
# Remote Code Execution from SSTI in the Sandbox: Automatically Detecting and Exploiting Template Escape Bugs

Yudi Zhao, Yuan Zhang, Min Yang

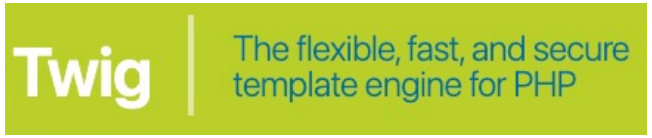
Fudan University

# Agenda

- Background
- Problem
- Approach
- Evaluation
- Conclusion

# Template Engine (TE)

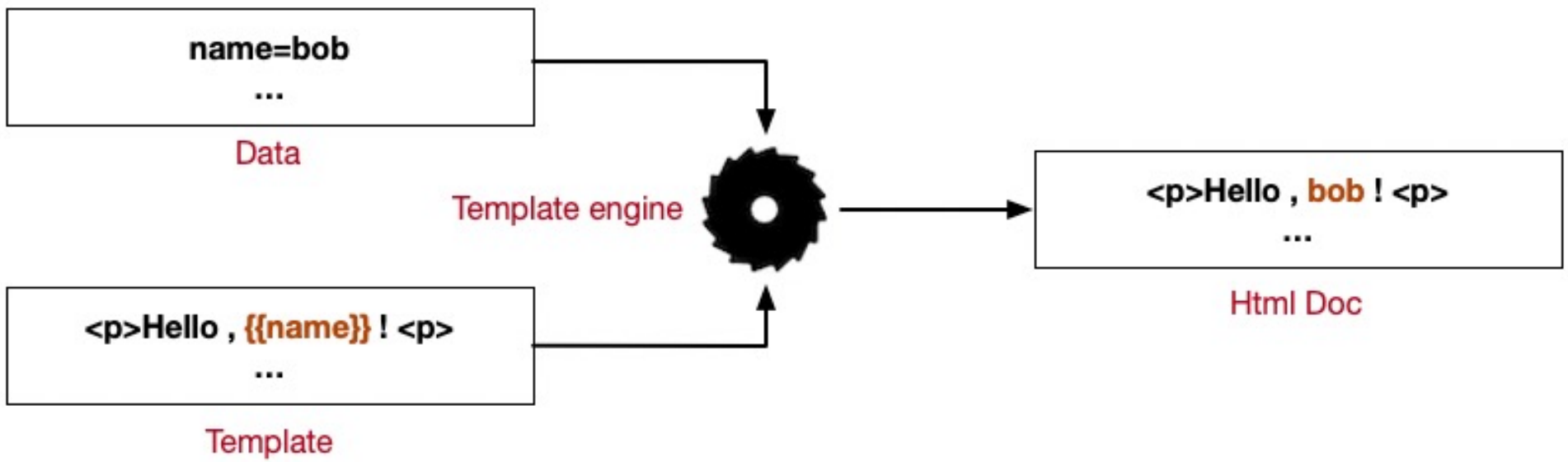
- Template engines help web applications generate dynamic HTML views from the template.




- Template engines are widely used in CMS applications.
  - More than 65% of popular PHP applications on GitHub use TEs to generate front-end views.

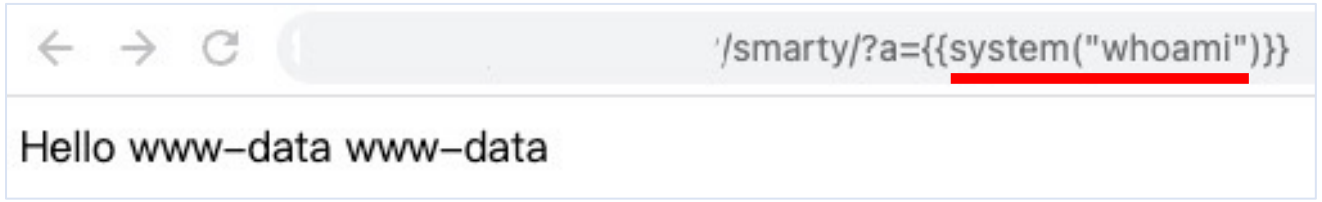
# How do TEs work?

- TEs render input data into HTML documents according to a pre-defined template.

