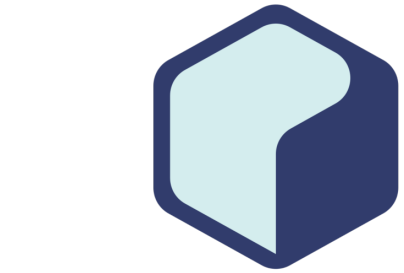
UVScan: Detecting Third-Party Component Usage Violations in IoT Firmware

Binbin Zhao[1,2], Shouling Ji[2], Xuhong Zhang[2], Yuan Tian[3], Qinying Wang[2], Yuwen Pu[2], Chenyang Lyu[2], Raheem Beyah[1]

[1]Georgia Tech [2]Zhejiang University [3]UCLA

Background

- Third-party components are widely used in IoT firmware to shorten the development cycle.

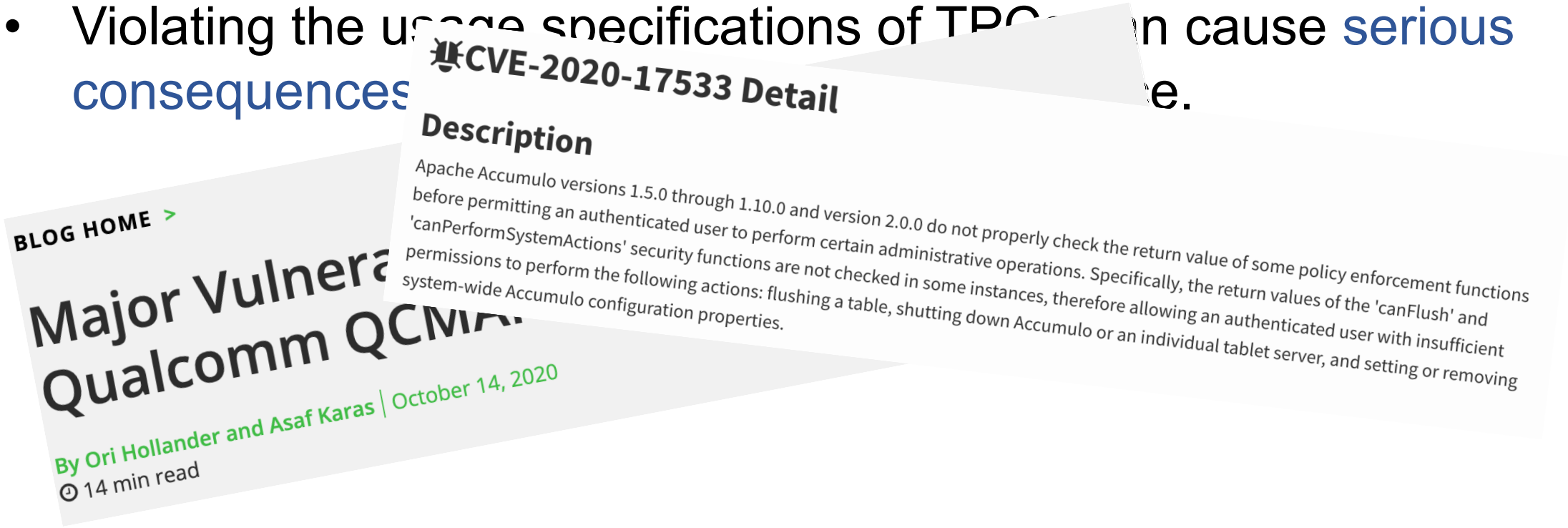


BusyBox

OpenSSL
Cryptography and SSL/TLS Toolkit

Background

- TPCs usually have strict usage specifications, e.g., checking the return value of the function.
- Violating the usage specifications of TPC… can cause serious consequences…e.



TPC Usage Violation

- **Deprecated API violation:** A set of APIs will be deprecated or abandoned for various reasons, e.g., security issues.
- **Return value violation:** Return values usually need to be checked after the API call.
- **Argument violation:** The arguments that are passed into APIs often have strict constraints.
- **Causality violation:** Many APIs may have a strict causal relationship, e.g., lock/unlock, fopen/fclose, and malloc/free.

