Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning

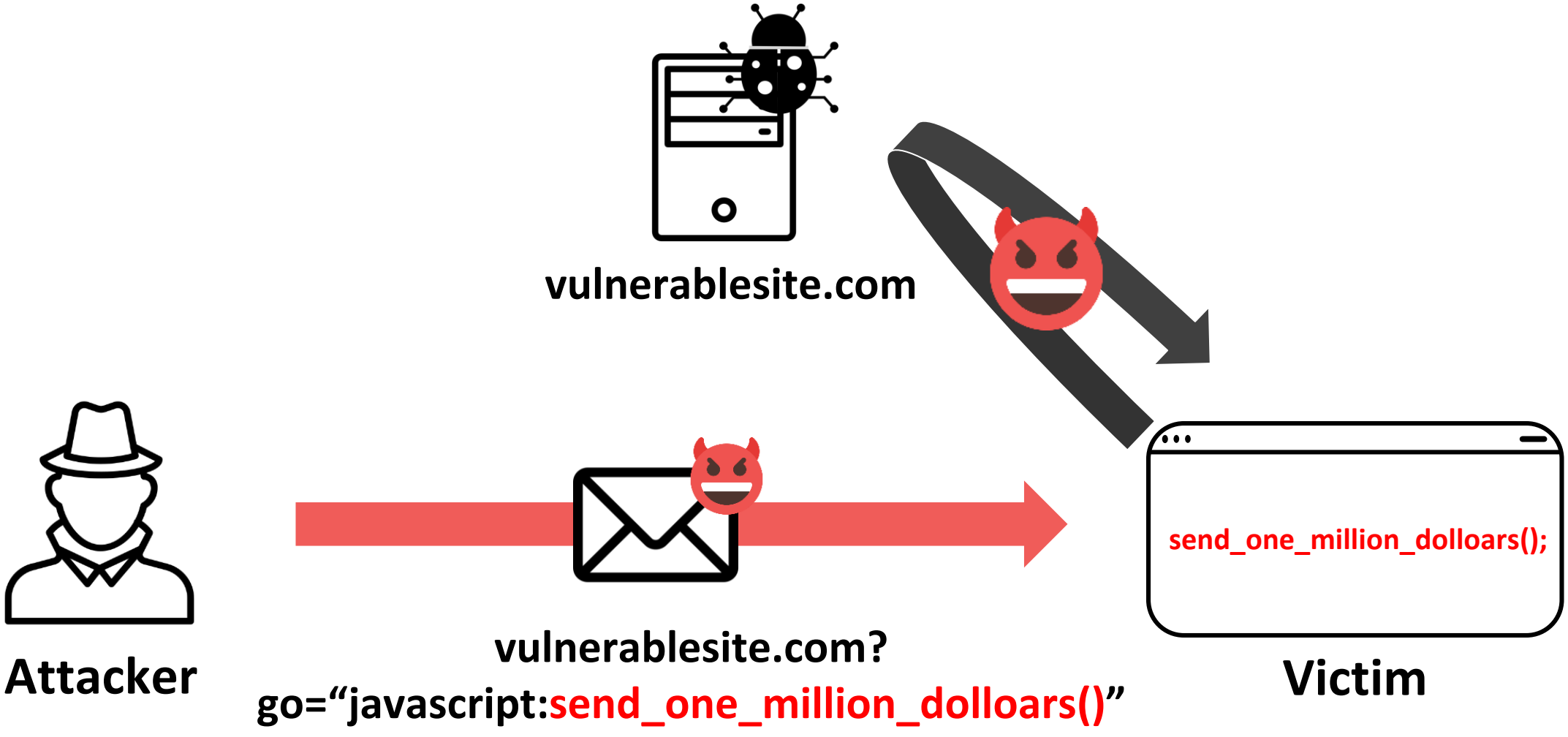
Soyoung Lee, Seongil Wi, Sooel Son

KAIST

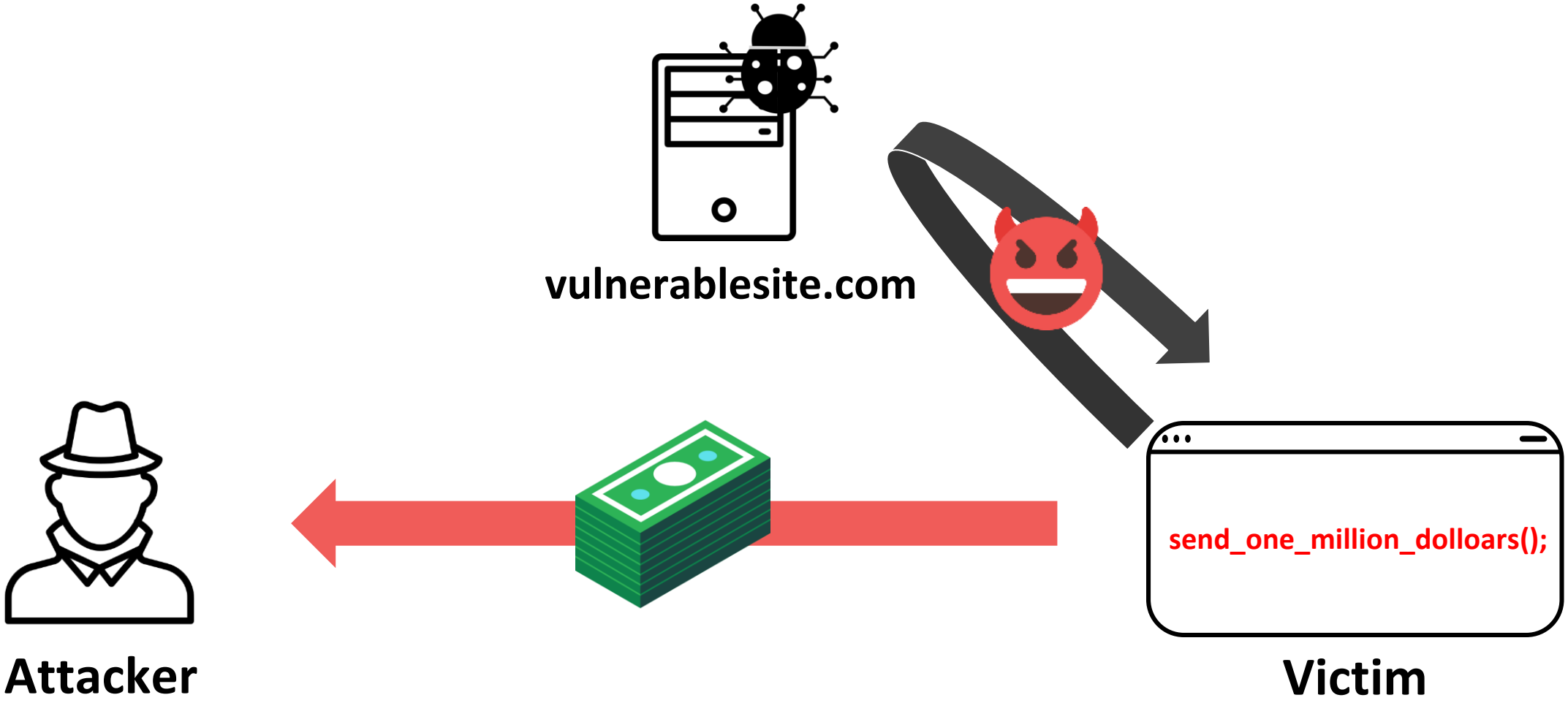
TheWebConf 2022

KAIST Web Security & Privacy Lab