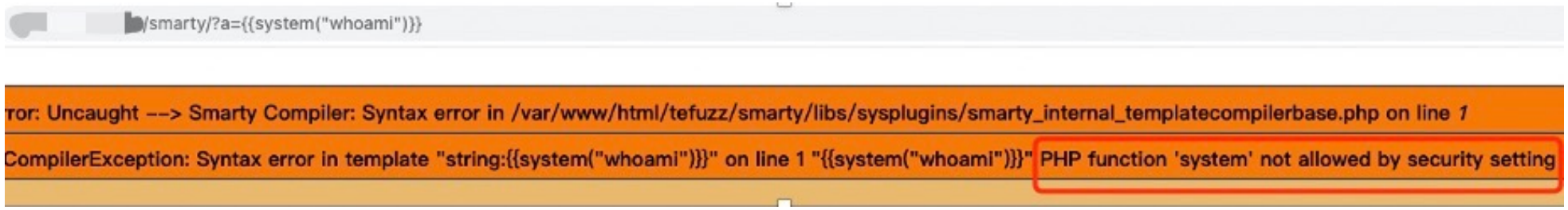


# SSTI and Sandbox Mode

- New Injection Vectors in TE: server-side template injection (SSTI)
  - abuse the TE capabilities by controlling the template
  - can be used to achieve high-risk exploit primitives, e.g., LFI, XSS, and RCE




- TE Sandbox Mode
  - defeat SSTI attacks by restricting the TE capabilities given to a template



# TE Sandbox Bypass (Template escape bug)

- A kind of TE bug that can bypass the sandbox and gain RCE with SSTI

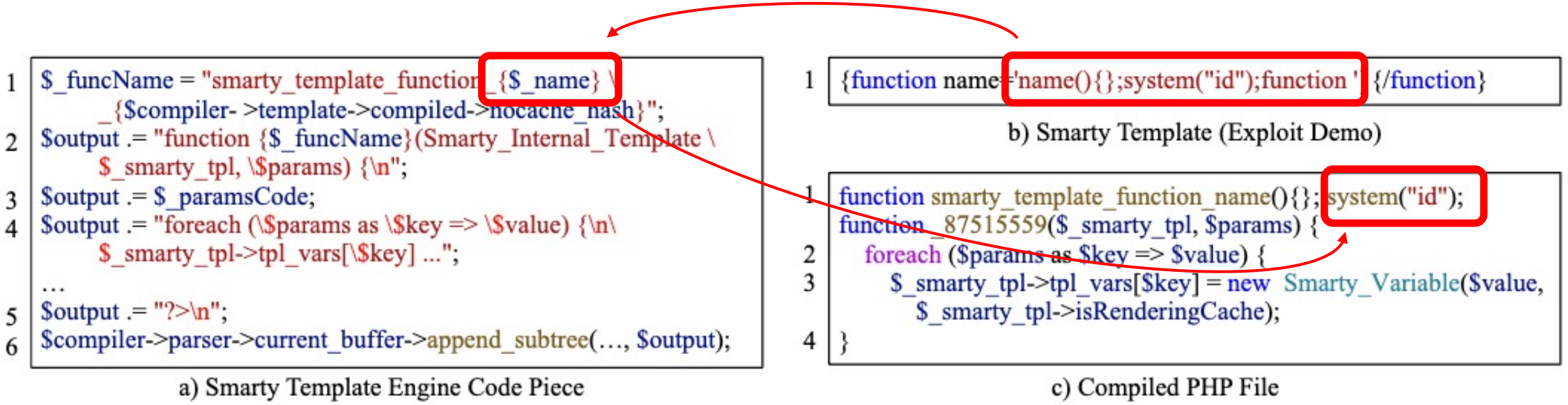



Figure 1: A Template Escape Bug in Smarty (CVE-2021-26120).

# Research Problem

- This work: an indepth study on template escape bugs
  - What is the cause of the template escape bug?
  - How to automatically detect and exploit template escape bugs?
  - What is the severity and prevalence of template escape bugs in real world?

# Challenges

- Challenge-I: It requires a fine-grained analysis of the template input.
  - different TEs have their specific grammar
  - it is hard to learn the syntax of the template input
- Challenge-II: It requires a specific payload to trigger and exploit such bug.
  - only carefully-constructed payloads could trigger a template escape bug
  - synthesizing an exploit is also quite challenging
- Challenge-III: There lacks an oracle for identifying template escape bugs.
  - even if an input successfully exploits a template escape bug, it is hard to judge whether the generated PHP file has been injected with executable code

