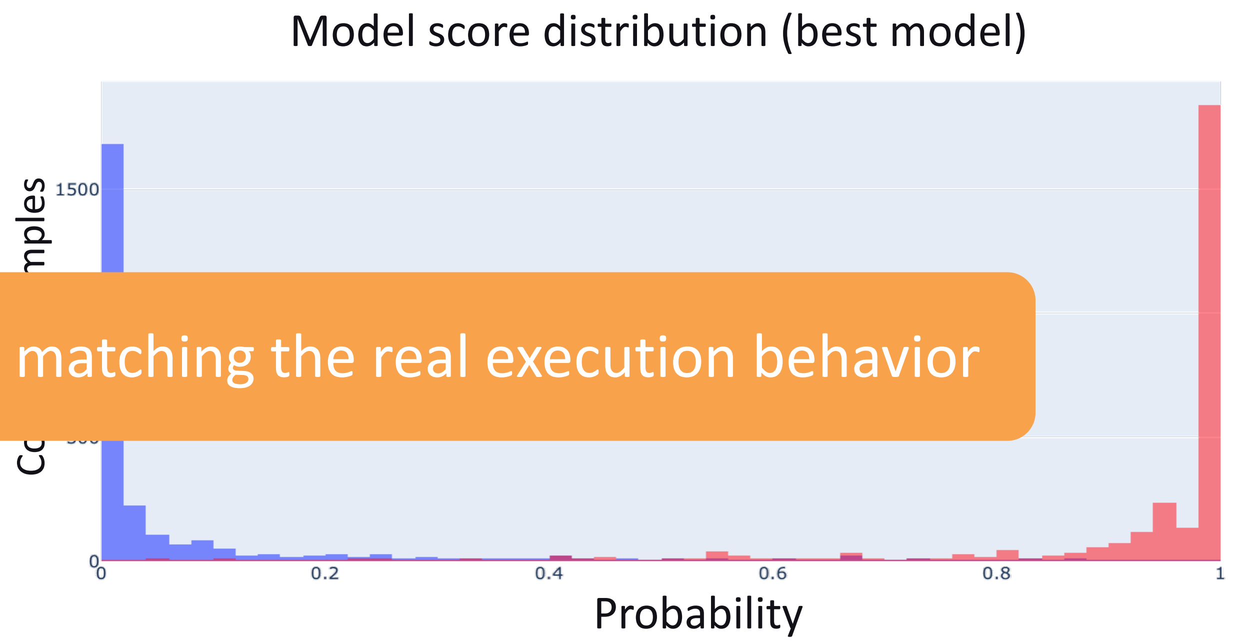




	Before	After
Total APIs	6,958,540	19,213,248
Total memory allocations	1,868,206	3,445,727

Improvements on Speakeasy



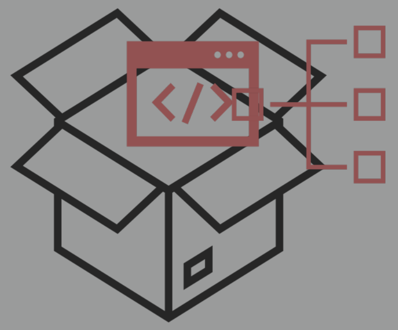


Getting more data is preferable to matching the real execution behavior

	Before	After
Total APIs	6,958,540	19,213,248
Total memory allocations	1,868,206	3,445,727

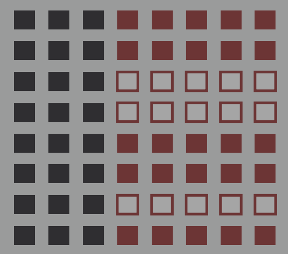
Results: Accuracy and Speed

Emulation Intro



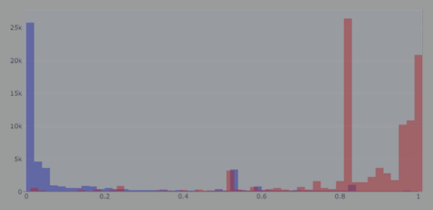
Background and questions we'll answer today

