VulTerminator: Bringing Back Template-Based Automated Repair for Fixing Java Vulnerabilities

Quang-Cuong Bui, Emanuele Iannone, Riccardo Scandariato

Institute of Software Security
Hamburg University of Technology, Germany

SEC 4 AI 4 SEC

SANER 2026

SANER'26 – 17th March, 2026
Limassol, Cyprus

Template-based Automatic Vulnerability Repair (AVR) is still underexplored

- Current *SOTA* AVR research relies on Large Language Models and treats repair as a code translation task
  - Struggle with the complex nature of vulnerability fixes
  - Lack of high-quality training datasets (especially for Java)
- Template-based APR remains a popular approach to fix general bugs, yet a few applied to vulnerabilities
  - Vulnerability patches also exhibit recurring fix templates [1]

[1] Bui et al. APR4Vul: An empirical study of automatic program repair techniques on real-world Java vulnerabilities. EMSE 2024.

Motivation examples

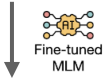
```
// Human patch of CVE-2017-5662 (XXE vulnerability)
try {
  saxParser = saxFactory.newSAXParser();
  parser = saxParser.getXMLReader();
  ...
+ parser.setFeature("http://xml.org/sax/features/" +
+    "external-general-entities", false);
+ parser.setFeature("http://xml.org/sax/features/" +
+    "external-parameter-entities", false);
catch (SAXException e) {
  e.printStackTrace();
}

// Human patch of CVE-2013-4378 (XSS vulnerability)
- write(remoteAddr);
+ write(htmlEncodeButNotSpace(remoteAddr));
```

OWASP XXE Prevention Cheatsheet[1] provides full remediation for the six most common XML Parsers

Heuristic-based Fix Template

<sanitization_method>(remoteAddr)

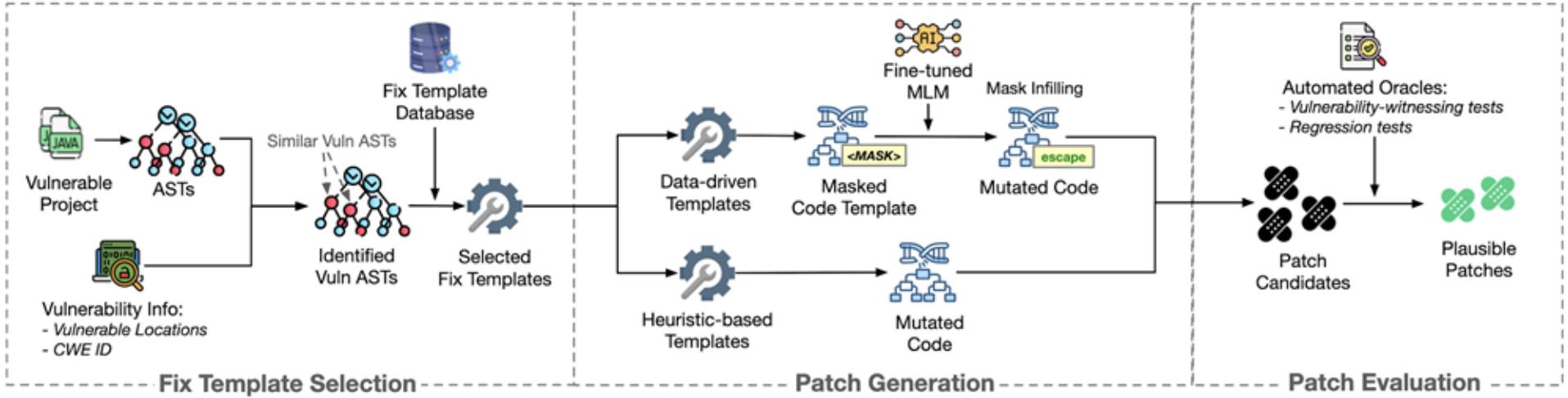


htmlEncodeButNotSpace(remoteAddr)

Data-driven Fix Template

[1] https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

VulTerminator's main workflow



Heuristic-based templates

ID	Fix Template	Explanation / Target Vulnerability
FT1	Prevent XXE Vulnerabilities	Configures XML parsers to disallow custom DTDs or external entities (CWE-611)
FT2	Instantiate Secure Random Generator	Replaces predictable PRNGs (e.g., `java.util.Random`) with `SecureRandom`
FT3	Secure SnakeYaml Instantiation	Uses `SafeConstructor` to prevent the Deserialization of Untrusted Data (CWE-502)
FT4	Prevent Path Traversal	Validates and normalizes file paths using the Java NIO Path API (CWE-22)
FT5	Secure Temp File Creation	Uses `Files.createTempFile` to enforce restrictive access permissions on Unix/Linux systems
FT6	Prevent Sensitive Data Exposure	Removes sensitive data variables from public endpoints, WebUIs, or logging outputs

Fix templates are formulated from repair patterns mined in our previous work [1]