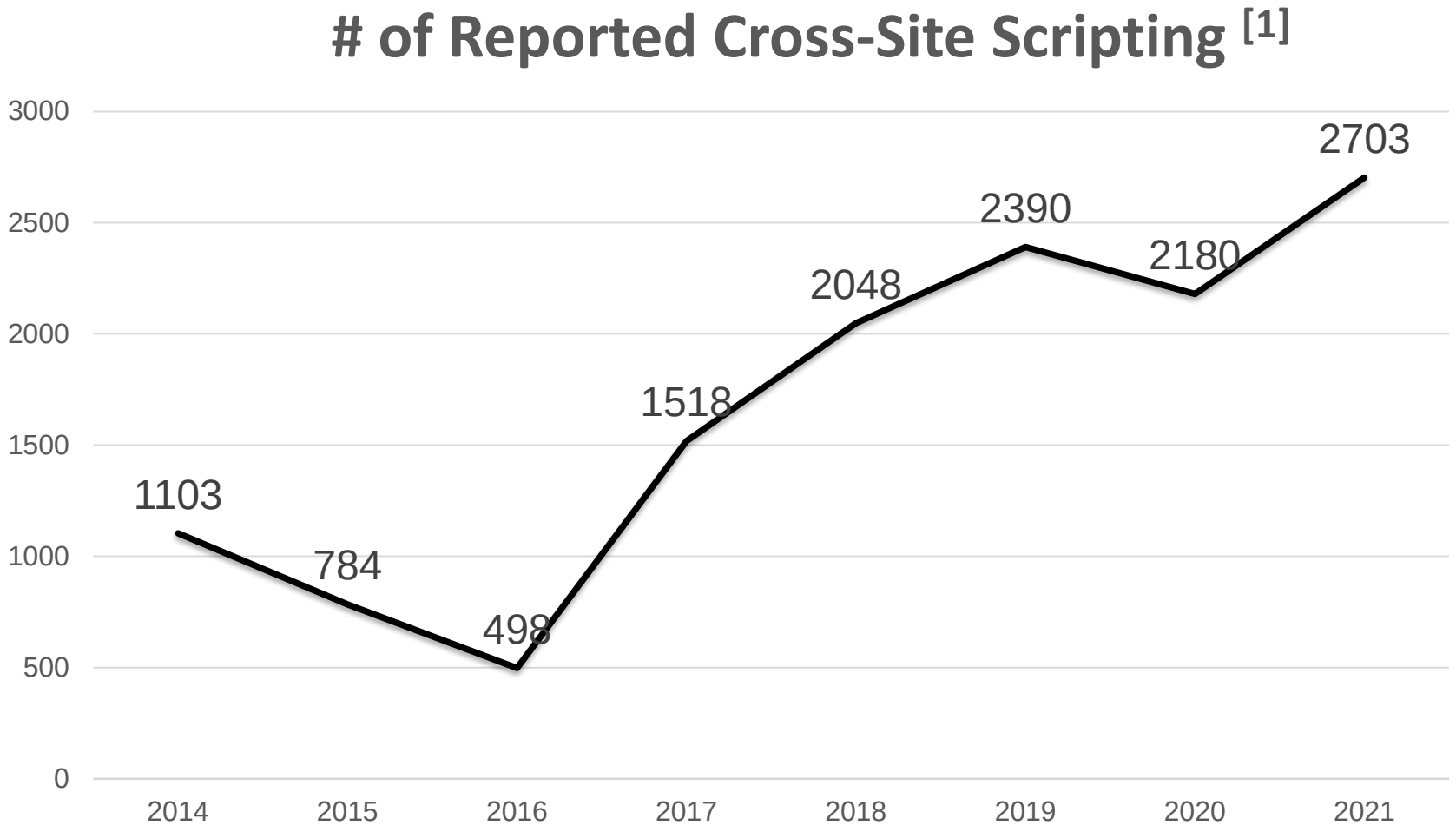
Reflected Cross-Site Scripting



Reflected Cross-Site Scripting

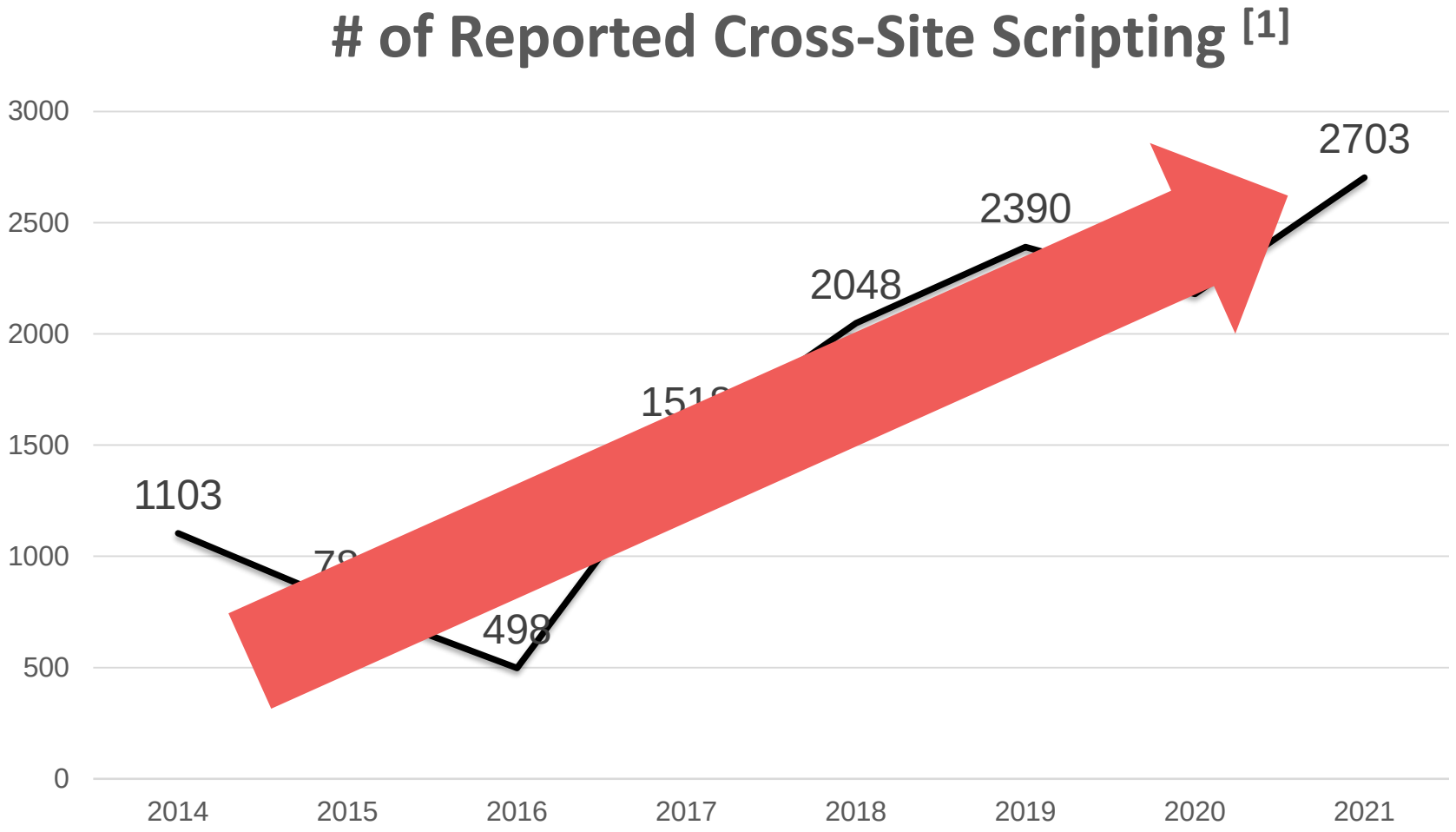


Cross-Site Scripting



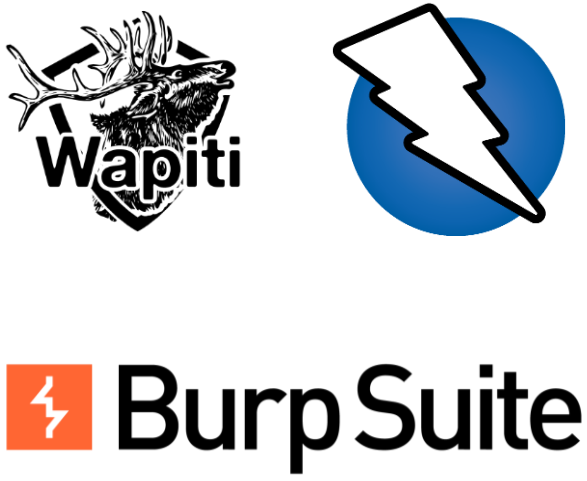
