An LSTM-Based Method for Section Boundary Detection in Firmware Analysis

Riccardo Remigio, Alessandro Bertani, Mario Polino,
Michele Carminati, Stefano Zanero

OWASP Italy Day 2023

Politecnico of Milan - 11th September 2023

Agenda

- Background
- Problem Statement
- Approach
- Experimental Evaluation
- Future Work and Limitations
- Conclusion

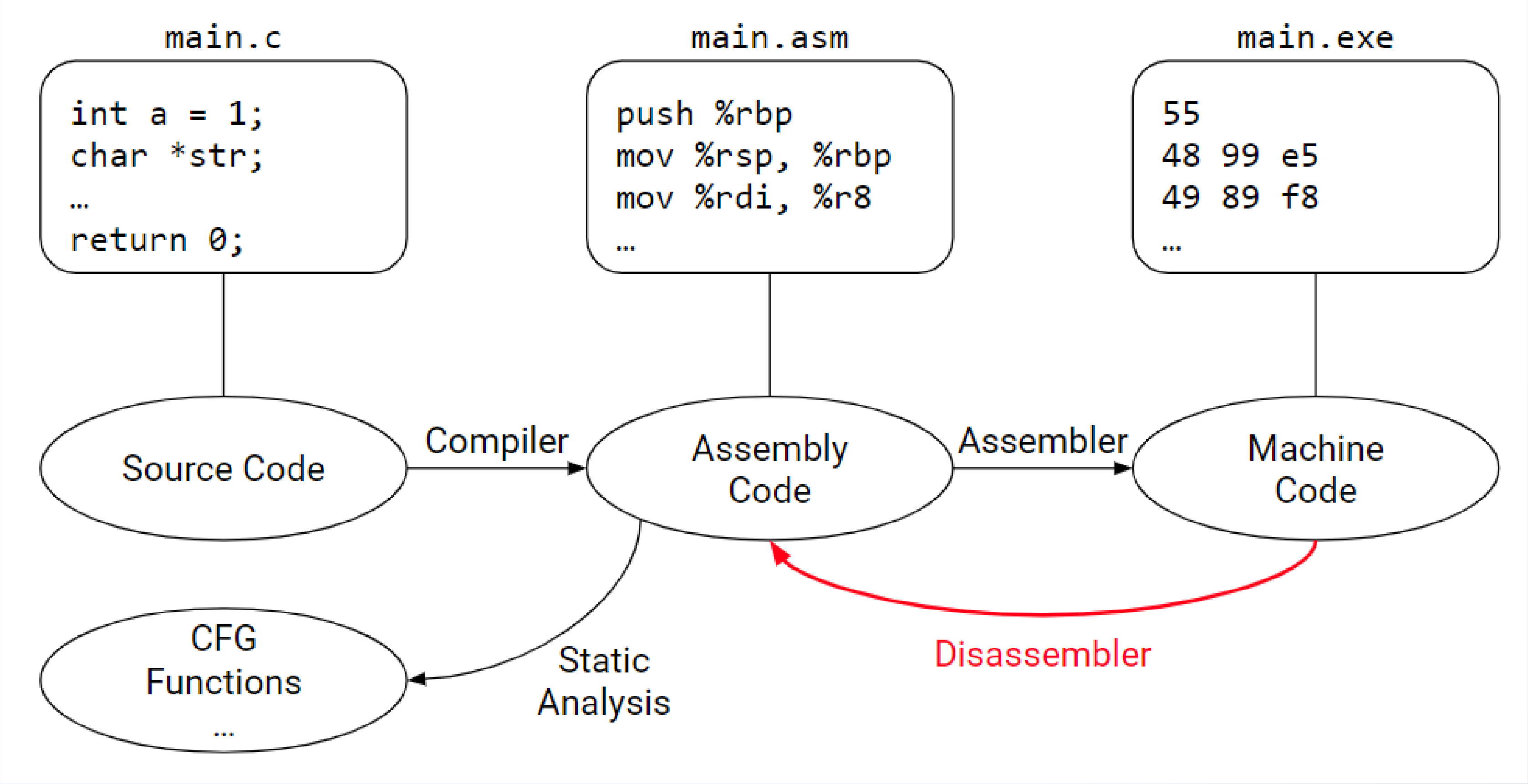
Open: Everything at OWASP is radically transparent from our finances to our code.

Innovative: We encourage and support innovation and experiments for solutions to software security challenges.

Global: Anyone around the world is encouraged to participate in the OWASP community.

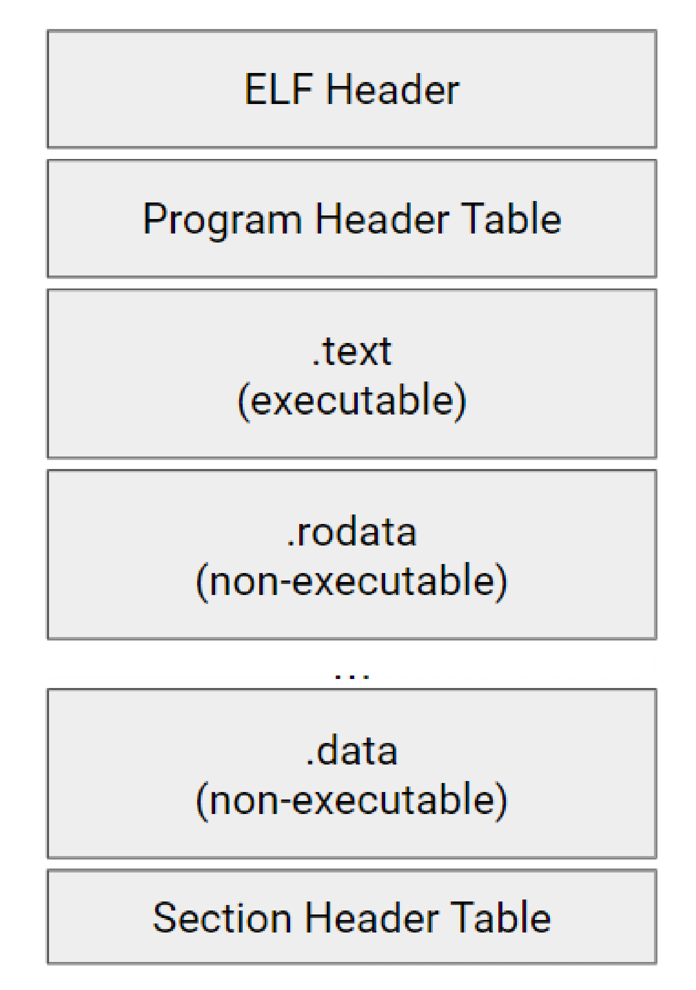
Integrity: Our community is respectful, supportive, truthful, and vendor neutral