Experiment Design



Emulation features and data sets

First Attempt

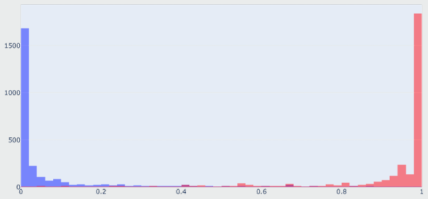


Things don't quite go as planned

Modifying the Emulator

Modifying things for ML purposes not reverse engineering

Results

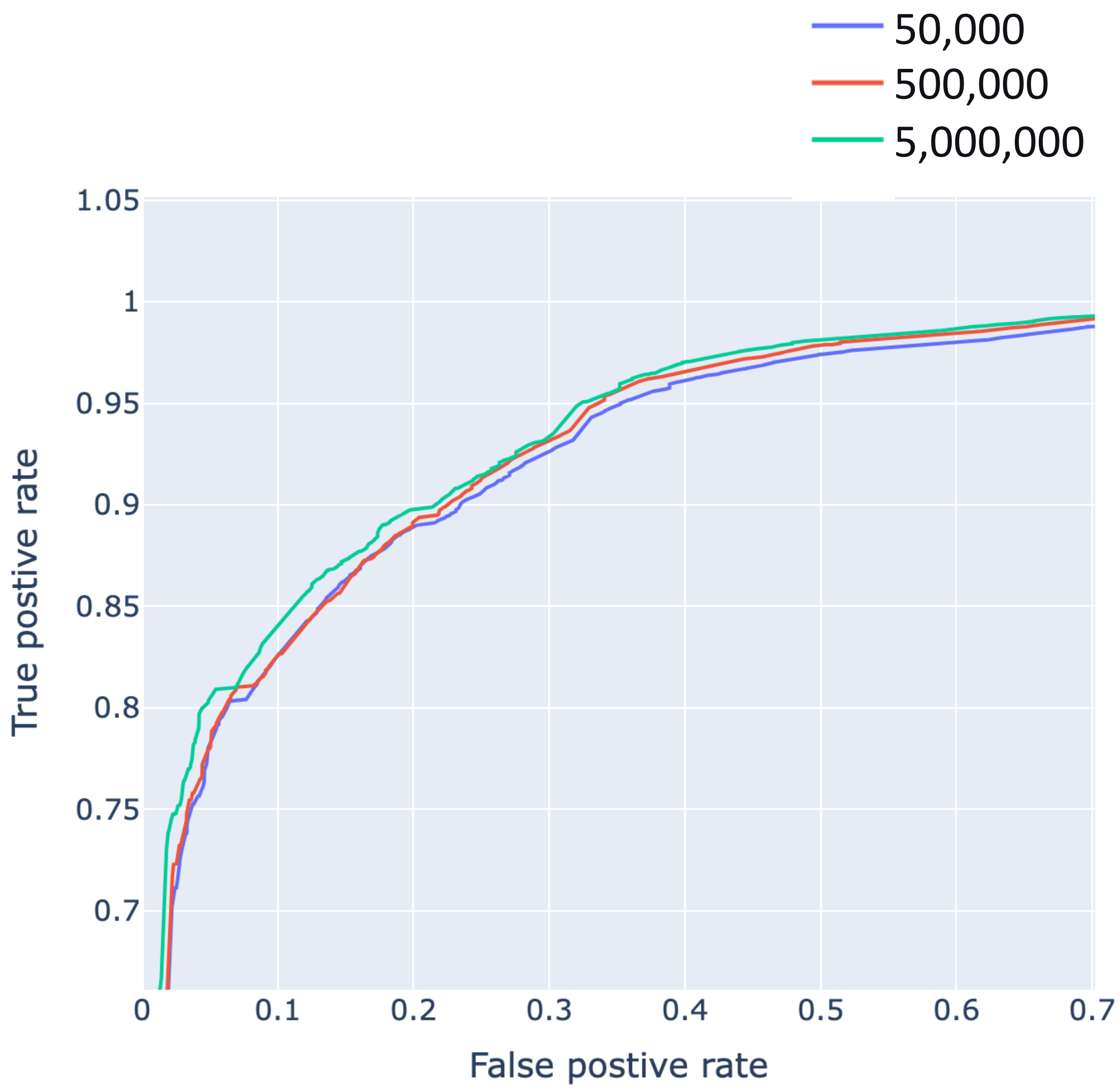


Accuracy and speed results for goodware/malware and malware family tasks

Goodware/Malware Task

Just on emulated files

Maximum instructions	Median emulation time (s)	AUROC
50,000	0.96	0.9375
500,000	1.40	0.9409
5,000,000	1.82	0.9457