Building a Practical System from Scratch

Two Challenges:

1. How to fill the gap between the *high-level specifications* from TPC documents, and the *low-level implementations* in the IoT firmware?

   - Previous works only perform well on ***well-formatted*** TPC documents and are hard to handle ***unusual*** or ***ambiguous*** API specifications.

Building a Practical System from Scratch

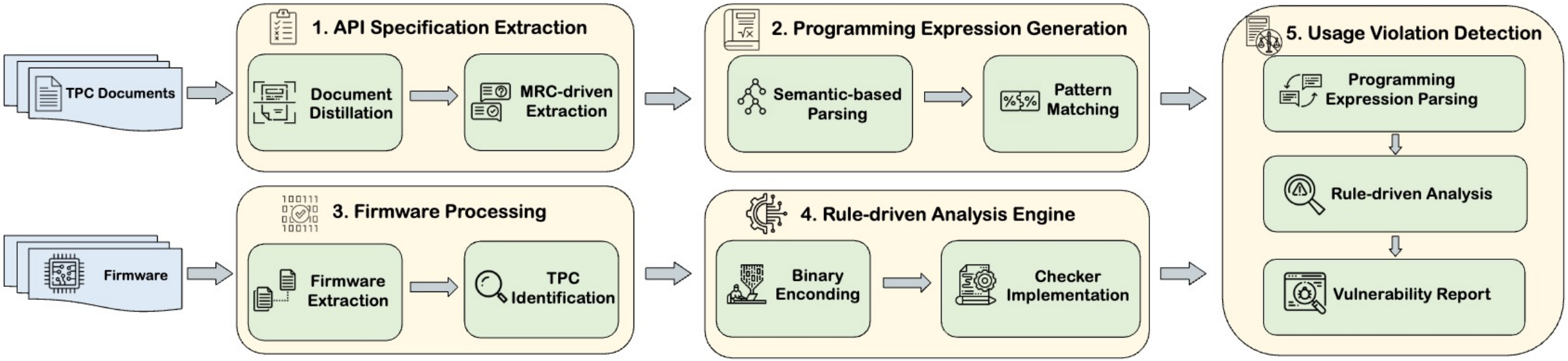
Two Challenges:

2. How to perform the TPC usage violation analysis on *closed-source* binaries?

   - Previous works only focus on ***source-level*** API misuse detection.

UVScan

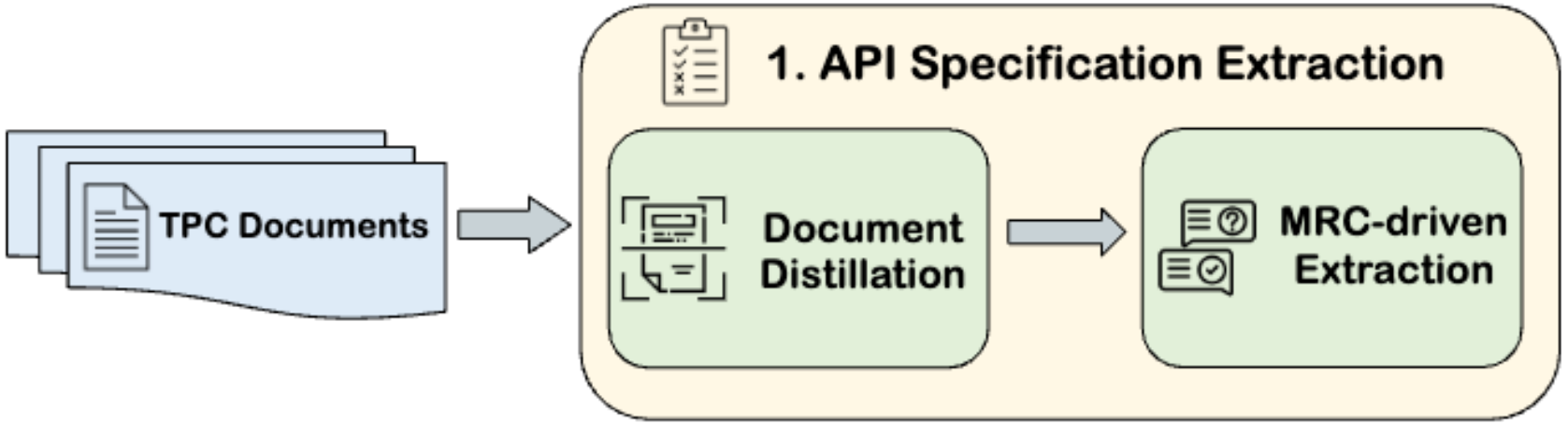
UVScan: The first automated and practical system to detect TPC usage violations in binary IoT firmware.