[1] https://www.cvedetails.com/vulnerabilities-by-types.php

Cross-Site Scripting

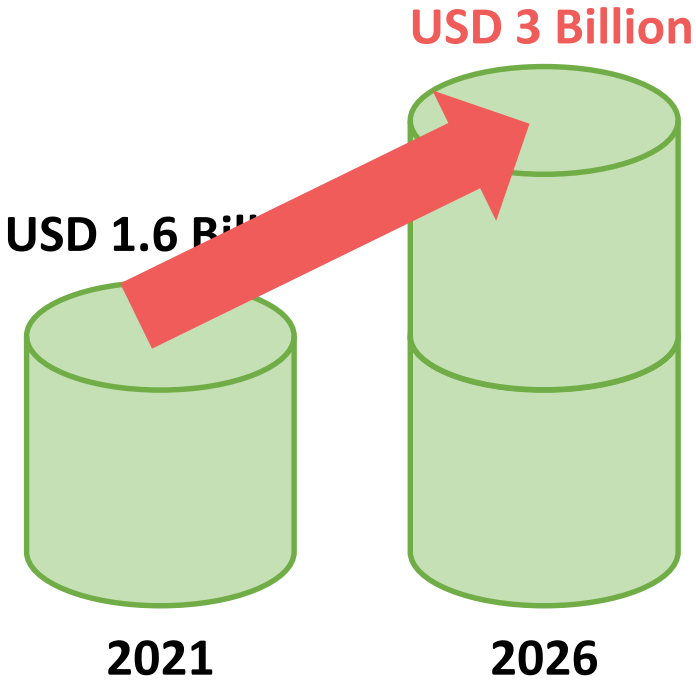


[1] https://www.cvedetails.com/vulnerabilities-by-types.php

XSS Detection - Penetration Testing



Penetration Testing tools



Market Size of Penetration Testing [2]

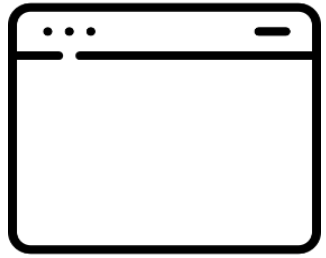
[2] https://www.researchandmarkets.com/reports/5403275/penetration-testing-market-by-offering