Static Analysis and Disassembly



Executable Files

- A binary file contains executable and non-executable sections.
- The binary may be **header-less**
- The disassembler does not know where the code section is.
- **Disassembled data = garbage!**



Inlined Data

```
9214: add r3, pc, r3
9218: ldr r2, [r3, r2]
921c: cmp r2, #0
9220: bxeq lr
9224: b 9050
9228: andeq pc, r0, r8, lsr #3
922c: andeq r0, r0, r8, lsl r1
```

```
9214: add r3, pc, r3
9218: ldr r2, [r3, r2]
921c: cmp r2, #0
9220: bxeq lr
9224: b 9050
9228: .word 0x0000f1a8
922c: .word 0x00000118
```

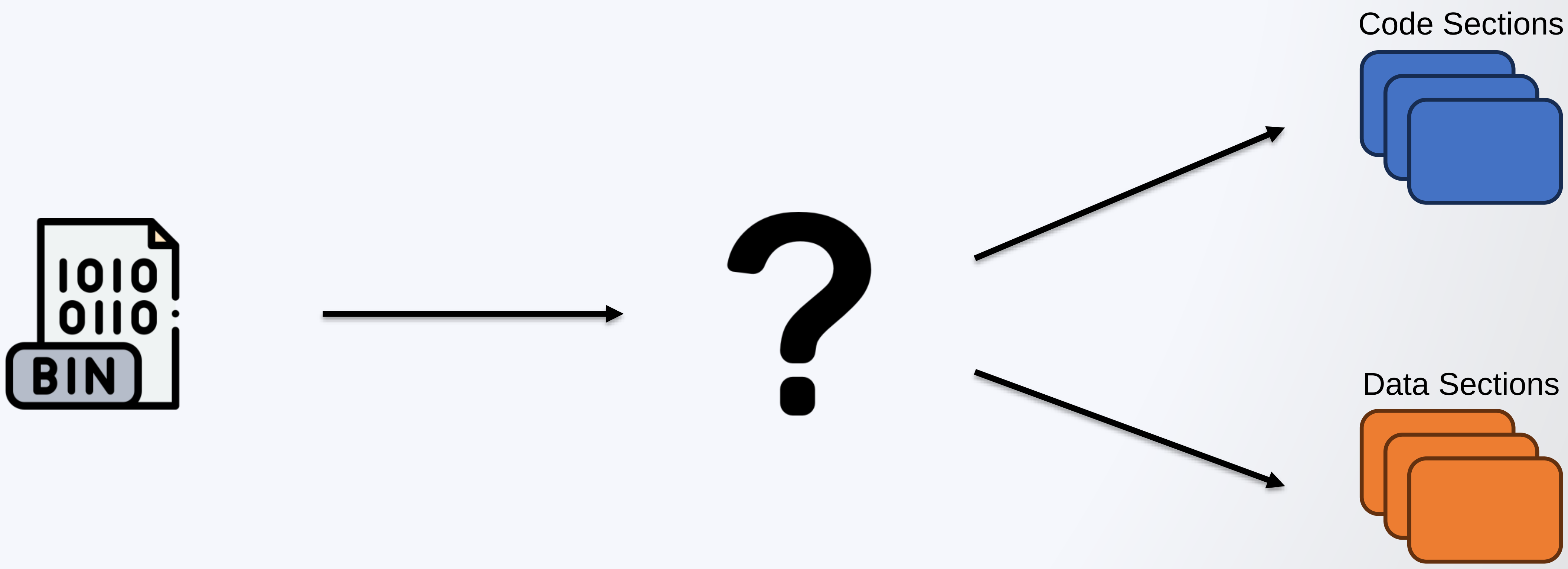
Inlined Data

Disassembly starts at offset:

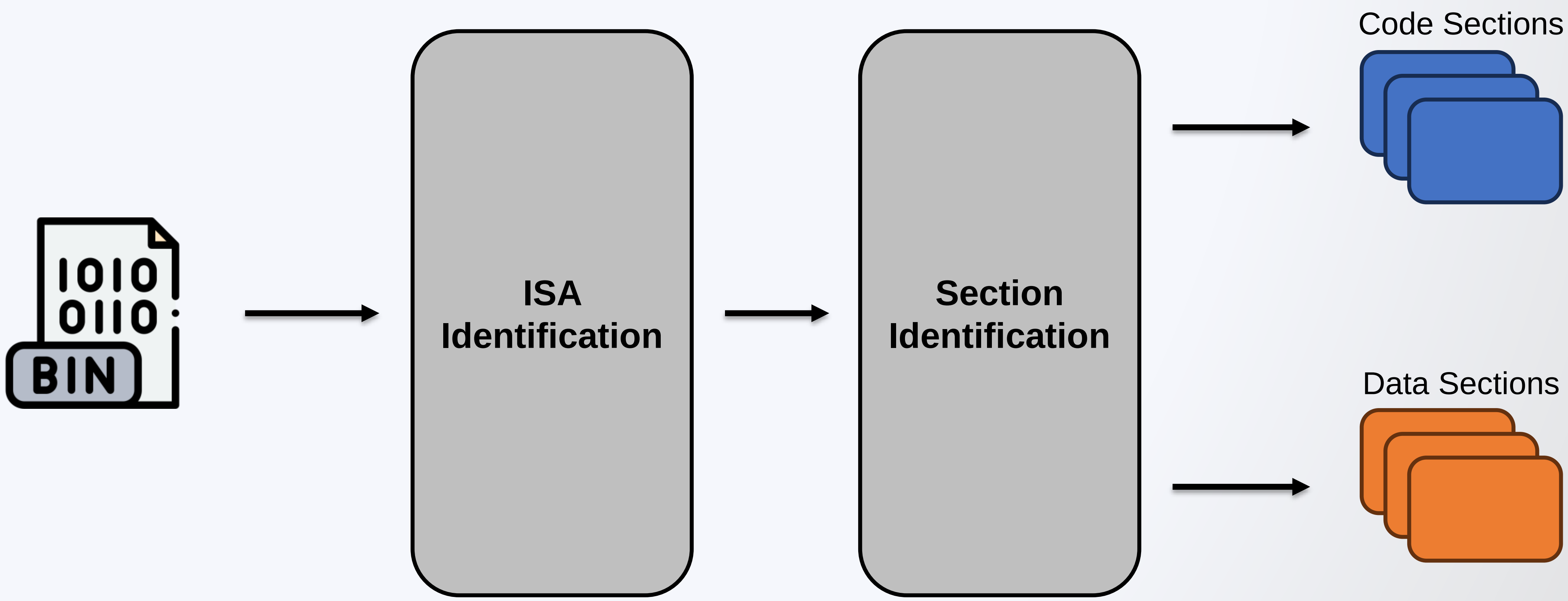
<table>
<tr><th>Binary Code Bytes</th><th>0</th><th>+1</th><th>+5</th></tr>
<tr><td>14</td><td rowspan="2">add 14,esp</td><td>Data (1 byte)</td><td rowspan="4">Data (4 bytes)</td></tr>
<tr><td>53</td><td>push ebx</td></tr>
<tr><td>83</td><td rowspan="3">sub c,esp</td><td rowspan="3">sub c,esp</td></tr>
<tr><td>ec</td></tr>
<tr><td>0c</td><td rowspan="2">or 8b,al</td></tr>
<tr><td>8b</td><td rowspan="4">mov 14(esp), ebx</td><td rowspan="4">mov 14(esp), ebx</td></tr>
<tr><td>5c</td><td>pop esp</td></tr>
<tr><td>24</td><td rowspan="2">and 14,al</td></tr>
<tr><td>14</td></tr>
<tr><td>8b</td><td rowspan="3">mov 4(ebx), edx</td><td rowspan="3">mov 4(ebx), edx</td><td rowspan="3">mov 4(ebx), edx</td></tr>
<tr><td>53</td></tr>
<tr><td>04</td></tr>
<tr><td>8b</td><td rowspan="2">mov (ebx),ecx</td><td rowspan="2">mov (ebx),ecx</td><td rowspan="2">mov (ebx),ecx</td></tr>
<tr><td>0b</td></tr>
</table>

Disassembled Instruction Sequences

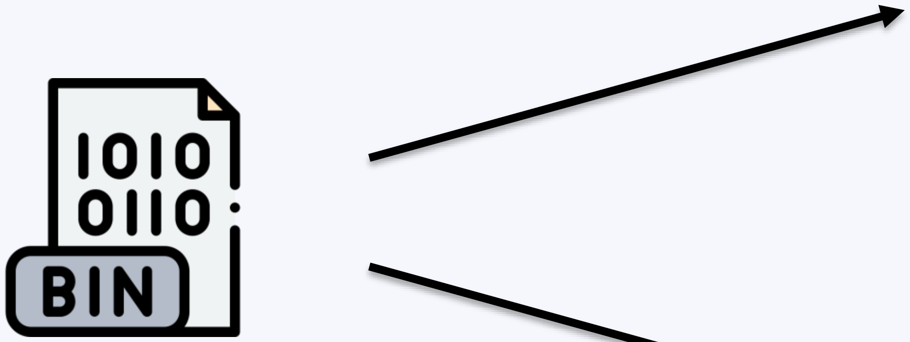
The problem we want to solve



How we solve it



ISA Identification: Goal

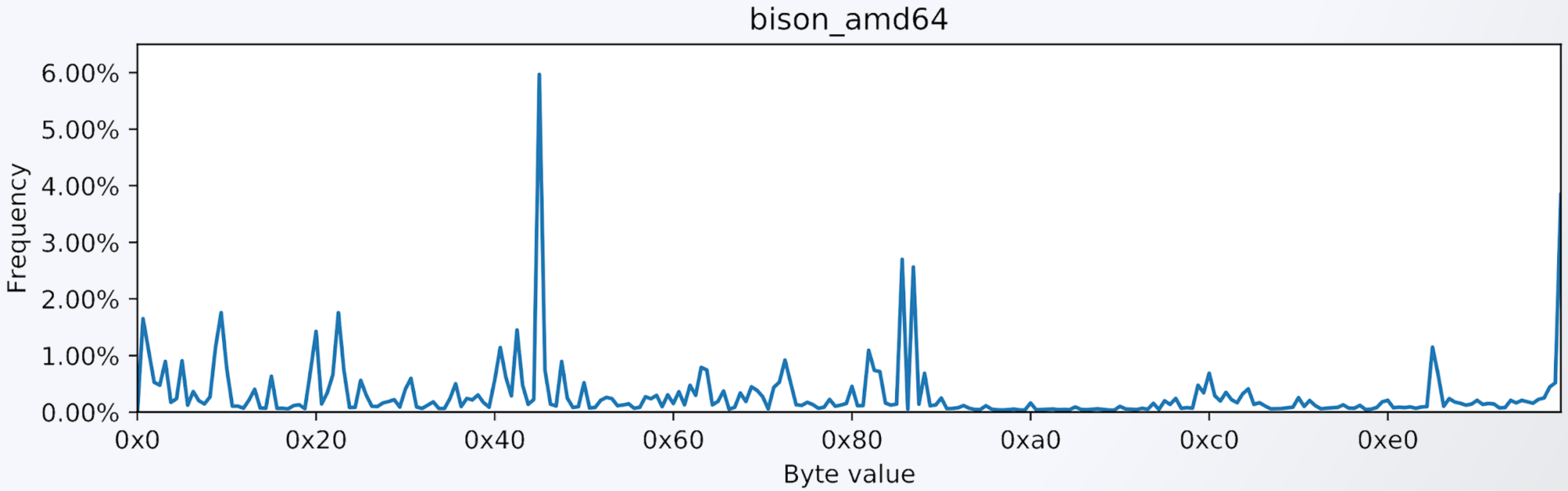


Instruction Set Architecture (ISA)