Framework of UVScan

Component 1: API Specification Extraction

- **Goal:** Extract the API specifications from corresponding TPC documents.



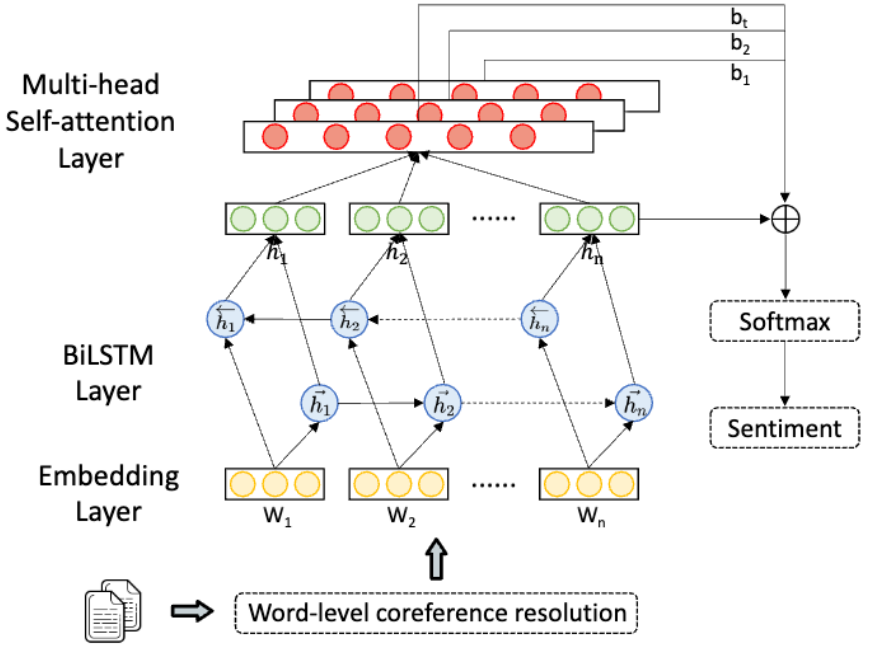
Step 1-1: Document Distillation

- **Goal:** Filter out irrelevant API descriptions and dig for relevant API descriptions.
- Previous research indicates that relevant API descriptions usually have a strong sentiment[1].
- This observation does not apply to all scenarios when analyzing the TPC document and will introduce false positives.
  - The sentence "*additionally it indicates that the session ticket is in a renewal period and should be replaced*" has a strong sentiment word "*should*" but it is not a relevant API description.

[1]Lv et al., "*Automatic Assumption Discovery and Verification Derivation from Library Document for API Misuse Detection*", CCS 2021.

Step 1-1: Document Distillation

- **Our Approach:**
  - Leverage WL-Coref, an off-the-shelf coreference resolution model, to resolve the coreferences in the TPC document.
  - Adopt BiLSTM model with the multi-head self-attention mechanism to capture the sentiment of a sentence.



Sentiment-based Document Distillation Model

Step 1-2: MRC-driven Extraction

- **Goal:** Extract precise API specifications from relevant API descriptions.
- **Our Approach:** Adopt the Machine Reading Comprehension (MRC) system with well-designed question sets.

"SQLITE_OK be returned by sqlite3_snapshot_recover if successful, or an SQLite error code otherwise"

"SQLite error code be returned by sqlite3_snapshot_recover If failed."