Penetration Testing

Attack scenario

Attacker

vulnerablesite.com?
go="javascript:send_one_million_dolloars()"



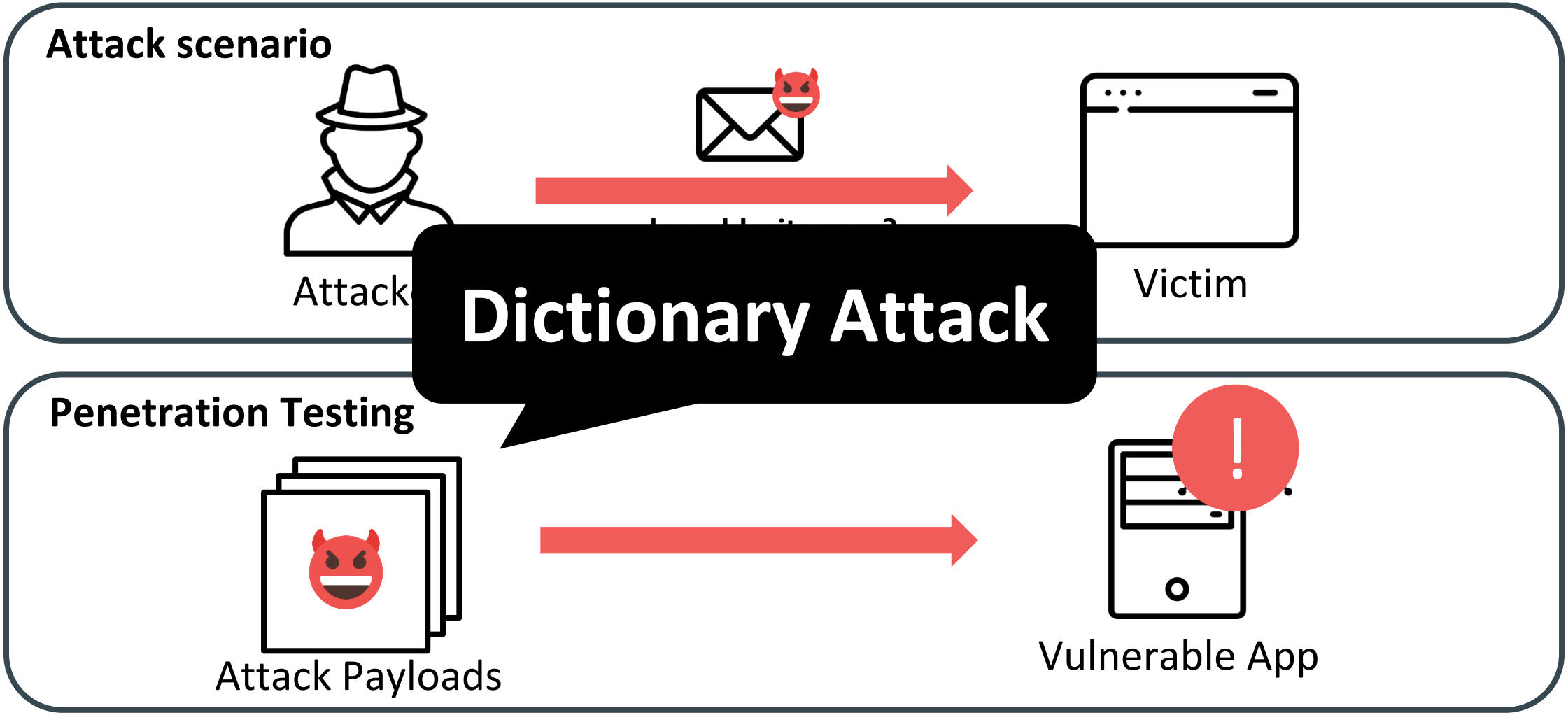
Victim

Penetration Testing

Attack Payloads

Vulnerable App

Penetration Testing



Penetration Testing - Dictionary Attack

Vulnerable Target Application

```php
<?php
    $input = removeHTMLTagName($_GET["go"]);
    echo $input;
?>
```