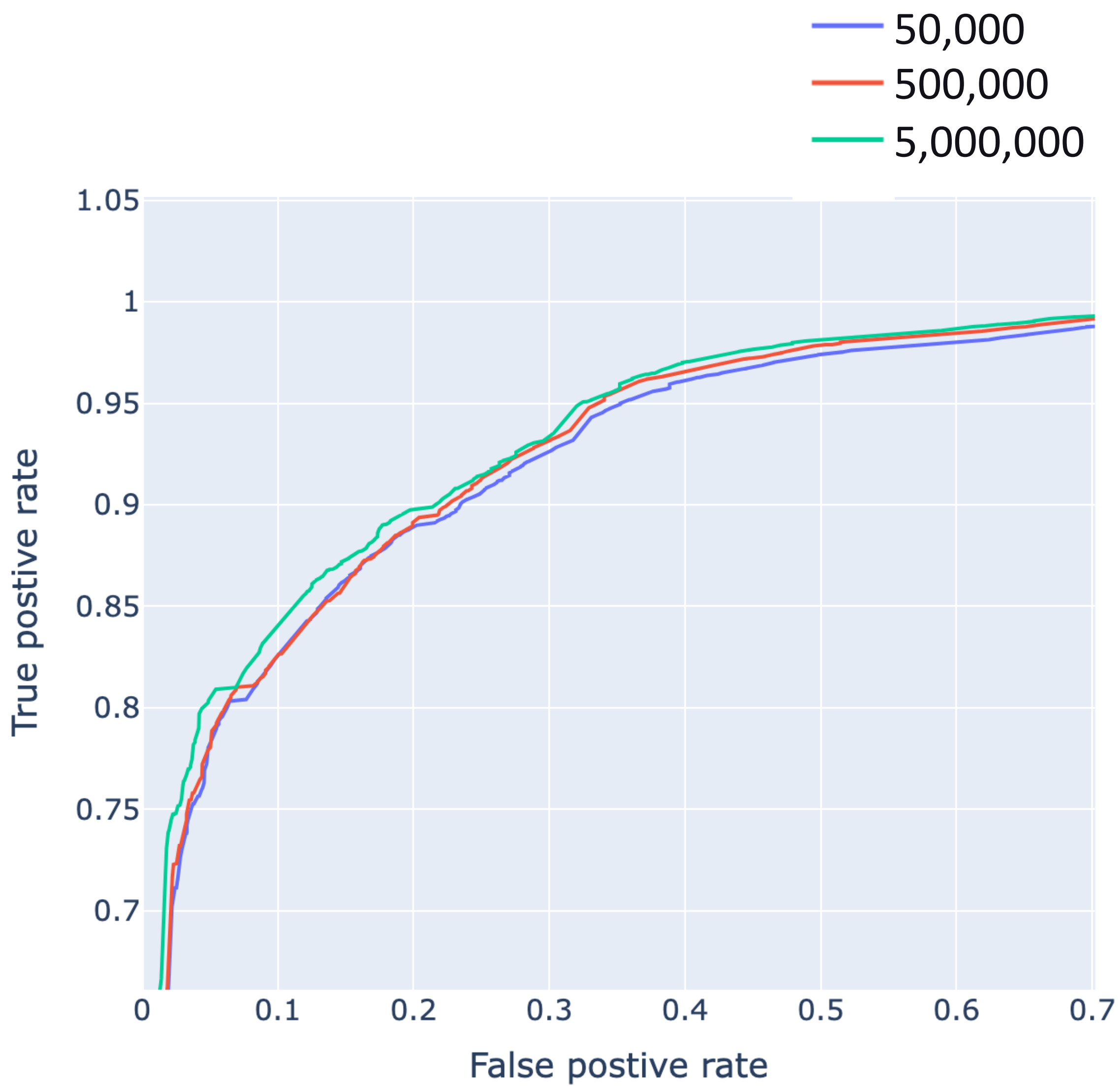
Our classifier performs better on longer emulation runs. Note however that even at a fast setting you getting reasonable performance

Goodware/Malware Task

Just on emulated files

Maximum instructions	Median emulation time (s)	AUROC
50,000	0.96	0.9375
500,000	1.40	0.9409
5,000,000	1.82	0.9457

We can get reasonable performance with <1s emulation time.