Endianness

ISA Identification: Approach

- The approach is from (De Nicolao et al., 2018)
- Idea: the Byte Frequency Distribution characterizes a specific ISA
- We extended it to classify packed binaries



ISA Identification: Model

Multiclass logistic regression. Features:

- Byte Frequency Distribution (BFD)

$$f_i = \frac{count(i)}{\sum_{j=0}^{255} count(j)} \qquad \forall i \in [0, 255]$$

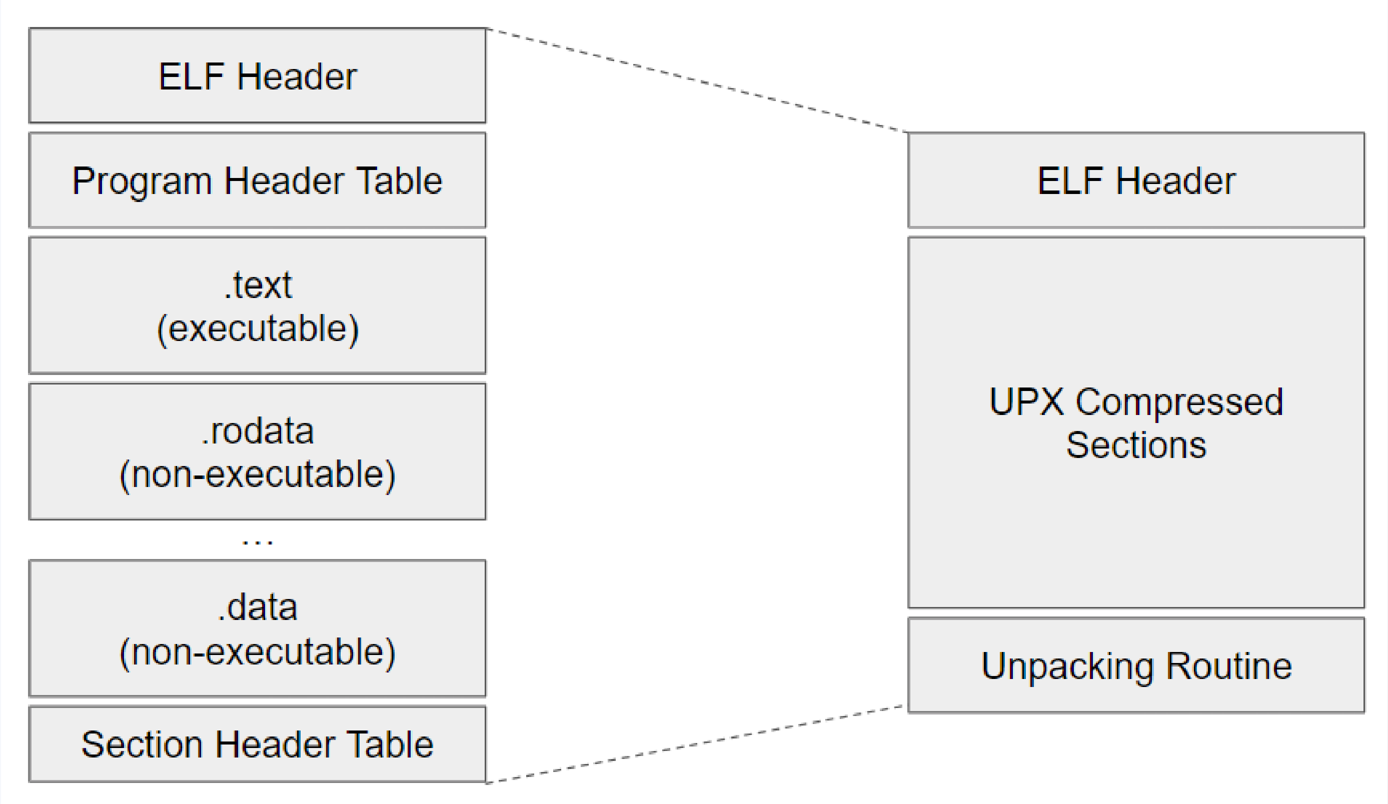
- Bi-gram frequencies of the endianness marker

$$\text{0x0001} \qquad \text{0x0010} \qquad \text{0xFFFE} \qquad \text{0xFEFF}$$

- Patterns of known function prologues/epilogues (angr's archinfo)