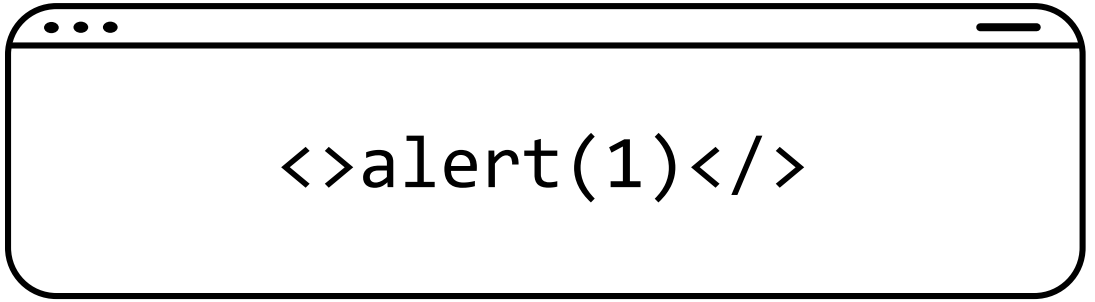
vulnerablesite.com?go=

Attack Payload Dictionary	
1	<script>alert(1)</script>
2	</script><script>alert(1)</script>
3	
4	<scrscriptipt>alert(1);</scrscriptipt>

Working Exploit payload

<scrscriptipt>alert(1);</scrscriptipt>

Browser (Response)



KAIST Web Security & Privacy Lab

Penetration Testing - Dictionary Attack

Vulnerable Target Application

```
<?php
    $input = removeHTMLTagName($_GET["go"]);
    echo $input;
?>
```

vulnerablesite.com?go=

Working Exploit payload

<scrscriptipt>alert(1);</scrscriptipt>

Attack Payload Dictionary	
1	<script>alert(1)</script>
2	</script><script>alert(1)</script>
3	
4	<scrscriptipt>alert(1);</scrscriptipt>

Browser (Response)

<script>alert(1);</script>

Script is executed! (Vulnerability)

Disadvantages #1 – False negative

Vulnerable Target Application

```php
<?php
    $input = removeHTMLTagName($_GET["go"]);
    echo $input;
?>
```

Attack Payload Dictionary	
1	<script>alert(1)</script>
2	</script><script>alert(1)</script>
3	

Working Exploit payload

<scrscriptipt>alert(1);</scrscriptipt>

No exploit payload in the dictionary!

Detection fails (False negative)

Disadvantages #2 - Request overhead

Vulnerable Target Application

```php
<?php
    $input = removeHTMLTagName($_GET["go"]);
    echo $input;
?>
```

Working Exploit payload

```
<scrscriptipt>alert(1);</scrscriptipt>
```

Attack Payload Dictionary	
Unnecessary 1000 Requests!	
1001	<scrscriptipt>alert(1);</scrscriptipt>

Request overhead 

Why disadvantages?

1. **Predefined attack payload dictionary**

2. **No consideration about the target**

Why disadvantages?

Context-unaware Payload

What we want?

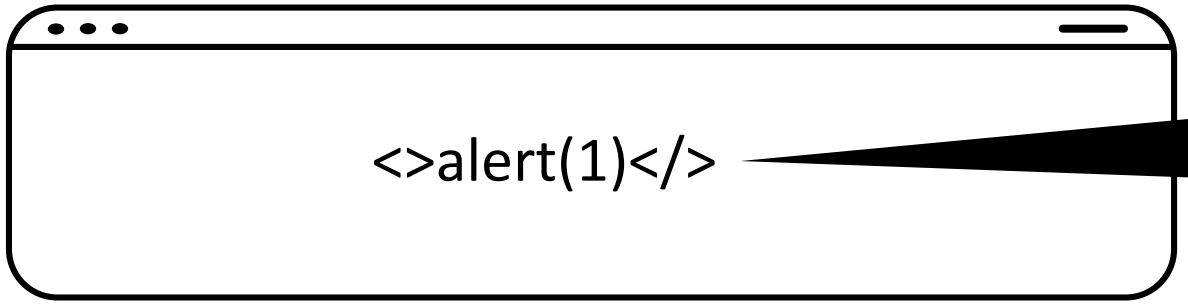
- Context-aware payloads

Vulnerable Target Application

```php
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
```

Attempt #1: <script>alert(1)</script>

Browser (Response)



“script” is removed

What we want?

- Context-aware payloads

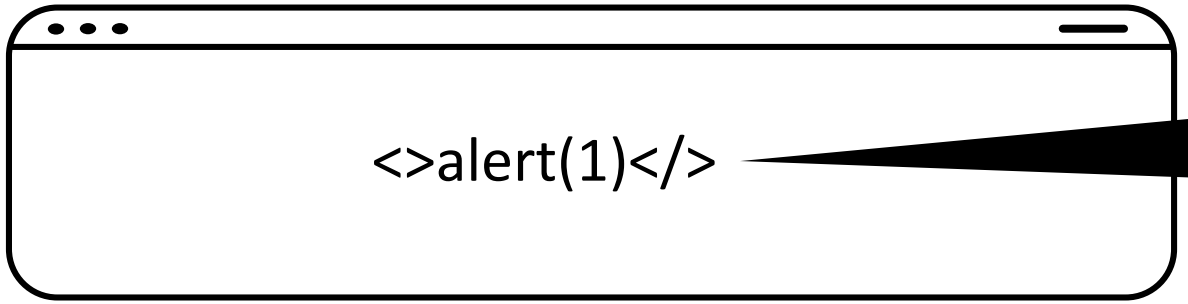
Vulnerable Target Application

```
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
```

Attempt #1: <script>alert(1)</script>

Let's insert "script"

Browser (Response)



"script" is removed

KAIST Web Security & Privacy Lab

What we want?

