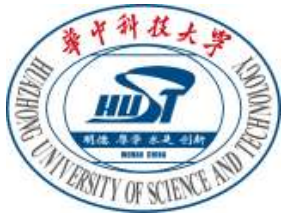
From Intention to Practice: Towards Systematic Validation of NIDS Rule Enforcement

Huan Liu[1,*], **Haoyu Chen[2,*]**, Biang Xu[1,3], Jingyao Zhou[1],
Bin Yuan[1,4,†], Qiankun Zhang[1], Deqing Zou[1,3], Hai Jin[5]

[1]School of Cyber Science and Engineering, HUST [2]Zhejiang Lab [3]Jinyinhu Laboratory
[4]Songshan Laboratory [5]School of Computer Science and Technology, HUST



之江实验室
ZHEJIANG LAB

Rule-Based Network Intrusion Detection

Rule-based NIDS plays a pivotal role in cybersecurity:

- Employs **Deep Packet Inspection (DPI)** to monitor network traffic
- Matches traffic against **predefined detection rules** to identify attacks
- Rules are either manually authored or selected from expert-maintained rulesets

A rule consists of:

- **Header**: defines which traffic to evaluate
- **Options**: specify how traffic is evaluated

```
1  alert tcp $EXTERNAL_NET any -> $HOME_NET 80
2    (msg: "Attack attempt!";
3     service: http;
4     http_raw_body;
5     content: "malicious payload", fast_pattern;
6     http_uri;
7     pcre: "/[?&]user=(intruder|attacker)/i";
8     gid:1; sid:10000; rev:1;)
```

Rules Are Not Enforced as Intended

Rule-Related Issues

Syntactic Errors

Minor errors (e.g., whitespace in patterns) fundamentally undermine detection

Poorly Designed Rules

Crafted packets can alter malicious payload appearance to fool matching

Rule Interactions

Conflicts and overlaps between rules cause unexpected consequences

Implementation-Related Issues

Inconsistent Protocol Parsing

NIDS and endpoints parse protocols differently, causing evasion

Regex/String Sanitization

Worst-case complexity in regex/string sanitization causes timeouts

Performance Degradation

Coding errors prevent packets from being evaluated in required time windows

There is **no inherent guarantee** that deployed rules are enforced as intended — gaps arise from both rule composition and implementation flaws.

Existing Approaches & Limitations

Existing testing approaches are often **rule-irrelevant** and **lack systematic** methodologies

Static Analysis

Proposed to identify potential issues using theoretical models or formal verification.

Limitation: Ruleset may still fail during enforcement due to implementation flaws in the NIDS that static analysis cannot catch.

Protocol-Based Fuzzing

Tools like Geneva and Boofuzz mutate specific protocol fields to test NIDS implementations.

Limitation: Primarily target specific protocols rather than NIDS rulesets. Little chance to trigger targeted rules accurately.

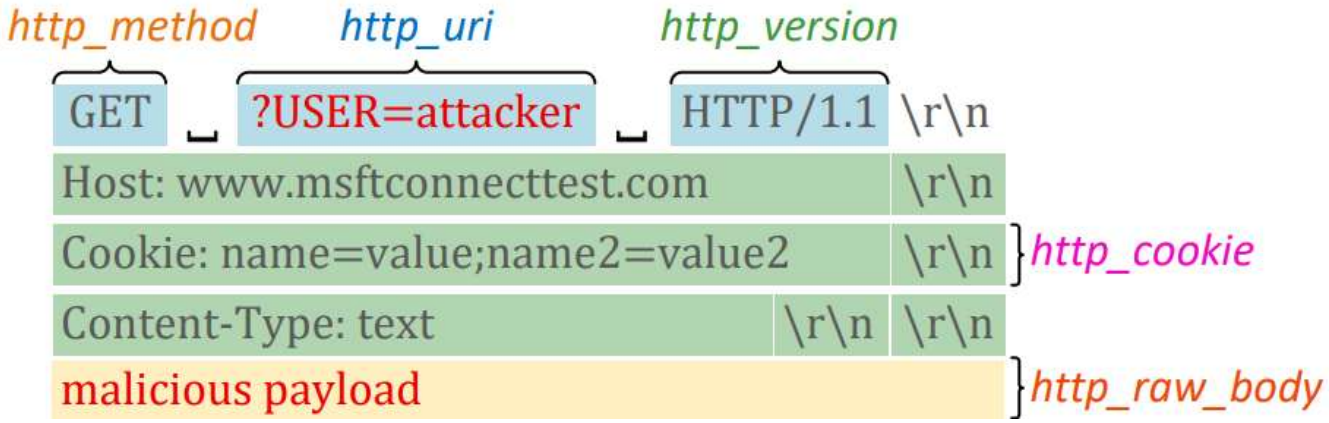
How to validate whether rules deployed can be enforced as expected in practice?