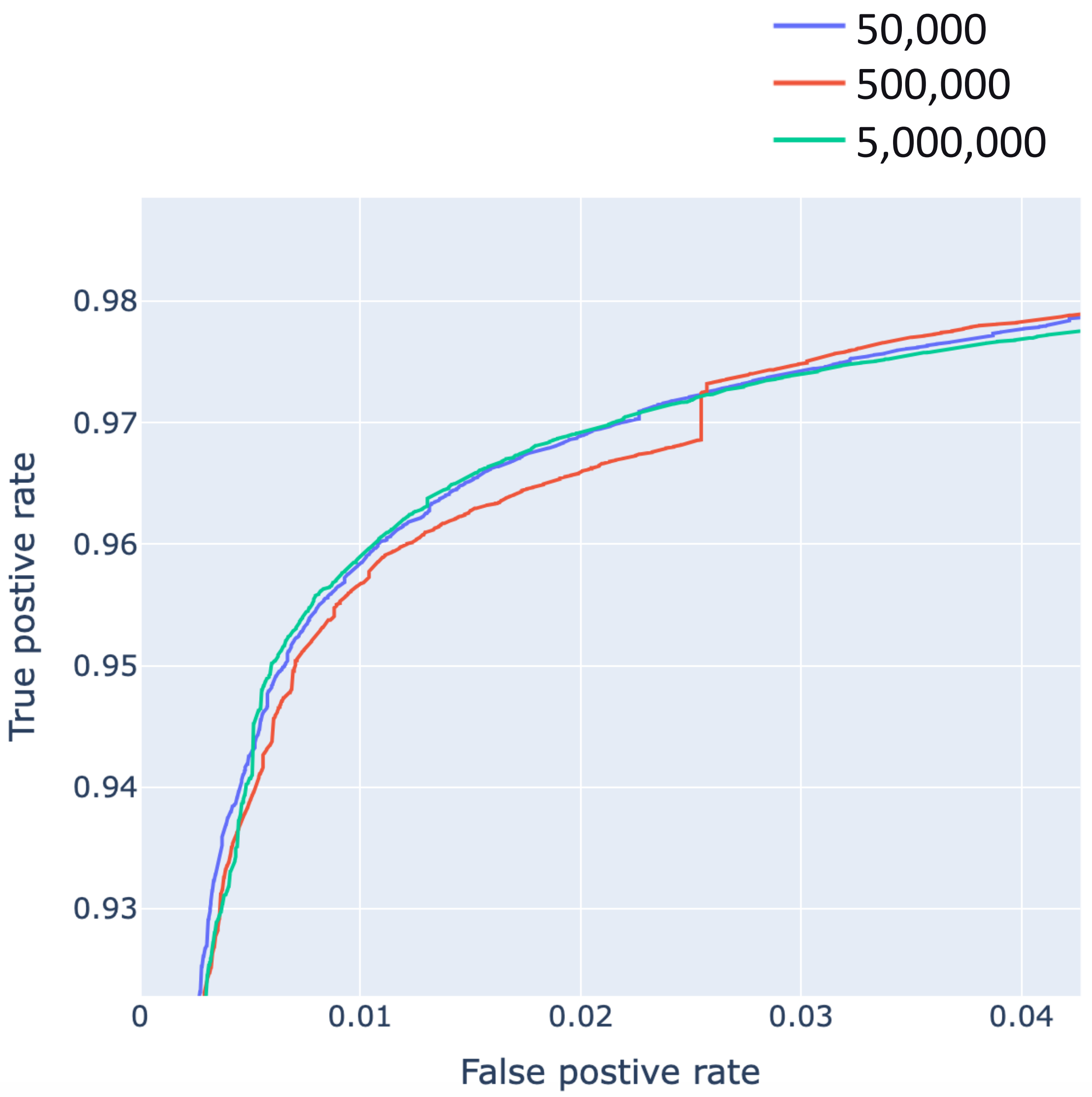


Our classifier performs better on longer emulation runs. Note however that even at a fast setting you getting reasonable performance

Goodware/Malware Task

All of EMBER 2018

Maximum instructions	Median emulation time (s)	AUROC
50,000 + Static	0.96	.9954
500,000 + Static	1.40	.9953
5,000,000 + Static	1.82	.9951
Only Static features	-	.9951



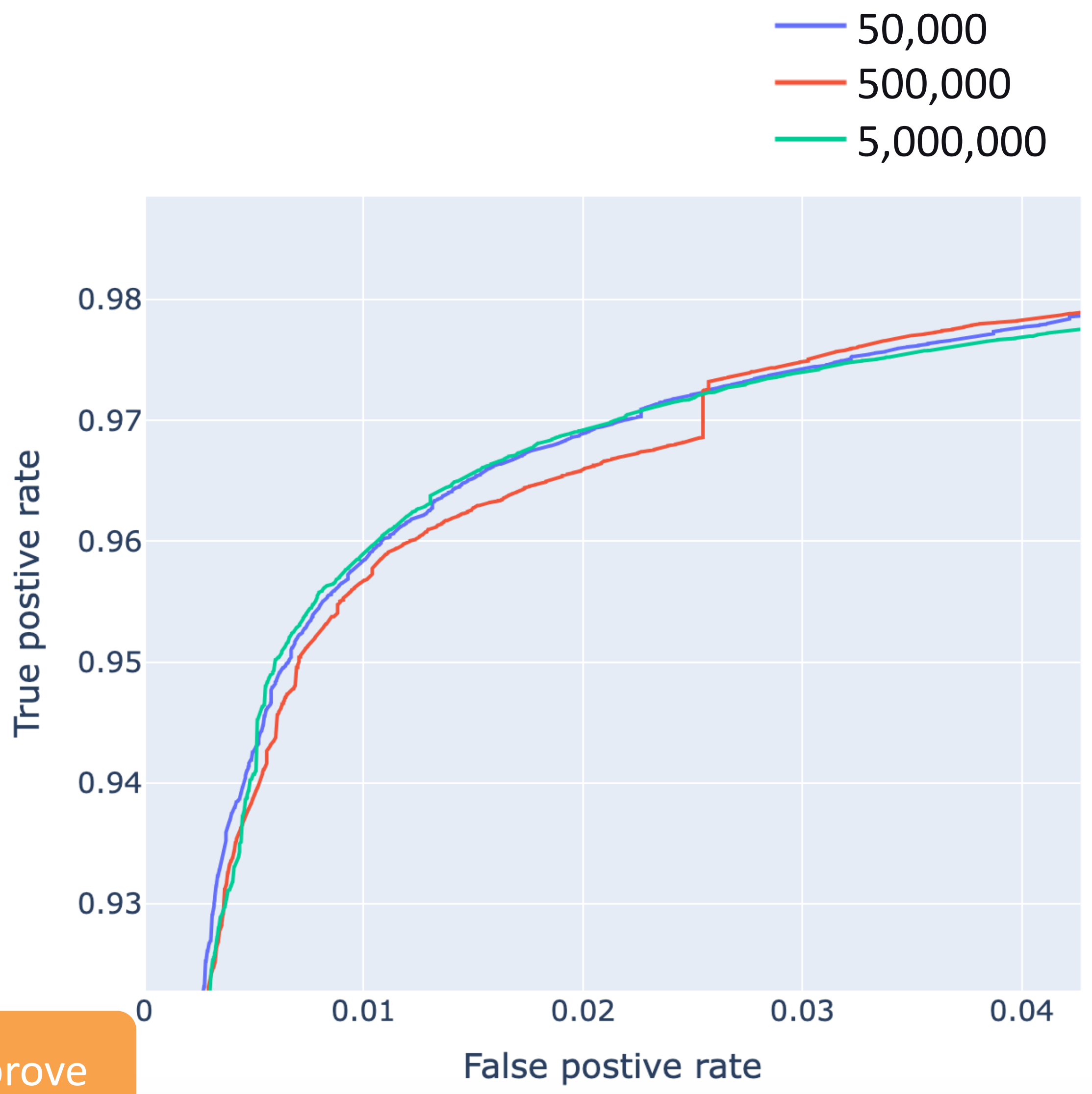
Static + Emulation gives you a slight performance increase over just static features. Longer emulation runs don't necessarily improve things!