- Context-aware payloads

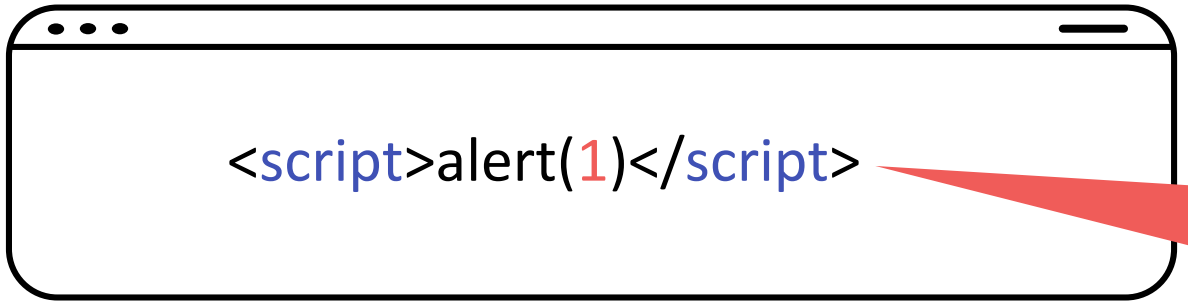
Vulnerable Target Application

```php
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
```

Attempt #1: <script>alert(1)</script>

Attempt #2:
<scrscriptipt>alert(1)</scrscriptipt>

Browser (Response)

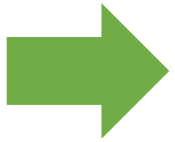


Script is executed! (Vulnerability)

How can we do this?

- Heuristics?

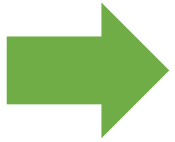
Attempt #1: <script>alert(1)</script>
Response: <>alert(1)</>