# Approach Overview

- TEFuzz
  - a testing framework for different TEs
  - create testcases to discover and exploit template escape bugs
- Design Principles
  1. Balancing Exploration and Exploitation
     - Probing-based Interesting Testcase Identification
     - PHP Syntax-Guided PoC Generation
  2. Improving Code Coverage while Avoiding Redundant Testing
     - Testcase Adaption by Leveraging Error Feedback
     - Testcase Clustering by Leveraging Runtime Information

# Approach Overview

## Testcase

```
{block name="title" append}Page Title{/block}
```

*Probe each location to find interesting points*

**Escape Point (EP)**

```
{block name="Un1QuEtitle" append}Page Title{/block}
```

```
<?php
$_smarty_tpl->inheritance->instanceBlock($_smarty_tpl,
    'Block_440558559642af93c8ee782_96314246', "Un1QuEtitle");

/* {block "Un1QuEtitle"} */
class Block_440558559642af93c8ee782_96314246 extends Smarty_Internal_Block
{
    public $subBlocks = array (
    'Un1QuEtitle' => array (
        0 => 'Block_440558559642af93c8ee782_96314246',
    ),);
```

**Escape Context (EC):** *comments*

## Interesting Testcase

*a testcase contains at least one EP*

# Approach Overview

## Interesting Testcase

```
{block name="Un1QuEtitle" append}Page Title{/block}
```

## PoC

*Use PHP syntax string to replace the payload*

```
{block name="*/title" append}Page Title{/block}
```

```
<?php
$_smarty_tpl->inheritance->instanceBlock($_smarty_tpl,
    'Block_440558559642af93c8ee782_96314246', "*/title");

/* {block "*/title"} */
class Block_440558559642af93c8ee782_96314246 extends Smarty_Internal_Block
{
...
```

## Exploit

*Wrap the payload according to the escape context*

```
{block name="*/phpinfo();/*title" append}Page Title{/block}
```

```
<?php
$_smarty_tpl->inheritance->instanceBlock($_smarty_tpl,
    'Block_440558559642af93c8ee782_96314246', "*/phpinfo();/*title");

/* {block "*/phpinfo();/*title"} */
class Block_440558559642af93c8ee782_96314246 extends Smarty_Internal_Block
{
...
```

# Approach Overview (Workflow)

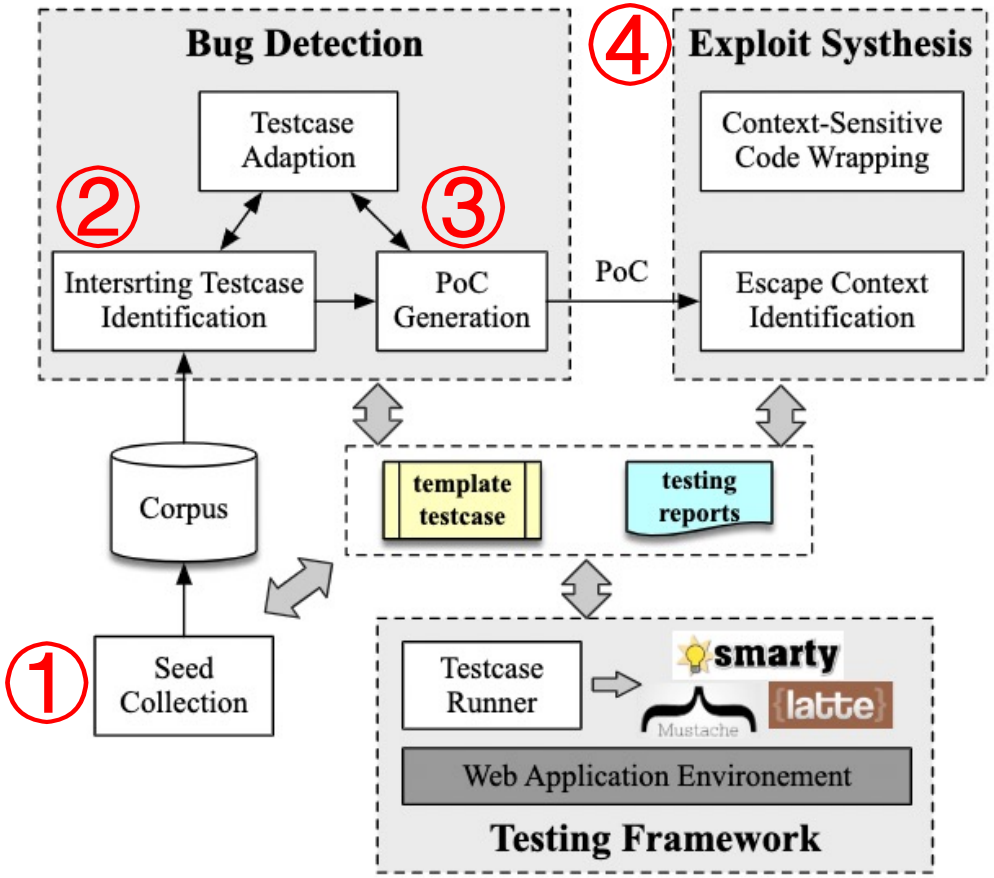



Figure 2: Overall Architecture of the TEFuzz Framework.