$$freq(pattern) = \frac{matches(pattern, file)}{len(file)}$$

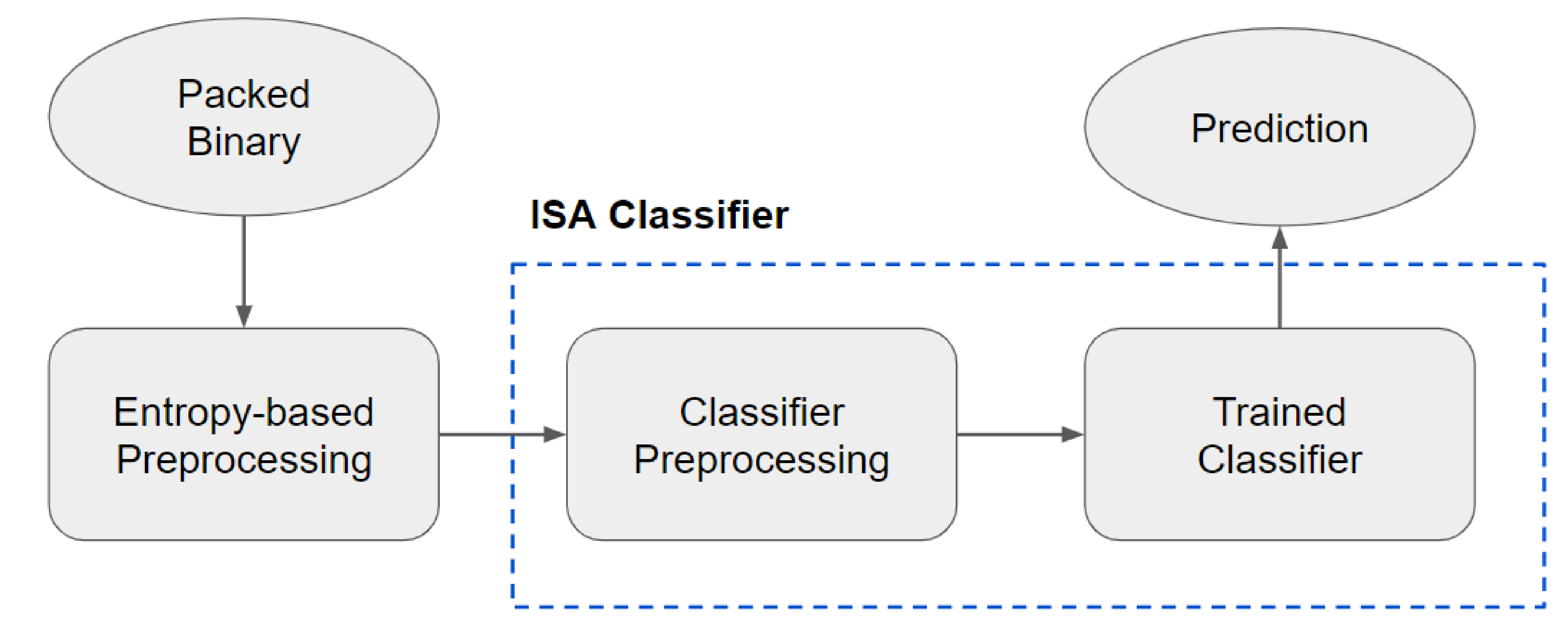
ISA Identification: Packed Binaries



ISA Identification: Packed Binaries

- **Assumption**: the compression routine alters the BFD of the binary. This would fool the classifier.
- **Solution**: detect compressed sections and extract the BFD only from uncompressed bytes.
- **Approach**: compute entropy of 256-bytes blocks, delete a block if its entropy exceeds a certain value.
- Empirically chosen value: 6.3.

ISA Identification: Full Pipeline



Section Identification: Goal

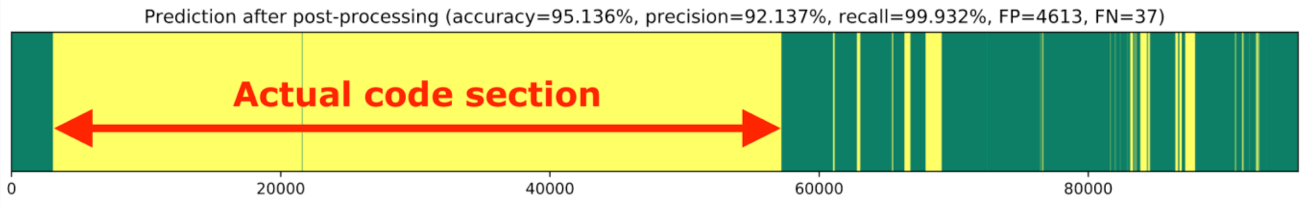
1. Learn a model from a training sequence of labeled bytes

...	01	00	7d	02	c3	48	83	c6	68	49	83	...
...	D	D	D	C	C	C	C	C	C	D	D	...

C: code D: data

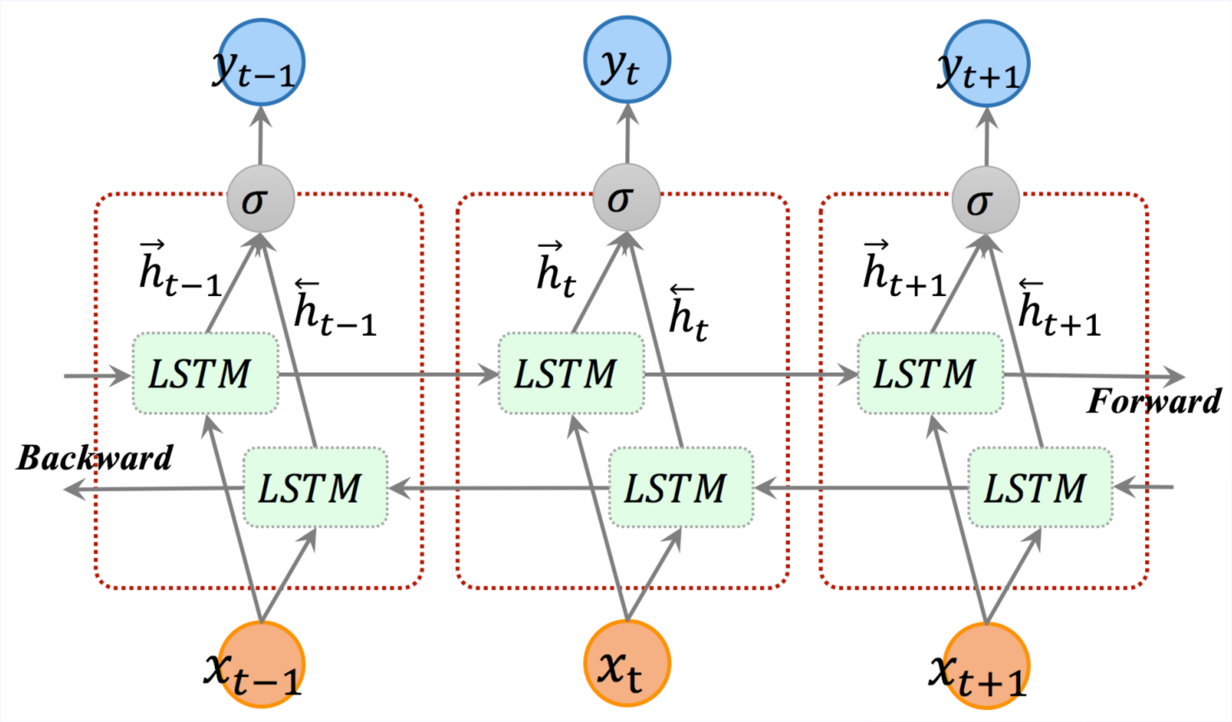
2. Use the model to classify the bytes of an unlabeled file