Question Sets

Category	Question
Return Value	What are return values supposed to be? In which condition does the function have a return value?
Causality	What operation is required if the return value is $ReturnValue_i$? What operation is required if $Condition_i$? Which function should be called before the API? Which function should be called after the API?
Argument	What is the value of the N-th argument supposed to be before the API? What is the value of the N-th argument supposed to be after the API? How to check the N-th argument before the API? How to check the N-th argument after the API?

Step 1-2: MRC-driven Extraction

- **Goal:** Extract prec… relevant API descriptions.
- **Our Approach:** Ac… Comprehension (MRC) system with… ets.

"SQLITE_OK be returned by sqlite3_… successful, or an SQLite error code …"

"SQLite error code be returned by s… If failed."

API: int pcap_activate(pcap_t *p);
Distilled Document: pcap_activate() returns 0 on success without warnings, a non-zero positive value on success with warnings, and a negative value on error. A non-zero return value indicates what warning or error condition occurred. A program should check for positive, negative, and zero return codes, and treat all positive return codes as warnings and all negative return codes as errors. If pcap_activate() fails, the pcap_t * is not closed and freed; the pcap_t * should be closed using pcap_close().

Question 1: What are return values supposed to be?
Answer: 0 on success without warnings; a non-zero positive value on success with warnings; a negative value on error

Question 2: In which condition does pcap_activate have a return value?
Answer: success without warnings; success with warnings; error

Question 3: What operation is required if the return value is 0?
Answer: No answer

Question 4: What operation is required if the return value is a non-zero positive value?
Answer: No answer

Question 5: What operation is required if the return value is a negative value?
Answer: the pcap_t * should be closed using pcap_close()

Question 6: What operation is required if successful without warnings?
Answer: No answer

Question 7: What operation is required if successful with warnings?
Answer: No answer

Question 8: What operation is required if there is an error?
Answer: the pcap_t * should be closed using pcap_close()

Question 9: Which function should be called before pcap_activate?
Answer: No answer

Question 10: Which function should be called after pcap_activate?
Answer: No answer

Question 11: What is the value of the first argument pcap_t *p supposed to be before pcap_activate?
Answer: No answer

Question 12: What is the value of the first argument pcap_t *p supposed to be after pcap_activate?
Answer: No answer

Question 13: How to check the first argument pcap_t *p before pcap_activate?
Answer: No answer

Question 14: How to check the first argument pcap_t *p after pcap_activate?
Answer: the pcap_t * should be closed using pcap_close()

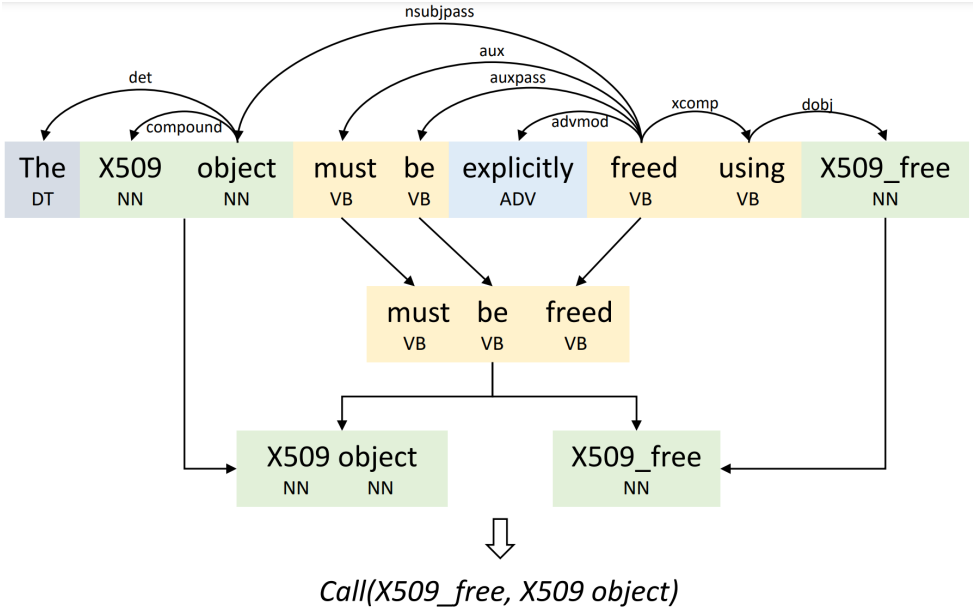
Question Sets

Question
What are return values supposed to be?
hich condition does the function have a return value?
peration is required if the return value is $ReturnValue_i$?
What operation is required if $Condition_i$?
Which function should be called before the API?
Which function should be called after the API?
value of the N-th argument supposed to be before the API?
e value of the N-th argument supposed to be after the API?
How to check the N-th argument before the API?
How to check the N-th argument after the API?