Key Insight of NIDSFUZZ

RQ1: How to design test packet generation strategies for comprehensive validation?

Rule-based Generation

```
1  alert tcp $EXTERNAL_NET any -> $HOME_NET 80
2    (msg: "Attack attempt!";
3     service: http;
4     http_raw_body;
5     content: "malicious payload", fast_pattern;
6     http_uri;
7     pcre: "/[?&]user=(intruder|attacker)/i";
8     gid:1; sid:10000; rev:1;)
```



❶ *http_raw_body* : { *content* : ["malicious payload", fast_pattern]},
http_uri : { *pcre* : "/[?&]user=(intruder|attacker)/i" }

❷

Buffer Constraints

Key Insight of NIDSFUZZ

RQ1: How to design test packet generation strategies for comprehensive validation?

Tailored Mutation Strategies

- *Pass-Through Strategy*

Generate test packets directly without any mutation.

- *Obfuscation Strategy*

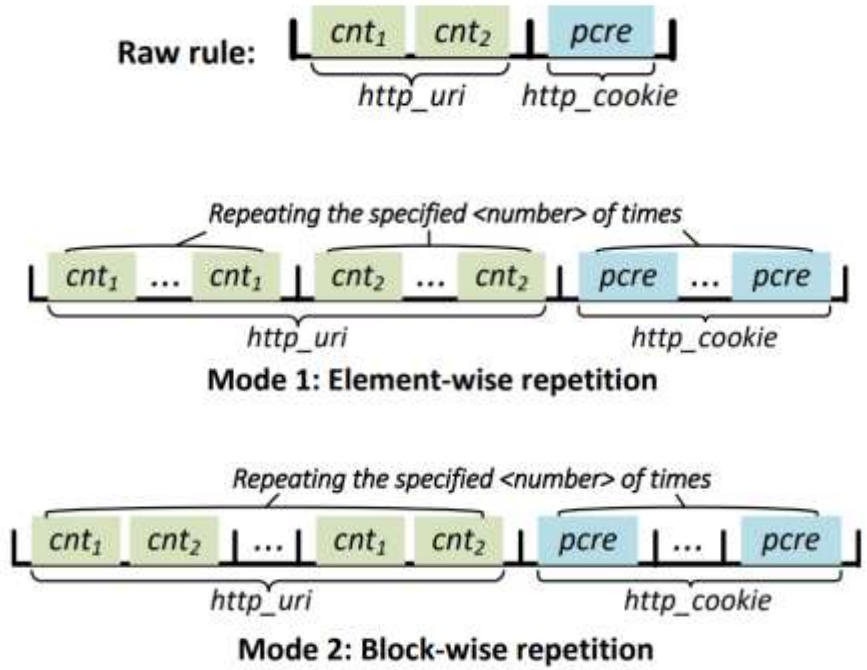
Applies obfuscation actions to payload detection options.

- *Repetition Strategy*

Repeats buffer constraints.

- Blending Strategy

Selects multiple rules and blends their buffer constraints.



Key Insight of NIDSFUZZ

RQ1: How to design test packet generation strategies for comprehensive validation?