Data-driven templates

ID	Fix Template	Fix Ingredients	Example Template Code
FT7	Sanitize Input	Static methods returning a String	`method(<mask0>(input));`
FT8	Break Infinite Loop	Atomic conditional expressions	`if (<mask0>) break;`
FT9	Check Permission	Method calls & conditional expressions	`if (<mask1>) <mask0>();`
FT10	AddIf_Throw	Conditional expressions & exception types	`if (<mask0>) throw new <mask1>(...);`
FT11	AddIf_Return	Conditional expressions & return values/variables	`if (<mask0>) return <mask1>;`

Example of Model Input/Output

Fine-tune UniXCoder on 4,147 high-quality repair instances for predicting Fix Ingredients

```
// ---------- Source ----------
// FIX_TEMPLATE: SanitizeInput
// FIX_INGREDIENTS: HTMLUtils decodeString urlEncode
↪  escapeHtml trimWhitespace getStringForJS...
private void writeSession(String input) {
  write(<mask0>(input));
}
// ---------- Target ----------
<mask0>HTMLUtils.escapeHtml
```

Research questions

- **RQ1:** How well does VulTerminator fix Java vulnerabilities compared to the state-of-the-art AVR and APR approaches?
- **RQ2:** How does VulTerminator perform in fixing vulnerabilities of different types?
- **RQ3:** To what extent are the vulnerabilities fixed by VulTerminator unique compared to the state-of-the-art AVR and APR approaches?
- **RQ4:** How does each fix template contribute to the overall repair performance of VulTerminator?

Evaluation setup & Datasets

- Baselines
  - **SOTA AVR:** VulMaster, VulRepair
  - **Templated-guided APR:** TBar, GAMMA, NTR
  - **LLMs:** GPT-4o, Gemini-2.5 Pro
- Datasets
  - **Vul4J+**: 106 vulns with vulnerability-witness tests
  - **Vul4L**: 169 vulns newly constructed
- Patch selection for assessing correctness
  - **Vul4J+**: first patch passing all the tests
  - **Vul4L**: top three patches generated by the tools

RQ1 – Overall repair performance

TABLE II: Repair results on 35 single-hunk vulnerabilities (a subset of Vul4J+).

*The repair results of VulMaster and VulRepair$_{NTR}$ are extracted from previous works [4], [16].

Tools/Models	*Template-guided APR*			*SOTA AVR*		*LLMs*		*Our Tool*
	TBar	**GAMMA**	**NTR**	**VulMaster**	**VulRepair$_{NTR}$**	**GPT-4o**	**Gemini-2.5 Pro**	**VulTerminator**
#Correct/#Plausible	5/8	6/9	11/14	9/-*	4/10*	13/14	15/17	17/19

TABLE III: Repair results on the Vul4J+ and Vul4JL datasets.

Tools/Models	**Vul4J+** (106 Vulns)	**Vul4JL** (169 Vulns)
	#Correct/#Plausible	#Correct
TBar	5/11	1
GAMMA	6/10	2
NTR	16/20	7
GPT-4o	17/22	28
Gemini-2.5 Pro	29/32	37
VulTerminator	31/35	47

VulTerminator achieves best repair performance

RQ2 – Repair performance by CWE

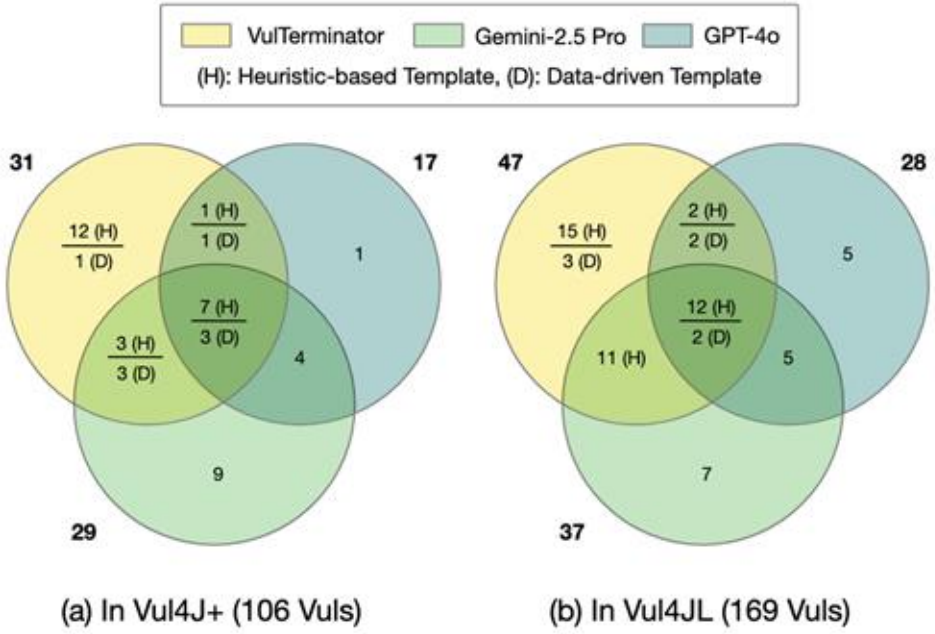
TABLE IV: Repair performance on different types of vulnerabilities in the Vul4JL dataset.