Goodware/Malware Task

All of EMBER 2018

Maximum instructions	Median emulation time (s)	AUROC
50,000 + Static	0.96	.9954
500,000 + Static	1.40	.9953
5,000,000 + Static	1.82	.9951
Only Static features	-	.9951

AUROC does not improve

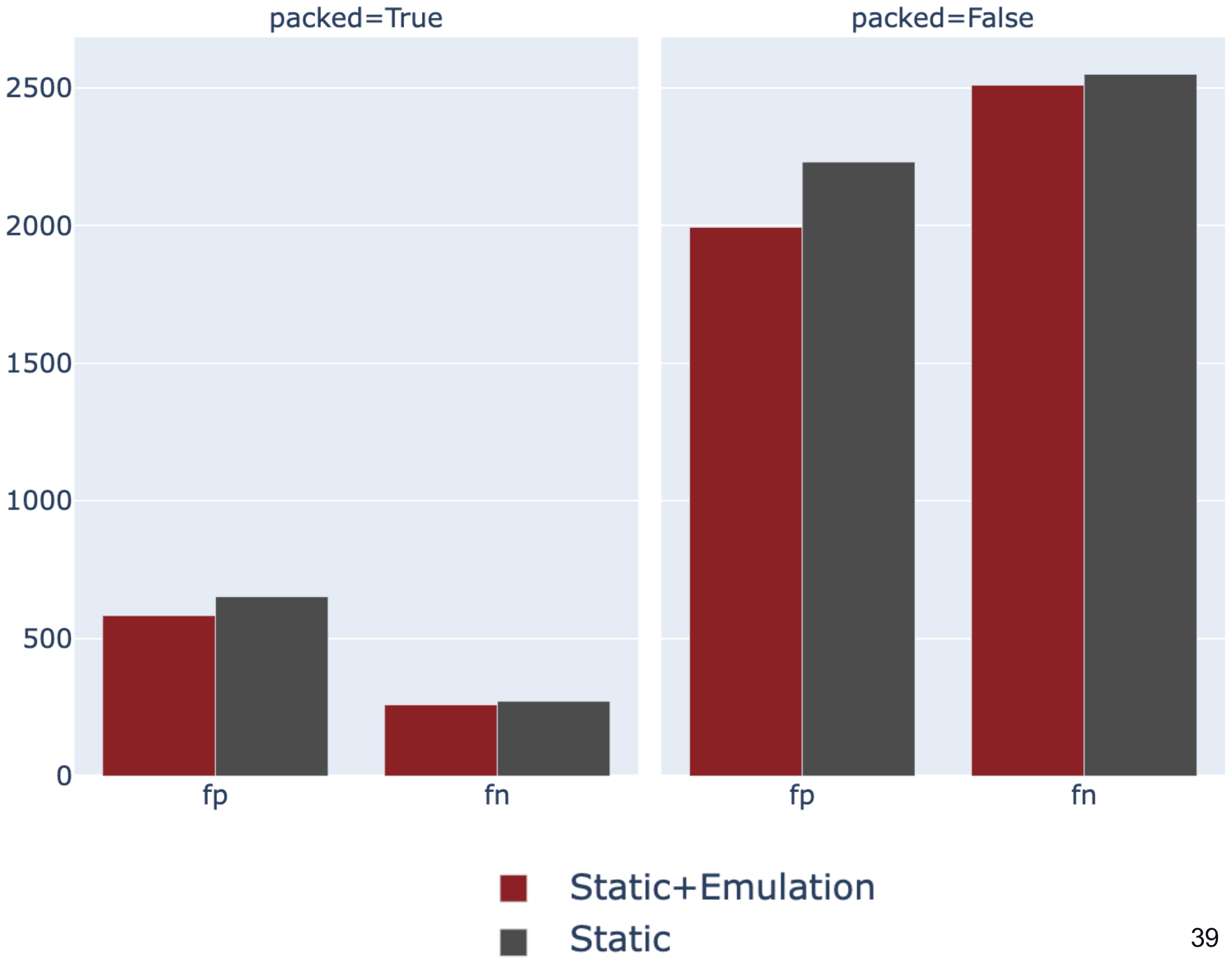


Static + Emulation gives you a slight performance increase over just static features. Longer emulation runs don't necessarily improve things!