3. Apply postprocessing to reduce noise



Section Identification: Model

The model we use is a Bidirectional LSTM:



Section Identification: Preprocessing

- Each byte is represented as one-hot encoding: a vector of size 256 where the i-th element is 1 if the value of the byte is i.
- **Training phase**:
  - Choose a random binary
  - Choose a random point inside the binary
  - Take a sequence (length depends on the structure of the model)
  - Repeat until the whole dataset has been sampled
- **Prediction**:
  - Apply one-hot encoding as above
  - Divide the binary into sequences of fixed length
  - Use them as inputs to the model

Section Identification: Training

- We use a Cross-Entropy loss function:

$$Loss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)))$$

- We use the Adam optimizer and train for a total of 5 epochs per model.

- We tested different values for the input and output dimensions of the model:

Table 8: Hyperparamters values for LSTM model

Architecture	amd64	arm32	arm64	armel	i386	mips	mips64	mipsel	powerpc
Input dimension	25	70	75	100	25	50	50	50	50
LSTM output dimension	24	24	24	24	32	32	32	16	32

Section Identification: Postprocessing

We use three different techniques:

- The one used by the authors of ELISA
- The one applied by the authors of Byteweight
- A third technique based on the frequency of the instructions

Section Identification: Postprocessing (1)

The authors of ELISA apply postprocessing to reduce noise.

Noise: small fragments of code or data

1. Invert the label of the smallest subsequence of code or data
2. Repeat until:
   - Fewer than k sections remain, or
   - Size of the smallest segment / size of the biggest segment > threshold

Section Identification: Postprocessing (2)

The authors of *Byteweight* use this postprocessing technique to recognize the prologue of functions inside a binary.

1. If the beginning of a predicted section is a function prologue $\rightarrow$ **correct**
2. Otherwise, we search for a function prologue around the start of the predicted section and adjust it.

Usually, the offset between the predicted section and the start of a function prologue is **less than 6 bytes**.

Section Identification: Postprocessing (3)

The last technique is based on instruction frequency.

We construct a dictionary of instruction frequencies taken from the training set.

Then we take groups of four bytes, and disassemble them:

1. If disassembling with an offset gives instructions with low frequency, the prediction is **correct**
2. Otherwise, we take the instruction with the higher frequency and adjust the start of the section accordingly

Experimental Validation: Dataset

Architecture	BW	DEB	DEBP	FW	FWP
amd64	588	385	277	971	75
arm32	572	-	-	275	75
arm64	572	382	-	496	19
armel	531	385	237	1000	762
armhf	-	385	192	-	-
i386	440	385	249	422	181
mips	572	384	255	795	398
mips64	572	-	-	482	-
mipsel	572	384	257	983	567
powerpc	572	-	-	934	282
ppc64el	-	380	278	-	-

Experimental Validation: Metrics

Classic machine learning metrics:

- Precision

$$\frac{True\ Positives}{True\ Positives + False\ Positives}$$

- Recall

$$\frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- Accuracy

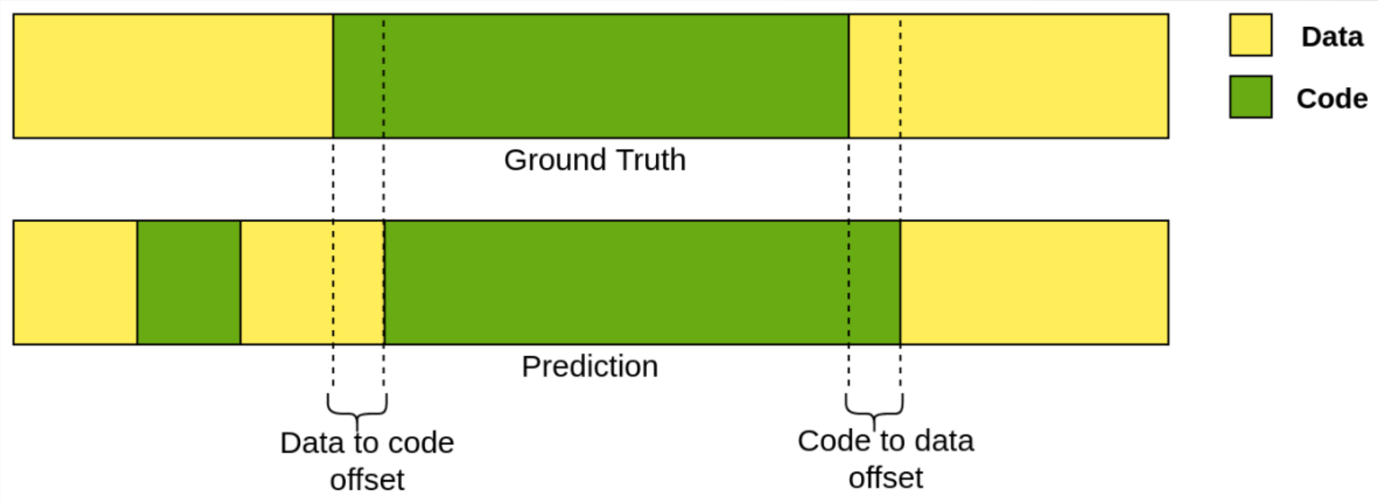
$$\frac{True\ Positives + True\ Negatives}{TP + TN + FP + FN}$$