Component 2: Programming Expression Generation

- **Goal:** Transfer natural language-based API specifications into machine-readable representations.

- **Our Approach:**
  - Use POS tagging to annotate each word in an API specification.
  - Create a dependency tree by combining words with a close relationship.
  - Map common phrases into programming expressions, *Operation(argument1, argument2, …)*.

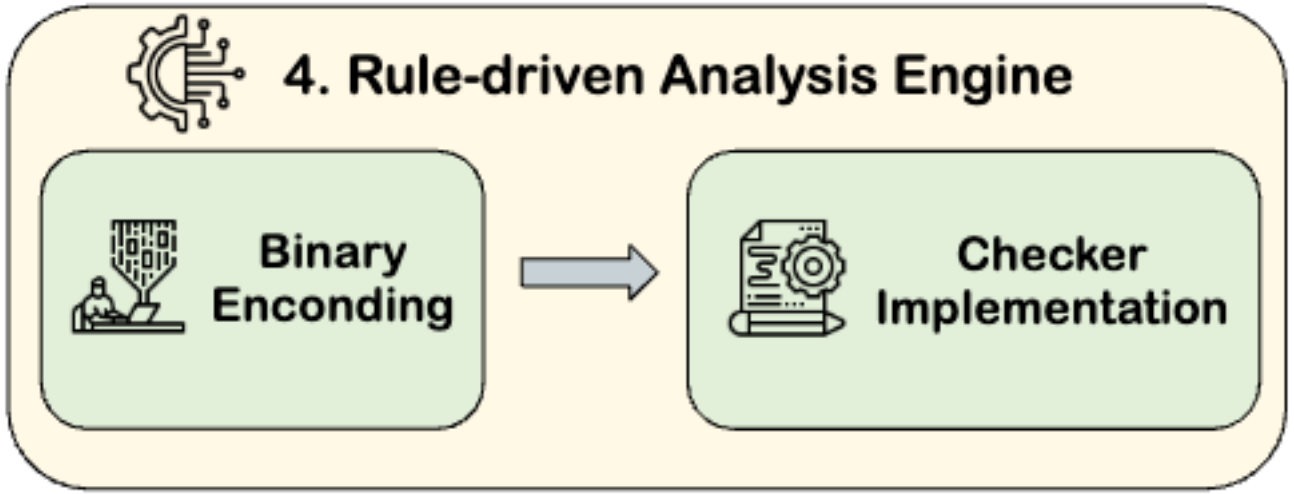


An Example of Programming Expression Generation

Component 3: Rule-driven Analysis Engine

- **Goal:** Leverage the generated programming expressions for usage violation detection.

- **Our Approach:** Encode the binary into Datalog facts and perform the usage violation check on the generated facts.



Step 3-1: Binary Encoding

- **Goal:** Encode the binary into Datalog facts.
- **Our Approach:** Leverage *Ddisasm* to encode the binary.