Where are the Improvements Coming From?

How are we getting lift from short emulation runs?

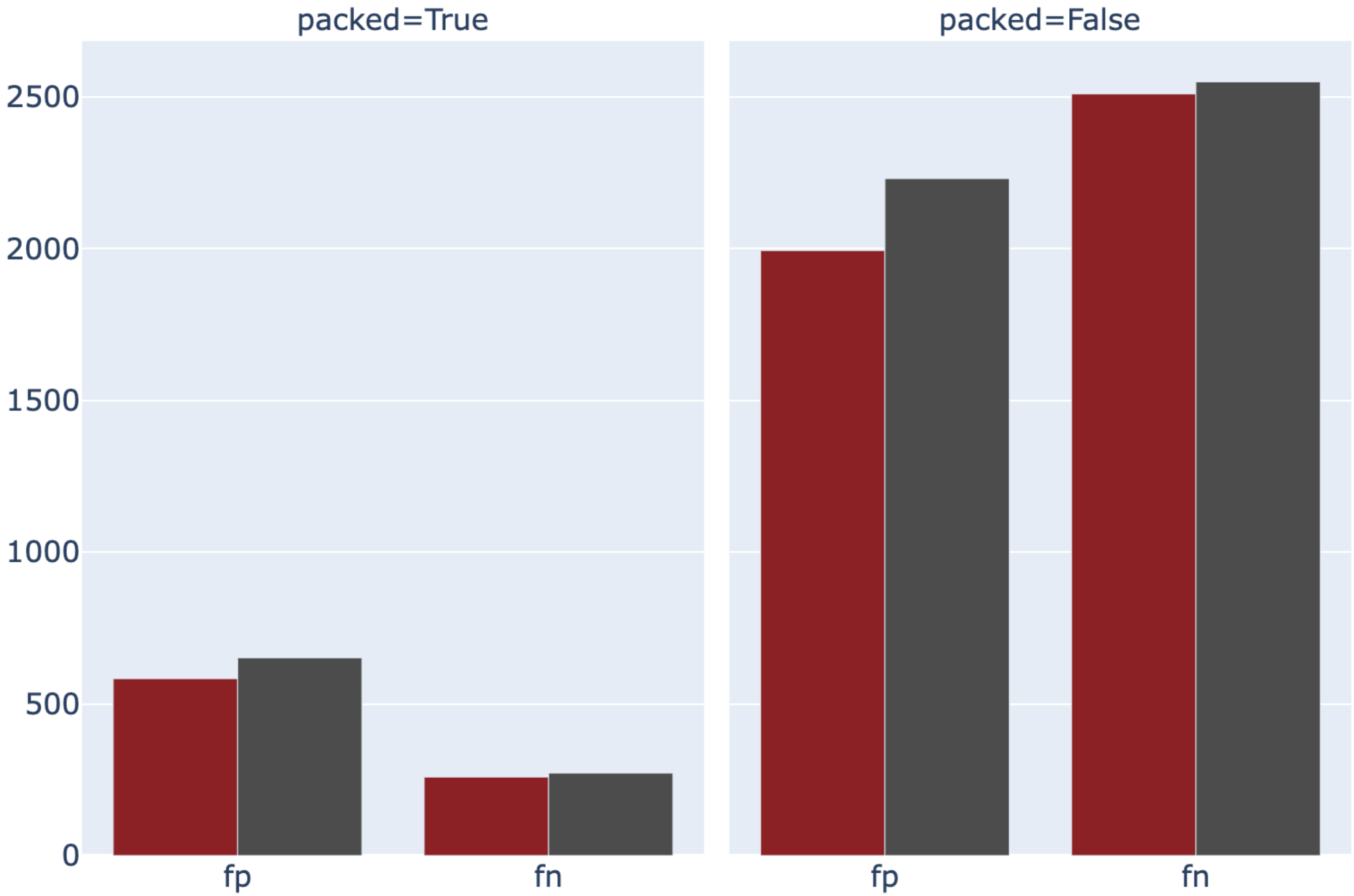
- Errors from the 5 million instruction emulation run
- Packed was determined by Detect-It-Easy
- Most improvements are on goodware
  - Specifically unpacked goodware



Where are the Improvements Coming From?