- Step 1: Seed Collection
- Step 2: Interesting Testcase Identification
- Step 3: PoC Generation
- Step 4: Exploit Synthesis

# Experimental Setup

- TE Dataset

Table 1: Dataset of the Target TEs and Their Basic Information.

| TE Name | Version | Stars | LoC | Mitigation | Delimeter | # of Seeds | # of Adaption Rules |
|---|---|---|---|---|---|---|---|
| Smarty | v3.1.39 | 2k | 25,986 | Sandbox | {,}; {*,*} | 523 | 13 |
| Twig | v3.3.1 | 7.5k | 18,378 | Sandbox | {{,}}; {%,%}; {#,#} | 339 | 9 |
| Dwoo | v1.3.7 | 168 | 80,405 | Sandbox | {,}; {*,*} | 208 | 4 |
| Latte | v2.10.5 | 802 | 6,949 | Sandbox | {,}; {*,*} | 289 | 5 |
| Mustache | v2.14.0 | 3.1k | 6895 | No PHP Execution | {{,}}; {{!,}} | 17 | 2 |
| Fenom | v2.12.1 | 431 | 11,974 | Sandbox | {,}; {*,*} | 181 | 4 |
| ThinkPHP | v6.0.12 | 2.4k | 2,280 | Sandbox | {,}; {/,/} | 171 | 1 |

- Seed Collection
  - Collected 1,728 testcases as the initial seeds from official documents and the testing files in its source code

# Research Questions


- RQ1: How prevalent are template escape bugs?
- RQ2: How severe are template escape bugs?
- RQ3: How does TEFuzz compare to SSTI scanners?
- RQ4: How feasible is exploiting template escape bugs in real-world applications?
- RQ5: How helpful are the internal designs of TEFuzz?

# RQ1: Prevalence

- Almost every TE has template escape bugs.

Table 2: Detected Bugs (RQ1 & RQ2).

| TE Name | Unique Bugs | Exploitable Bugs | RCE? |
|---|---|---|---|
| Smarty | 3 | 3 | ✓ |
| Twig | 0 | 0 | |
| Latte | 49 | 24 | ✓ |
| Mustache | 1 | 1 | ✓ |
| Dwoo | 38 | 2 | ✓ |
| Fenom | 10 | 10 | ✓ |
| ThinkPHP | 34 | 15 | ✓ |
| All | 135 | 55 | |

# RQ2: Severity

- TEFuzz successfully generates RCE exploits for 55 template escape bugs.

Table 2: Detected Bugs (RQ1 & RQ2).

| TE Name | Unique Bugs | Exploitable Bugs | RCE? |
|---|---|---|---|
| Smarty | 3 | 3 | ✓ |
| Twig | 0 | 0 | |
| Latte | 49 | 24 | ✓ |
| Mustache | 1 | 1 | ✓ |
| Dwoo | 38 | 2 | ✓ |
| Fenom | 10 | 10 | ✓ |
| ThinkPHP | 34 | 15 | ✓ |
| All | 135 | 55 | |

# RQ3: Comparison

- Baseline: tplmap[1]
  - We have enhanced tplmap to support TEs in our dataset.
- Results
  - Tplmap only discovers template injection points, but fails to bypass the TE sandbox.
  - With the RCE payloads generated by TEFuzz, tplmap successfully breaks the TE sandbox.

Table 3: Comparison Results between *tplmap* and TEFUZZ.

| TE Name | Version | *tplmap* | | | *tplmap* + TEFUZZ | | |
|---|---|---|---|---|---|---|---|
| | | SSTI | Escape[1] | RCE | SSTI | Escape[1] | RCE |
| Smarty | v3.1.39 | ✓ | × | × | ✓ | ✓ | ✓ |
| Twig | v3.3.1 | ✓ | × | × | ✓ | × | × |
| Dwoo | v1.3.7 | ✓ | × | × | ✓ | ✓ | ✓ |
| Latte | v2.10.5 | ✓ | × | × | ✓ | ✓ | ✓ |
| Mustache | v2.14.0 | ✓ | × | × | ✓ | ✓ | ✓ |
| Fenom | v2.12.1 | ✓ | × | × | ✓ | ✓ | ✓ |
| ThinkPHP | v6.0.12 | ✓ | × | × | ✓ | ✓ | ✓ |

[1] Triggering a template escape bug

[1] https://github.com/epinna/tplmap

# RQ4: Full Exploitation

1. Searching Known Vulnerabilities
   - Search keywords in the CVE database, and read security blogs and vulnerability reports.