Tailored Mutation Strategies

- *Pass-Through Strategy*

Generate test packets directly without any mutation.

- *Obfuscation Strategy*

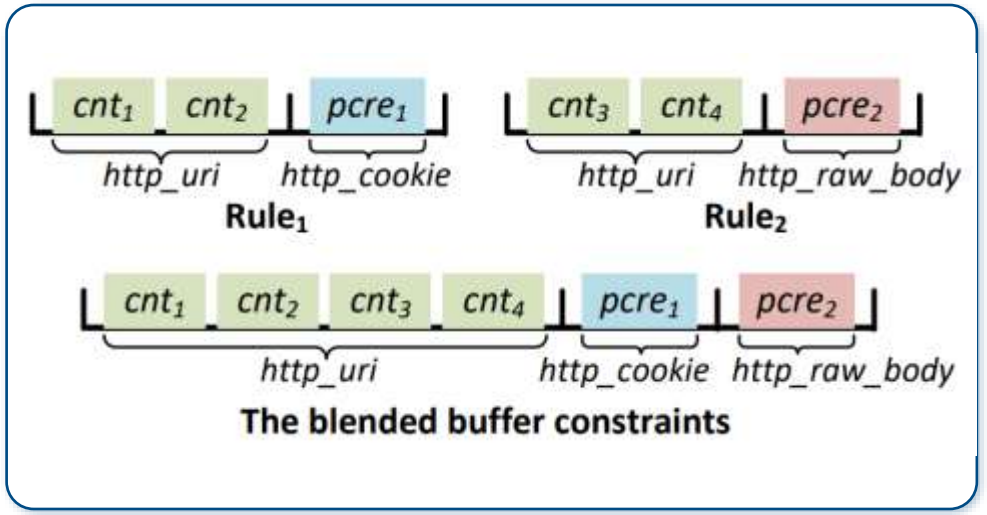
Applies obfuscation actions to payload detection options.

- *Repetition Strategy*

Repeats buffer constraints.

- Blending Strategy

Selects multiple rules and blends their buffer constraints.



Key Insight of NIDSFUZZ

RQ2: How to efficiently inject test traffic into a NIDS?

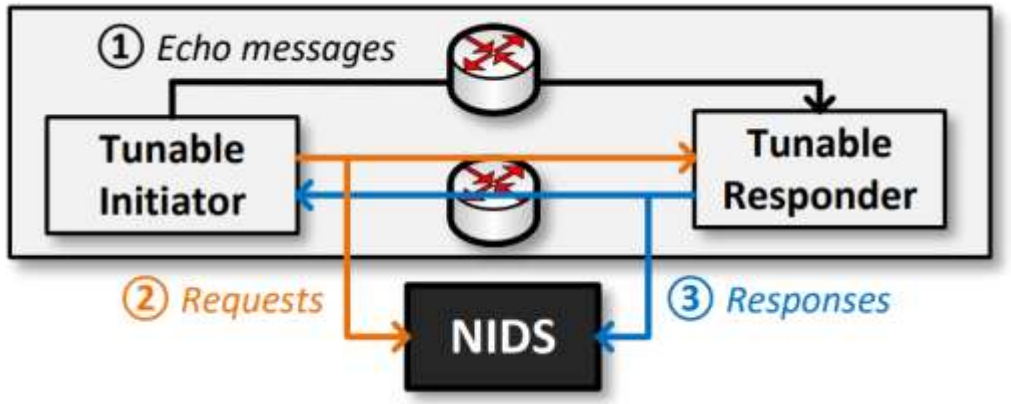
Tunable Initiator & Responder

A **tunable** injection method without executing packet processing logic:

① Echo Transmission

② Request Transmission

③ Response Transmission



Benefit: No real program processing overhead. Both request and response packets are fully tunable — ports, order, and content can be conveniently adjusted.