How are we getting lift from short emulation runs?

- Errors from the 5 million instruction emulation run
- Packed was determined by Detect-It-Easy
- Most improvements are on goodware
  - Specifically unpacked goodware



Short emulation runs provide additional goodware signal in combination with static features

Malware Family Prediction

Maximum instructions	Median emulation time (s)	Accuracy	Macro F1
50,000 + Static	0.96	.93	.87
500,000 + Static	1.40	.94	.88
5,000,000 + Static	1.82	.94	.88
Static	-	.92	.86

- Top 19 families (AVCLASS) in EMBER 2018 present in both train and test
- Slight improvements with emulation length

Predictions

True labels	cerber	emotet	wannacry	ursnif	chapak	ramnit	lethic	sdbot	sality	fareit	wapomi	sivis	downloadguide	startsurf	prepscram	zbot	xtrat	virlock	kovter	qbot
qbot	0	76	0	45	0	3	0	1	0	0	0	0	0	95	0	122	0	0	0	474
kovter	4	1	0	0	0	5	0	0	0	0	0	0	0	0	0	19	0	0	1004	0
virlock	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	604	0	0
xtrat	0	0	0	1	0	5	0	0	7	22	1	0	0	2	0	102	19140	0	0	0
zbot	23	8	0	28	19	35	1	13	50	45	4	0	0	15	0	13033	9	0	10	0
prepscram	0	0	0	0	255	0	0	0	0	0	0	0	0	322	316	0	0	0	0	0
startsurf	0	0	0	0	38	0	0	0	0	0	0	0	0	2197	130	0	0	0	0	0
downloadguide	0	0	0	0	0	0	0	0	0	0	0	0	2640	0	0	0	0	0	0	0
sivis	0	0	0	0	0	19	0	0	6	0	0	2775	0	0	0	3	0	0	0	0
wapomi	0	0	0	3	0	104	0	0	28	2	777	0	0	0	0	41	0	0	0	0
fareit	2	0	0	4	0	2	0	0	0	657	0	0	0	0	0	15	1	0	0	0
sality	0	0	1	1	0	83	1	40	8838	4	30	0	0	0	1	47	4	0	0	0
sdbot	0	0	0	0	1	3	0	1088	26	0	1	0	0	0	0	24	13	0	0	0
lethic	0	0	0	1	0	1	2027	0	2	4	0	0	0	0	0	24	10	0	0	0
ramnit	2	11	0	7	1	10032	1	2	70	1	81	0	0	1	3	108	0	0	0	0
chapak	0	0	0	2	251	16	0	0	0	0	0	0	0	286	201	14	0	0	0	0
ursnif	28	457	0	3403	30	23	21	0	1	7	112	0	0	0	0	1624	0	26	1	0
wannacry	0	0	2044	0	1	1	0	0	3	2	0	0	0	0	0	15	0	0	0	0
emotet	2	4754	0	33	1	19	0	0	10	0	0	0	0	2	0	77	0	0	4	2
cerber	697	1	0	14	0	11	0	0	1	2	0	0	0	1	0	34	0	0	0	0

Test set predictions for the 50,000 + static model

Malware Family Prediction

Maximum instructions	Median emulation time (s)	Accuracy	Macro F1
50,000 + Static	0.96	.93	.87
500,000 + Static	1.40	.94	.88
5,000,000 + Static	1.82	.94	.88
Static	-	.92	.86

- Top 19 families (AVCLASS) in EMBER 2018 present in both train and test
- Slight improvements with emulation length