The CWEs are ranked according to their frequency in the dataset. **OWASP10*** denotes whether the CWE is listed in the ten most critical Web Application Security Risks for 2021, as published by OWASP [53].

Rank	CWE ID	CWE Name	Count	OWASP10*	GPT-4o	Gemini-2.5 Pro	VulTerminator
1	CWE-79	Cross-site Scripting (XSS)	20	✓	20%	15%	20%
2	CWE-22	Path Traversal	16	✓	12.5%	18.75%	68.75%
3	CWE-502	Deserialization of Untrusted Data	16	✓	6.25%	18.75%	18.75%
4	CWE-611	XML External Entity Injection (XXE)	11	✓	27.27%	36.36%	81.82%
5	CWE-200	Exposure of Sensitive Information	8	✓	0%	0%	100%
6	CWE-918	Server-Side Request Forgery (SSRF)	6	✓	0%	16.67%	16.67%
7	CWE-20	Improper Input Validation	5	✓	0%	0%	20%
8	CWE-668	Exposure of Resource to Wrong Sphere	5	✓	20%	80%	100%
9	CWE-89	SQL Injection (SQLi)	5	✓	20%	40%	0%
10	CWE-203	Observable Discrepancy	4		25%	50%	0%
		Total (Top 10)	**96**	-	13.54%	22.92%	36.46%
		Total (Whole dataset)	**169**	-	16.57%	21.89%	27.81%

VulTerminator is efficient in fixing vulnerabilities for:
CWE-22, CWE-611, CWE-200, CWE-668

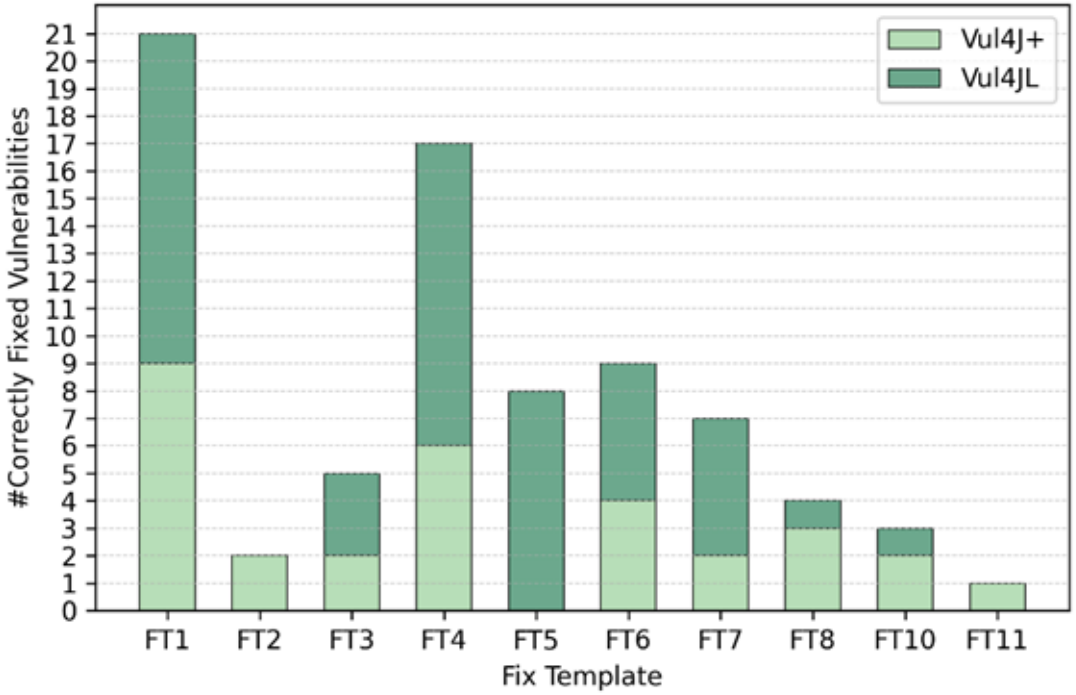
RQ3 – Overlapped fixed vulnerabilities



(a) In Vul4J+ (106 Vuls) (b) In Vul4JL (169 Vuls)

Heuristic-based templates help VulTerminator fix the most unique vulnerabilities

RQ4 – Individual Fix Template Contribution



FT1 (for XXE) and
FT4 (for Path Traversal)
contribute the most success

Key takeaways

/tuhh-softsec/VulTerminator

- VulTerminator is a hybrid AVR approach, leveraging both *Heuristic-based* and *Data-driven Fix Templates*
- VulTerminator achieves best repair performance, outperforms SOTA AVR and Templated-guided APR approaches
- While heuristics drive current success, expanding *data-driven templates* remains a key area for future improvement