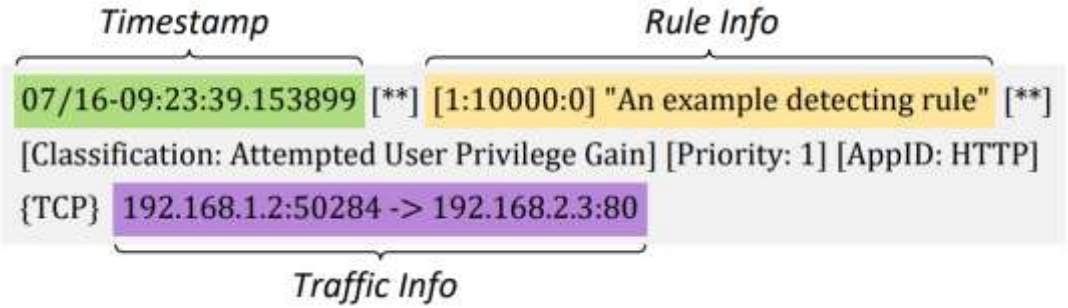
Key Insight of NIDSFUZZ

RQ3: How to identify potential rule enforcement issues based on alerts?

Alert-based Differential Analysis

Port-Based Alignment

- Examines traffic port in alert against initiator ports
- Port window handles delayed alerts
- Continuously increasing ports prevent ambiguity



Key Insight of NIDSFUZZ

RQ3: How to identify potential rule enforcement issues based on alerts?

Alert-based Differential Analysis

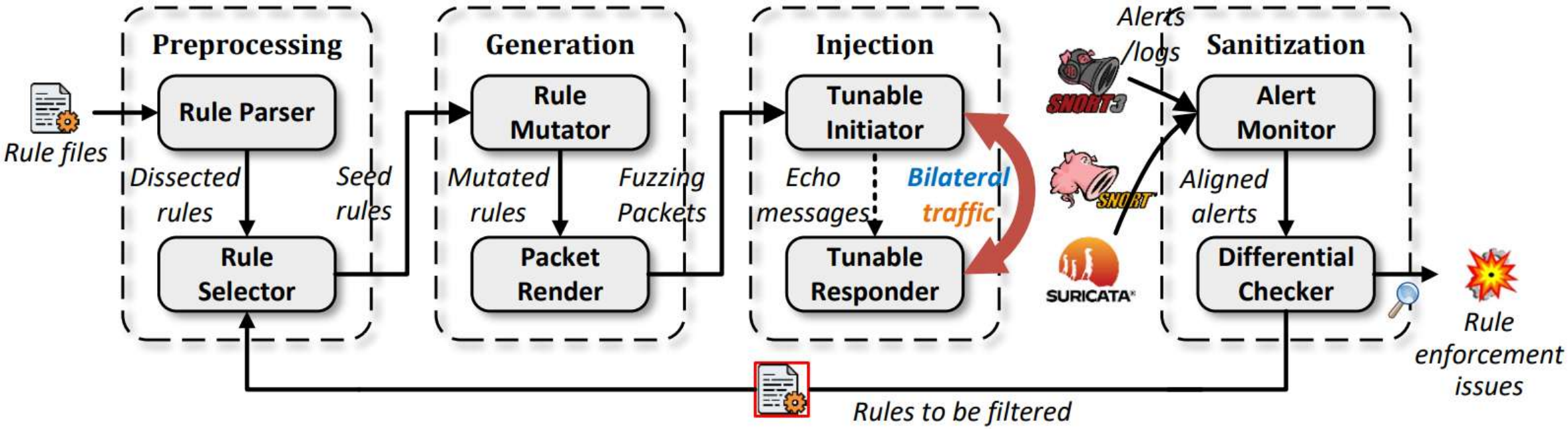
Oracle 1: Unexpected Rule Triggering

If alerts attribute to rules not selected in the current iteration, the generated packet unexpectedly triggered unselected rules.

Oracle 2: Cross-NIDS Inconsistency

If two NIDS with the same ruleset produce different alerts for identical test traffic, it indicates a potential rule enforcement issue.