Predictions

True labels	cerber	emotet	wannacry	ursnif	chapak	ramnit	lethic	sdbot	sality	fareit	wapomi	sivis	downloadguide	startsurf	prepscram	zbot	xtrat	virlock	kovter	qbot
qbot	0	76	0	45	0	3	0	1	0	0	0	0	0	95	0	122	0	0	0	474
kovter	4	1	0	0	0	5	0	0	0	0	0	0	0	0	0	19	0	0	1004	0
virlock	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	604	0	0
xtrat	0	0	0	1	0	5	0	0	7	22	1	0	0	2	0	102	19140	0	0	0
zbot	23	8	0	28	19	35	1	13	50	45	4	0	0	15	0	13033	9	0	10	0
prepscram	0	0	0	0	255	0	0	0	0	0	0	0	0	322	316	0	0	0	0	0
startsurf	0	0	0	0	38	0	0	0	0	0	0	0	0	2197	130	0	0	0	0	0
downloadguide	0	0	0	0	0	0	0	0	0	0	0	0	2640	0	0	0	0	0	0	0
sivis	0	0	0	0	0	19	0	0	6	0	0	2775	0	0	0	3	0	0	0	0
wapomi	0	0	0	3	0	104	0	0	28	2	777	0	0	0	0	41	0	0	0	0
fareit	2	0	0	4	0	2	0	0	0	657	0	0	0	0	0	15	1	0	0	0
sality	0	0	1	1	0	83	1	40	8838	4	30	0	0	0	1	47	4	0	0	0
sdbot	0	0	0	0	1	3	0	1088	26	0	1	0	0	0	0	24	13	0	0	0
lethic	0	0	0	1	0	1	2027	0	2	4	0	0	0	0	0	24	10	0	0	0
ramnit	2	11	0	7	1	10032	1	2	70	1	81	0	0	1	3	108	0	0	0	0
chapak	0	0	0	2	251	16	0	0	0	0	0	0	0	286	201	14	0	0	0	0
ursnif	28	457	0	3403	30	23	21	0	1	7	112	0	0	0	0	1624	0	26	1	0
wannacry	0	0	2044	0	1	1	0	0	3	2	0	0	0	0	0	15	0	0	0	0
emotet	2	4754	0	33	1	19	0	0	10	0	0	0	0	2	0	77	0	0	4	2
cerber	697	1	0	14	0	11	0	0	1	2	0	0	0	1	0	34	0	0	0	0

Test set predictions for the 50,000 + static model

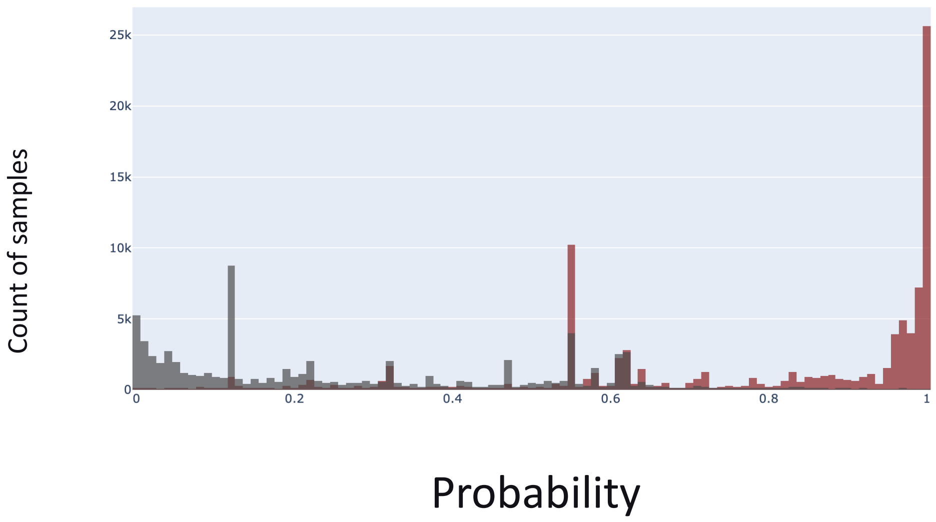
Confusion Matrix is not Symmetric

Conclusion

- More emulation data is better than high-fidelity emulation data
  - Clear benefits for even simple approaches to mocking API calls
- Emulation features provide clear lift over static-only features
  - Both goodware/malware and family classification tasks improve
- Surprisingly even short emulation runs help
  - Provides additional, high-quality goodware signal

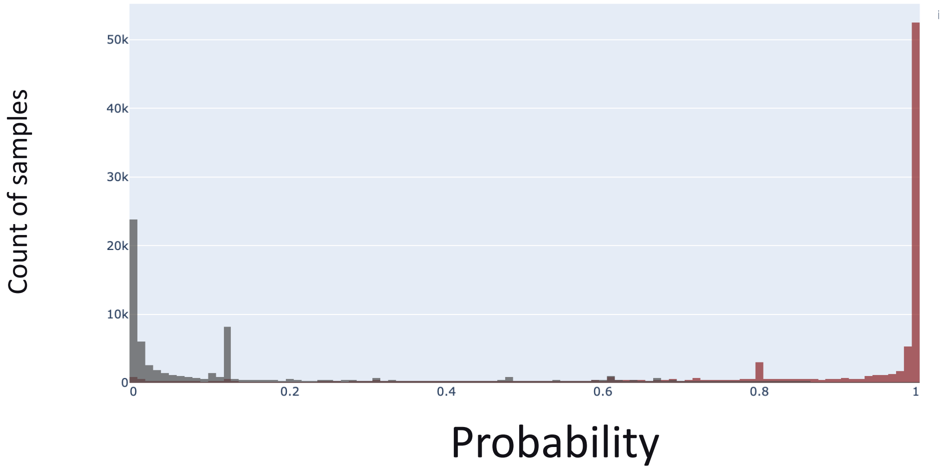
Model score distribution (initial)

Old



Improved

Model score distribution (best model)



MANDIANT

YOUR CYBERSECURITY ADVANTAGE

Thank You.

MANDIANT

YOUR CYBERSECURITY ADVANTAGE