   - Find 5 vulnerabilities that use a vulnerable TE in our dataset.
   - Achieve the full exploitation on all of them:
     - Smarty: CVE-2020-35625 ,CVE-2017-16783, CVE-2017-6070, CVE-2020-15906
     - ThinkPHP: CVE-2020-25967

# RQ4: Full Exploitation

2. Discovering 0-day Vulnerabilities
   - Collect 18 PHP applications that use a vulnerable TE in our dataset.
   - Experiment-I: tplmap + carwlergo + TEFuzz
     - find 0 vulnerability
   - Experiment-II: manual discovery
     - find 6 vulnerabilities

Table 5: Manually Discovered SSTI Vulnerabilities in Real-world PHP Applications.

| Application | Version | Stars | TE | RCE | Root Cause |
|---|---|---|---|---|---|
| CMSMS | 2.2.16 | | Smarty | ✓ | Normal Functionality of Template Modification |
| lmxcms | 1.41 | | Smarty | ✓ | Normal Functionality of Template Modification |
| Piwigo | 13.0.0 | 2.2k | Smarty | ✓ | File Upload to Template Overwrite + Normal Functionality of Template Selection |
| MediaWiki | 1.38.2 | 3.2k | Smarty | ✓ | Normal Functionality of Template Modification |
| TikiWiki CMS | 21.7 | | Smarty | ✓ | Normal Functionality of Template Modification |
| Ejucms | SP4 | | ThinkPHP | ✓ | Normal Functionality of Template Modification |

# RQ5: Internal Results

- Testcase Probing
  - Collect 1,728 seeds
  - Creates 63,975 new testcases
  - Identify 5,070 unique interesting testcases
- PoC Generation
  - Create 630,518 new testcases
  - Identify 170 unique PoCs
  - Report 135 bugs

Table 6: Internal Results of TEFuzz in Generating Interesting Testcases and PoCs (RQ5).

| TE | Seeds | Testcase Probing | | PoC Generation | |
|---|---|---|---|---|---|
| | | Created Testcases | Interesting Testcases (A/U)[1] | Created Testcases | PoCs (A/U)[1] |
| Smarty | 523 | 16,662 | (10,570 / 907) | 132,198 | (20 / 3) |
| Twig | 339 | 16,290 | (3,734 / 1,466) | 196,310 | (0 / 0) |
| Dwoo | 208 | 9,335 | (9,335 / 930) | 108,397 | (401 / 46) |
| Latte | 289 | 8,930 | (5,635 / 721) | 74,280 | (924 / 63) |
| Mustache | 17 | 368 | (299 / 136) | 10,385 | (81 / 1) |
| Fenom | 181 | 5,444 | (3,276 / 396) | 52,708 | (37 / 10) |
| ThinkPHP | 171 | 6,946 | (5,994 / 534) | 56,240 | (165 / 47) |
| All | 1,728 | 63,975 | (38,843 / 5,070) | 630,518 | (1,628 / 170) |

[1] A(ll) represents the number of all the cases; U(nique) represents the number of unique cases.

These modules help TEFuzz avoid redundant testing and detect real vulnerabilities.

# RQ5: Internal Results

- Testcase Adaption
  - Testcase Fix Rate: 69.4%
    - Meets TE errors in 63,576 testcases and fixes 44,103 testcases
  - Help to collect 6.7% more seeds, discover 21.6% more bugs, and synthesize 31.0% more exploits.
- Exploit Synthesis
  - Synthesize 135 exploits, of which 55 ones are useful.
  - Failed exploits:
    1) The payloads used to wrap the escape context in the PHP file make the TE fail to parse the template code.
    2) TE raises errors when checking the format of the exploit.

# Conclusion


- We study an overlooked and severe sandbox bypass vulnerability in template engines and demonstrate its root cause.

- We present an automatic tool to detect and exploit template escape bugs and introduce several new techniques.

- We discover 135 bugs in seven PHP template engines and construct 55 exploits that enable RCE attacks.

# Thanks
# Q&A