- F1-Score

$$2 \times \frac{Precision\ \times Recall}{Precision + Recall}$$

Experimental Validation: New Metrics

- Code to data offset
- Data to code offset
- Wrong sections



Exp. Validation: Architecture Classification

Class	ELISA			Our approach			Samples
	P	R	F1	P	R	F1	
amd64	1.000	0.018	0.035	0.829	0.776	0.830	277
armel	0.000	0.000	0.000	0.585	0.726	0.648	237
armhf	0.124	1.000	0.221	0.159	0.073	0.100	192
i386	1.000	0.076	0.142	0.830	0.763	0.795	249
mips	1.000	0.455	0.625	0.740	0.949	0.832	255
mipsel	1.000	0.027	0.053	0.903	0.942	0.922	257
ppc64el	1.000	0.194	0.325	0.869	0.932	0.899	278
Total	0.732	0.253	0.200	0.711	0.737	0.718	1745

Exp. Validation: Architecture Classification

Class	ELISA			Our approach			Samples
	P	R	F1	P	R	F1	
amd64	1.000	0.520	0.684	0.881	0.787	0.831	75
arm32	0.000	0.000	0.000	0.951	0.773	0.853	75
arm64	1.000	0.158	0.273	1.000	0.789	0.882	19
armel	0.730	0.298	0.423	0.814	0.987	0.892	762
i386	1.000	0.387	0.558	1.000	0.751	0.858	181
mips	0.227	0.997	0.370	0.935	0.942	0.939	398
mipsel	0.695	0.229	0.345	0.974	0.850	0.980	567
powerpc	0.000	0.000	0.000	1.000	0.922	0.959	282
Total	0.582	0.324	0.332	0.944	0.850	0.890	2359