System Implementation



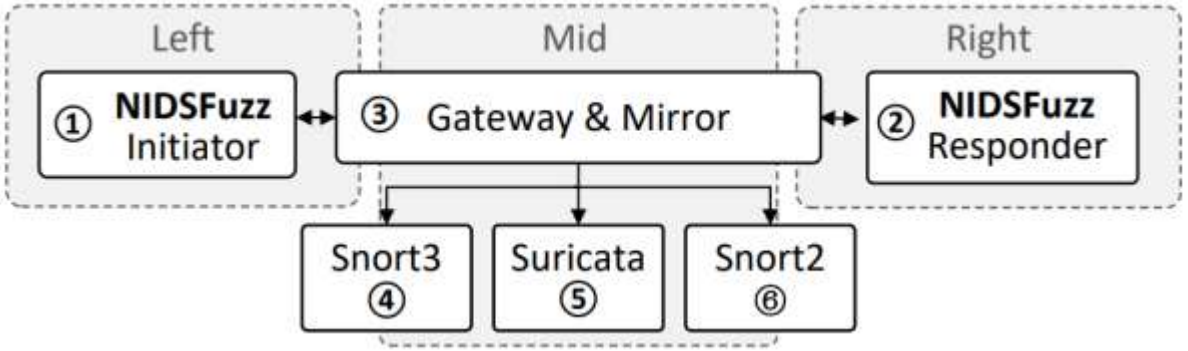
NIDSFUZZ is **rule-oriented** — it generates test traffic based on the very ruleset deployed, enabling targeted validation with guaranteed coverage.

Experiment Setup

Hardware:

AMD EPYC 7742 CPU, 128 GB RAM

Network Topology:



Tested NIDS and Rulesets:

NIDS Version		
Snort3	**Suricata**	**Snort2**
3.6.0	7.0.8	2.9.20

# of Rules			
HTTP	**FTP**	**DNS**	**SIP**
4033	297	112	219

Findings Overview

	Category	Issue & Severity	Stat	Strategy				Affected NIDS			Affected Rules			
				P	O	R	B	S3	Su	S2	HTTP	FTP	DNS	SIP
Rule Induced Issues	Intra-Rule	Redundant Sticky Buffers [H]	●	+	–	–	–	✗	✓	✗	✓	✗	✗	✗
		Outsized Option Values [H]	●	+	–	–	–	✓	✓	✓	✓	✓	✗	✗
		Misused Keywords [H]	●	+	–	–	–	✗	✓	✓	✗	✓	✗	✓
		Overbroad Signatures [M]	○	+	–	–	–	✓	✓	✓	✓	✓	✓	✓
	Inter-Rule	Duplicate Rules [M]	●	+	–	–	–	✓	✓	✓	✓	✗	✓	✓
		Overlapping Rules [M]	●	+	–	+	+	✓	✓	✓	✓	✓	✓	✓
Implementation Induced Issues	Protocol Parsing	Invalid Status Code [H]	●	–	–	–	+	✓	✗	✓	✓	✗	✗	✗
		Corrupted Content Length [H]	●	–	+	–	+	✗	✓	✗	✓	✗	✗	✗
		Mis-stripped Spaces [L]	●	–	–	–	+	✗	✓	✗	✓	✗	✗	✗
		Omitted Checks [L]	○	–	+	+	+	✓	✗	✗	✗	✗	✓	✗
	Rule Matching	Complex Regex [H]	●	+	+	+	+	✗	✓	✗	✓	✓	✓	✓
		Inconsistent Interpretation [H]	○	+	–	–	–	✓	✓	✓	✓	✓	✗	✓
		Reduced Reliability [M]	○	+	+	+	+	✗	✓	✗	✓	✓	✓	✓
		Direction Mismatch [M]	○	+	+	+	+	✓	✗	✗	✗	✗	✓	✗
		Transport Layer Mismatch [M]	●	+	–	–	+	✓	✗	✗	✗	✗	✓	✓
		Obfuscated Characters [H]	○	–	+	–	–	✓	✗	✗	✓	✓	✗	✓
		Limited Packet Size [H]	●	–	–	+	+	✓	✓	✓	✓	✓	✓	✓
		Long Segmentation [M]	●	–	–	+	–	✓	✓	✓	✗	✓	✓	✓