(address) A	
(size of the instruction) S_{instr}	
(size of the data element) S_{de}	
(register) R	
(segment register) R_{seg}	
(base register) R_{base}	
(index register) R_{idx}	
(instruction code) I	
(unique identifier of the i-th operand) O_i	
(immediate) IM	
(multiplier) M	
(displacement) D	
Predicate	$::= instruction(A, S_{instr}, P, I, O_1, O_2, O_3, O_4)$
	$\mid invalid(A)$
	$\mid op\_regdirect(O_i, R)$
	$\mid op\_immediate(O_i, IM)$
	$\mid op\_indirect(O_i, R_{seg}, R_{base}, R_{idx}, M, D, S_{de})$

Initial Datalog Facts Used by *Ddisasm*

```
.decl function_r0_usage_blez0(EA:address)
.output function_r0_usage_blez0

function_r0_usage_blez0(EA:address) :-
        instruction(EA,_,_,"BLEZ",_,_,_,_,_,_),
        instruction_get_op(EA,_,Op),
        op_regdirect_contains_reg(Op,"R0").
```

An example of A Rule that Checks the Return Value

Step 3-2: Checker Implementation

- **Goal:** Perform the usage violation check on the generated facts.
- **Our Approach:** Design four checkers based on the features of different TPC usage violations
  - **Deprecated API violation checker**: Maintain a list of deprecated APIs for each TPC.
  - **Return value violation checker**: Focus on the operation of the registers that hold the return value of the function, e.g., the R0 register in ARM32.
  - **Argument violation checker**: Focus on the operation of the registers that hold the arguments of the function, e.g., the R0-R3 register in ARM32.
  - **Causality violation checker**: Focus on the functions that are called before or after the API, and the operations under certain return values.

Evaluation

- **Goal:** Evaluate the performance of key components in UVScan and the overall performance of UVScan.

- Evaluate UVScan on four popular TPCs for concept validation.

Evaluation: API Description Extraction Accuracy

- **API description dataset (D_{DESC_test}):** Train and evaluate our sentiment-based document distillation model.

- Compare UVScan with three off-the-shelf tools: *Advance*, *RCNN*, and *ALICS*.

API Description Extraction Accuracy

Tool	$D_{DESC_{test}}$ Accuracy	F1
UVSCAN	92.41%	85.24%
Advance [30]	85.07%	74.37%
RCNN [27]	76.25%	61.50%
ALICS [33]	38.43%	29.02%

Evaluation: Usage Violation Detection Accuracy

- **Real-world usage violation dataset ($D_{Real\text{-}UV}$)** includes 77 known usage violations.
- **Artificial usage violation dataset ($D_{Real\text{-}UV}$)** includes 69 manually created usage violations.
- Compare UVScan with three state-of-the-arts: *Advance*, *APISAN*, and *APEx*.

Usage Violation Detection Accuracy

Performance	UVSCAN x86	UVSCAN ARM	UVSCAN MIPS	*Advance*	*APISAN*	*APEx*
$D_{Real-UV}$						
Precision	72.84%	74.70%	77.03%	80.72%	17.31%	23.07%
Recall	76.62%	80.52%	74.03%	87.01%	11.69%	7.79%
$D_{Artif-UV}$						
Precision	68.92%	74.32%	76.47%	77.33%	25.49%	34.62%
Recall	73.91%	79.71%	75.36%	84.06%	18.84%	13.04%

Large-scale Analysis on IoT Firmware

- Conduct a large-scale analysis on 4,545 firmware images.
- **Research Question** ①: Which are the most prevalent TPC usage violations in IoT firmware?
- Detect 27,621 potential usage violations of the four TPCs in the 4,545 firmware images.

Usage Violation Distribution

TPC	# Deprecated API Violation	#Causality Violation	# Return Value Violation	# Argument Violation
OpenSSL	4,831	3,679	3,521	1,073
SQLite	2,740	1,996	931	112
libpcap	3,359	2,515	1,364	857
libxml2	418	114	75	36
Overall	11,348	8,304	5,891	2,078

Large-scale Analysis on IoT Firmware

- **Research Question** ②: What are the practical impacts of TPC usage violations on IoT firmware?

- **Impacts:**
  - **Security vulnerabilities**: Can be exploited to perform attacks, e.g., the Man-In-The-Middle (MITM) attack.
  - **Ordinary bugs**: May result in the malfunctioning of firmware but cannot be leveraged for attacks.
  - **No impact**: Will not affect the operation of the device and cannot be used for attacks.

Conclusion

- Propose UVScan, the first automated and practical system to detect TPC usage violations in binary IoT firmware.
- Conduct the first large-scale analysis on TPC usage violation problem in IoT firmware.

Thanks!