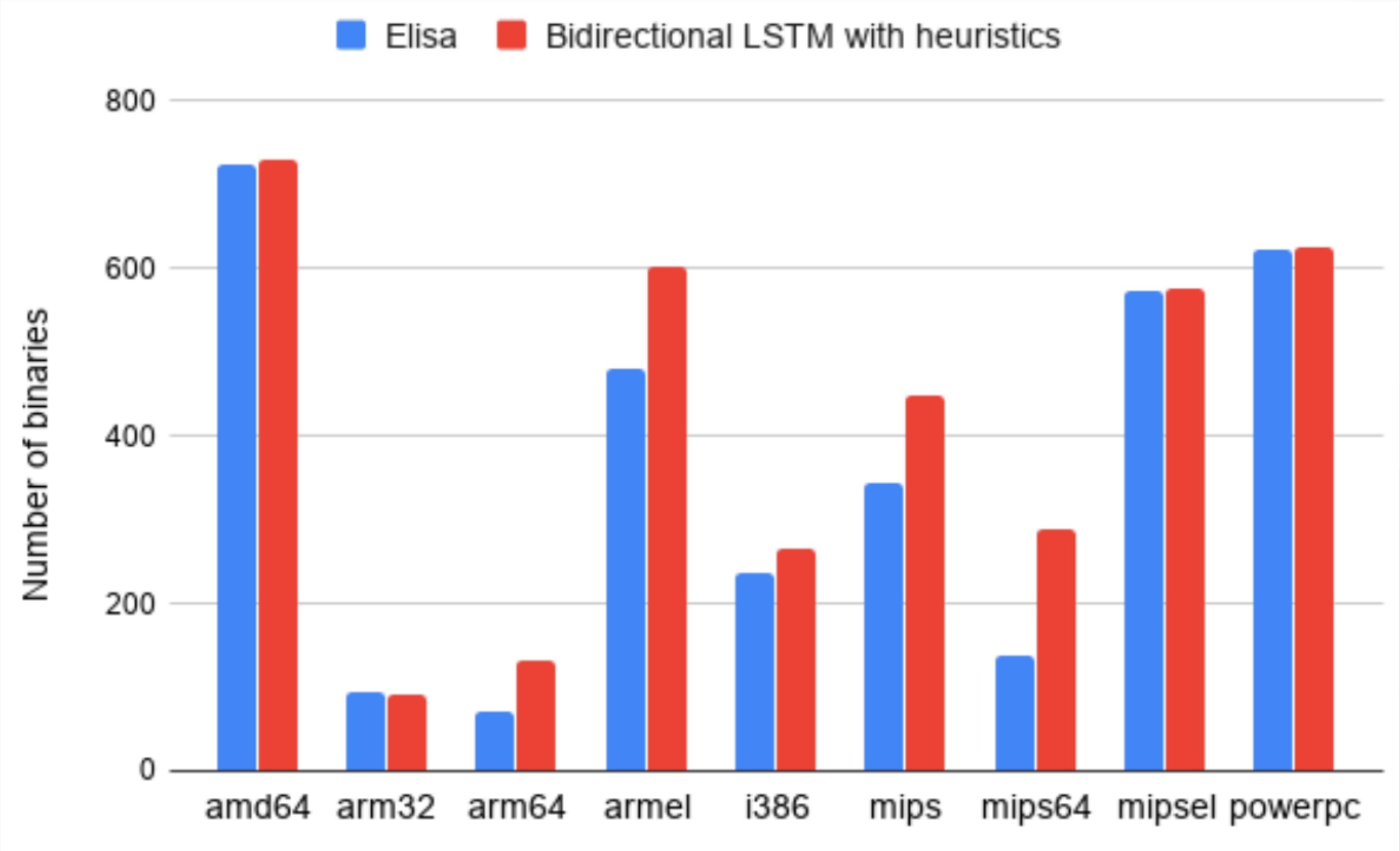
Exp. Validation: Section Identification

Architecture	ELISA			ELISA + H			Bi-LSTM			Bi-LSTM + H		
	CD	DC	WS	CD	DC	WS	CD	DC	WS	CD	DC	WS
amd64	85.0%	61.1%	10.7%	85.0%	63.9%	10.7%	85.8%	90.6%	6.3%	85.8%	91.0%	6.3%
arm32	60.7%	88.4%	18.1%	60.7%	94.8%	18.1%	58.7%	96.1%	14.8%	58.7%	96.1%	14.8%
arm64	18.9%	40.4%	48.9%	18.9%	45.5%	48.9%	35.4%	77.9%	39.1%	35.4%	77.9%	39.1%
armel	54.6%	90.3%	25.3%	54.6%	92.5%	25.3%	68.3%	86.5%	25.2%	68.3%	93.8%	25.2%
i386	78.2%	62.9%	41.4%	78.2%	63.3%	41.4%	88.1%	85.1%	38.7%	88.1%	90.4%	38.7%
mips	50.8%	**16.0%**	53.6%	50.8%	**72.2%**	53.6%	66.5%	63.9%	52.3%	66.5%	**74.2%**	52.3%
mips64	38.3%	**10.6%**	38.6%	38.3%	**81.6%**	38.6%	80.5%	74.6%	36.6%	80.5%	**84.6%**	36.6%
mipsel	66.1%	**8.4%**	59.9%	66.1%	**40.5%**	59.9%	66.5%	29.2%	59.9%	66.5%	**47.9%**	59.9%
powerpc	76.5%	53.4%	50.7%	76.5%	60.1%	50.7%	77.0%	57.9%	49.4%	77.0%	64.9%	49.4%

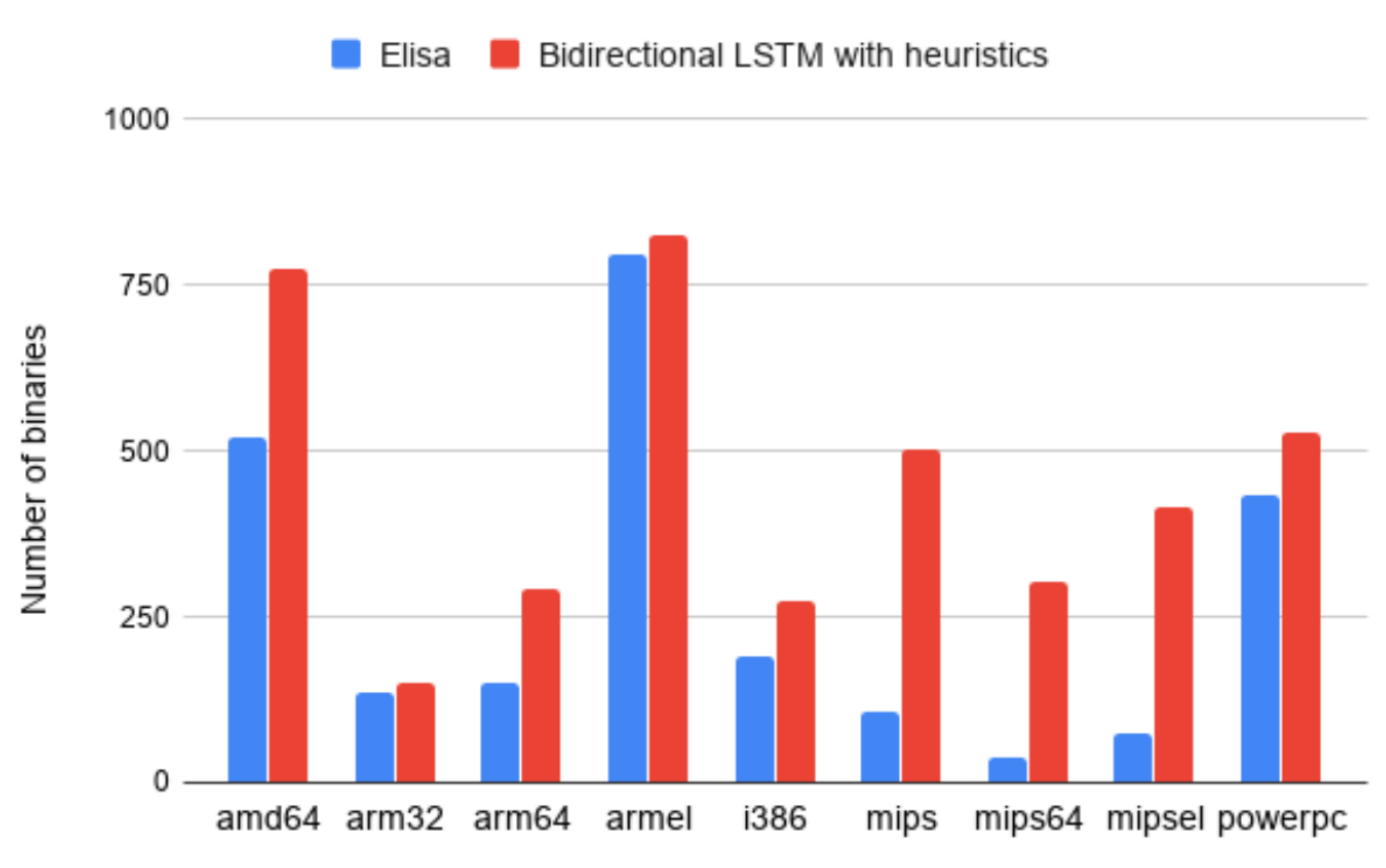
Exp. Validation: Section Identification

Architecture	amd64	arm32	arm64	armel	i386	mips	mips64	mipsel	powerpc
CD	+0.8%	-1.9%	+16.5%	+13.8%	+9.9%	+15.7%	+42.2%	+0.5%	+0.5%
DC	+29.9%	+7.7%	+38.0%	+3.4%	+27.5%	+58.2%	+74.0%	+39.5%	+11.4%
WS	-4.3%	-3.2%	-9.8%	-0.1%	-2.7%	-1.3%	-2.0%	0%	-1.4%

Exp. Validation: Code to Data Offset



Exp. Validation: Data to Code Offset



Limitations and Future Work

Limitations:

- An attacker could artificially alter the BFD of the binary to fool the architecture classifier

Future work:

- Possible implementation of heuristics that eliminate wrongly predicted sections
- Exploration of other models

Conclusions

- We implement our approach for architecture classification, extending an existing work, ELISA.
- We implement a novel approach for section identification based on a Bidirectional LSTM model.
- We provide three metrics that can be used to better measure the results of existing models.
- We prove that with both traditional and new metrics, our approach performs better with respect to the state of the art.

Thank you to our sponsors

Contrast Security

Qualys.

SecureFlag

IMQ group

OWASP 2023

ITALY DAY