P represents the pass-through strategy. **O** represents the obfuscation strategy. **R** represents the repetition strategy. **B** represents the blending strategy.
S3 stands for Snort3. **Su** stands for Suricata. **S2** stands for Snort2.
○ indicates the issue was reported to the official and is still under investigation, while ● means it was reported and confirmed.
\+ indicates that the mutation strategy reported related alert discrepancies, while – means no relevant discrepancies were found in the evaluation.
✓ indicates that the NIDS or the rules of the protocol is affected by the issue, while ✗ indicates neither is affected.
[H] denotes high-severity issues, [M] denotes medium-severity issues, and [L] denotes low-severity issues.

Rule-Induced Issues

Redundant Sticky Buffers

```
.....
  file_data; file_data;
  content:"CustomEvent",nocase;
  content:"connect",within 50,nocase;
  content:"CustomEvent",nocase;
  content:"message",within 50,nocase;
  content:"message_type",nocase;
  content:"launch_meeting",within 50,nocase;
  content:"GpcComponentName",fast_pattern;
  content:!"YXRtY2NsaS5ETEw=",within
20,nocase;
  sid:41408; rev:3;
  .....)
```

Rule 1:41408:3

Misused Keywords

```
.....
  content:"CSeq|3A|"; fast_pattern:only;
  pcre:"/^CSeq\x3A[^\r\n]+[\x01-\x08\x0B\
x0C\x0E-\x1F\x80-\xFF]/Hsmi";
  sid:20306; rev:4;
  .....)
```

```
.....
  content:"CSeq|3A|",fast_pattern,nocase;
  sip_header;
  pcre:"/^CSeq\x3A[^\r\n]+[\x01-\x08\x0B\
x0C\x0E-\x1F\x80-\xFF]/ims";
  sid:20306; rev:4;
  .....)
```

Rule 1:20306:4

Rule writing is **error-prone**, and these errors can be exploited to evade detection.

Rule-Induced Issues

Overlapping Rules

```
.....
   file_data;
   content:"|3A|first-letter { float|3A| ",fast_patte
rn,nocase;
   content:"|5B 22|setAttribute|22 5D 28|'style', '
display|3A| table-cell'";
   content:"|5B 22|style|22 5D 5B 22|display|22
5D|= 'none'",within 62;
   sid:44978; rev:3;
   .....)
```

Rule 1:44978:3

```
.....
   file_data;
   content:"|3A|first-letter { float|3A| ",fast_patte
rn,nocase;
   content:"|5B 22|setAttribute|22 5D 28|'style', '
display|3A| table-cell'";
   content:"|5B 22|style|22 5D 5B 22|display|22
5D|= 'none'",within 60;
   sid:29579; rev:3;
   .....)
```

Rule 1:29579:3

This can be exploited to launch **squealing attacks**, where a flood of false alerts overwhelms operators.

Implementation-Induced Issues

Invalid Status Code

Snort3 does not inspect HTTP responses with invalid status codes.

Corrupted Content-Length

Malformed Content-Length header causes Suricata to hang until timeout.

Inconsistent Interpretation

Same rule keywords interpreted differently across NIDS platforms.

Complex Regex

Suricata struggles with complex pcre patterns.

… …

Comparison with Existing Approaches

Method	Snort3	Suricata	Snort2
Geneva	0.02%	0%	0.02%
Boofuzz-HTTP	1.17%	1.21%	1.19%
CICIDS2017	1.18%	1.54%	1.18%
CICIDS2018	2.61%	2.92%	2.79%
NIDSFuzz	94.45%	91.72%	94.15%

- Geneva [1]

TCP-layer only, rule-irrelevant

- Boofuzz [2]

No rule-oriented packets, unidirectional

- CIC-IDS 2017/2018 [3]

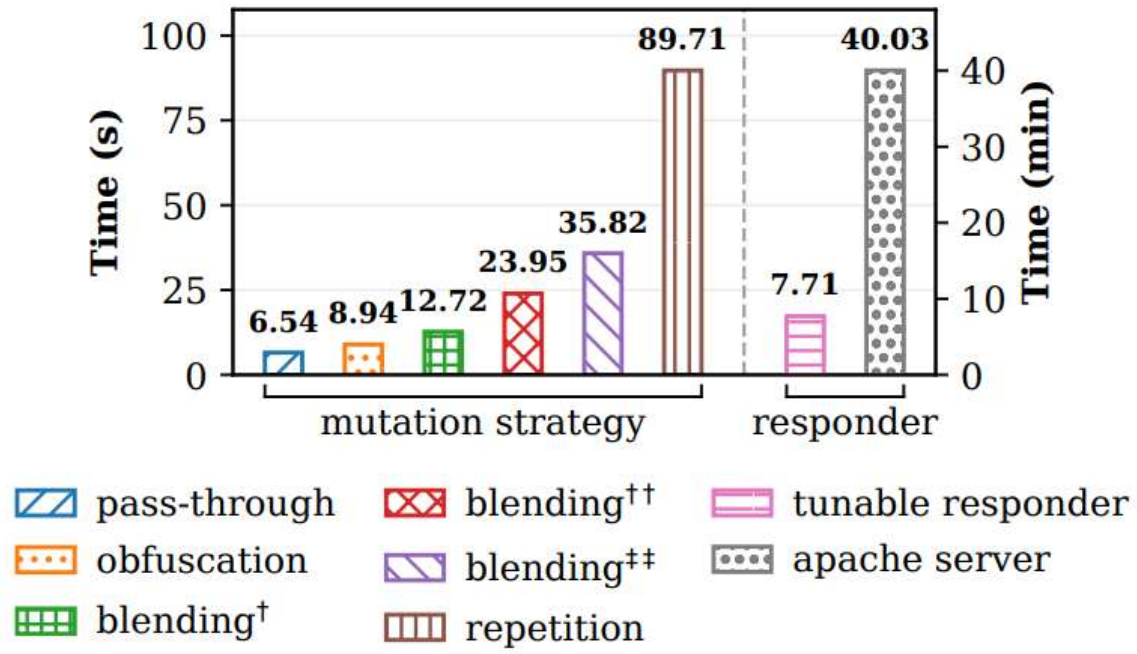
Narrow attack families only

[1] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. Geneva: Evolving Censorship Evasion Strategies. In Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security, pages 2199–2214, 2019

[2] https://github.com/jtpereyda/boofuzz

[3] https://www.unb.ca/cic/datasets/ids-2017.html

Performance Evaluation



NIDSFUZZ is efficient in both **generation** and **injection**. The tunable injection architecture eliminates real protocol overhead.

Conclusion

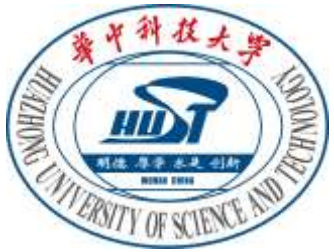
- We presented **NIDSFUZZ**, a systematic, rule-oriented fuzzing framework for validating NIDS rule enforcement.
- It addresses three core challenges. **The rule-based generation** for guaranteed coverage; **the tunable bilateral injection** for efficient traffic delivery; **and the differential alert analysis** for accurate issue identification.
- Substantial numbers of issues are uncovered.

nsdi '26
23rd USENIX Symposium on Networked Systems Design and Implementation

Thank you!

For more details, please read our paper:
From Intention to Practice: Towards Systematic Validation of NIDS Rule Enforcement

Contact: liuhuan0xf@hust.edu.cn
Source code available at: https://github.com/lh0xf/nidsfuzz



之江实验室
ZHEJIANG LAB