Attempt #2:
<scrscriptipt>alert(1)</scrscriptipt>

Feedback

If “ script” is removed

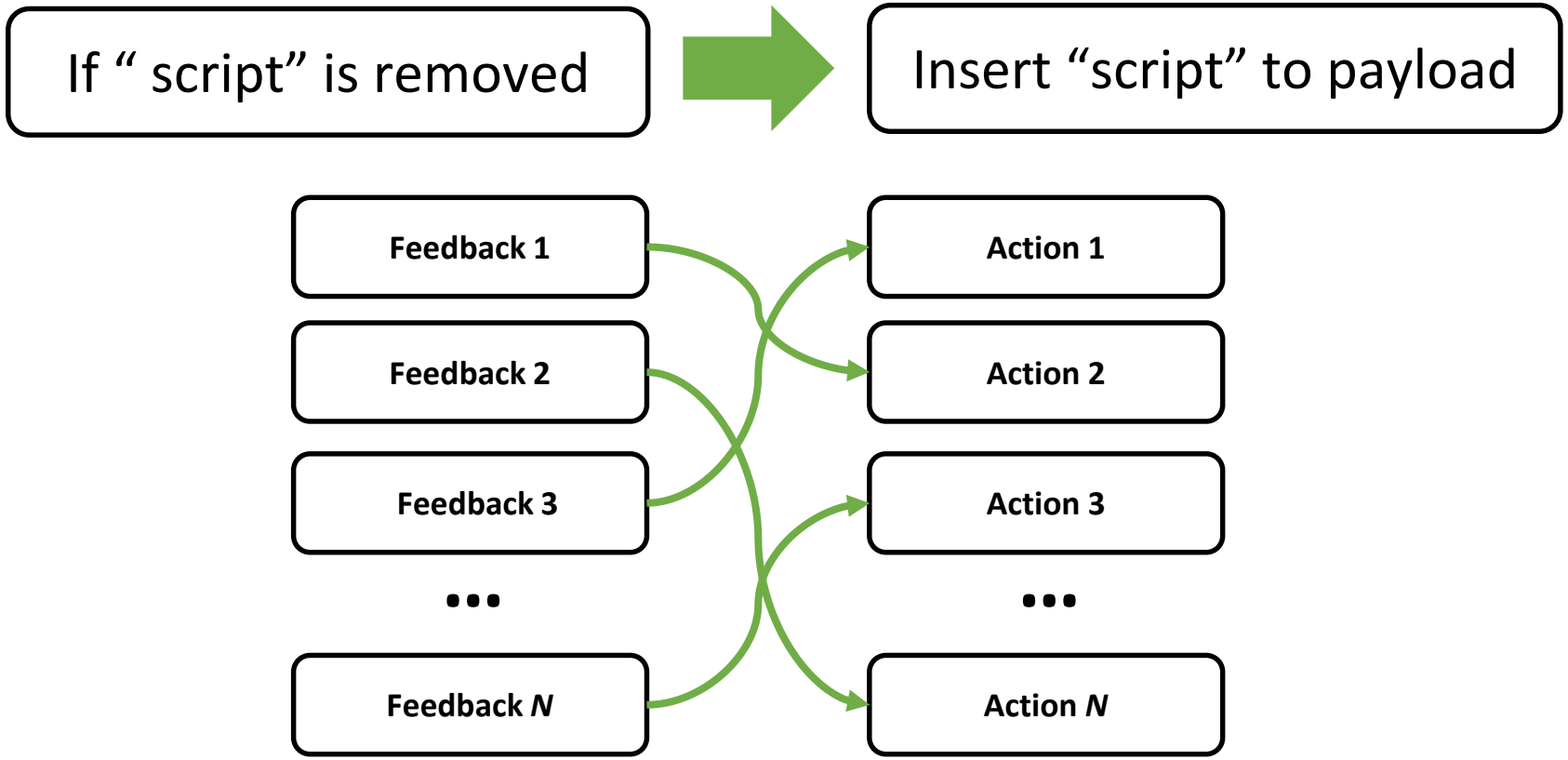


Action

Insert “script” to payload

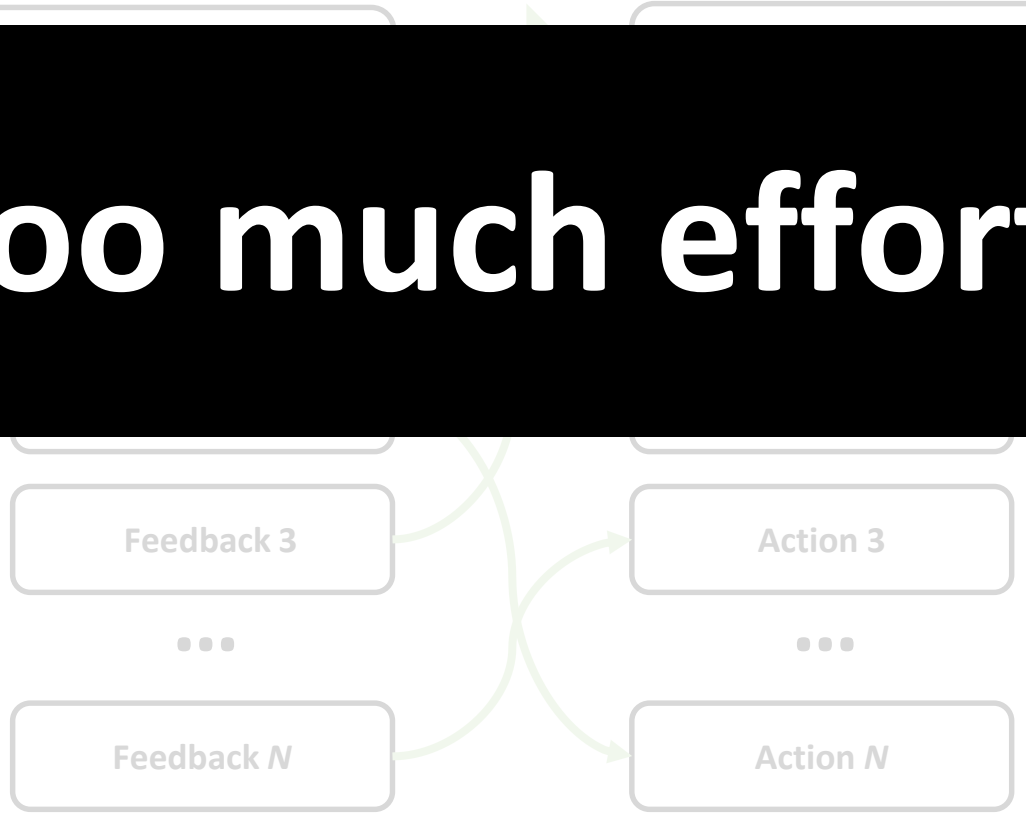
How can we do this?

- It is only one rule!



How can we do this?

- Heuristics?

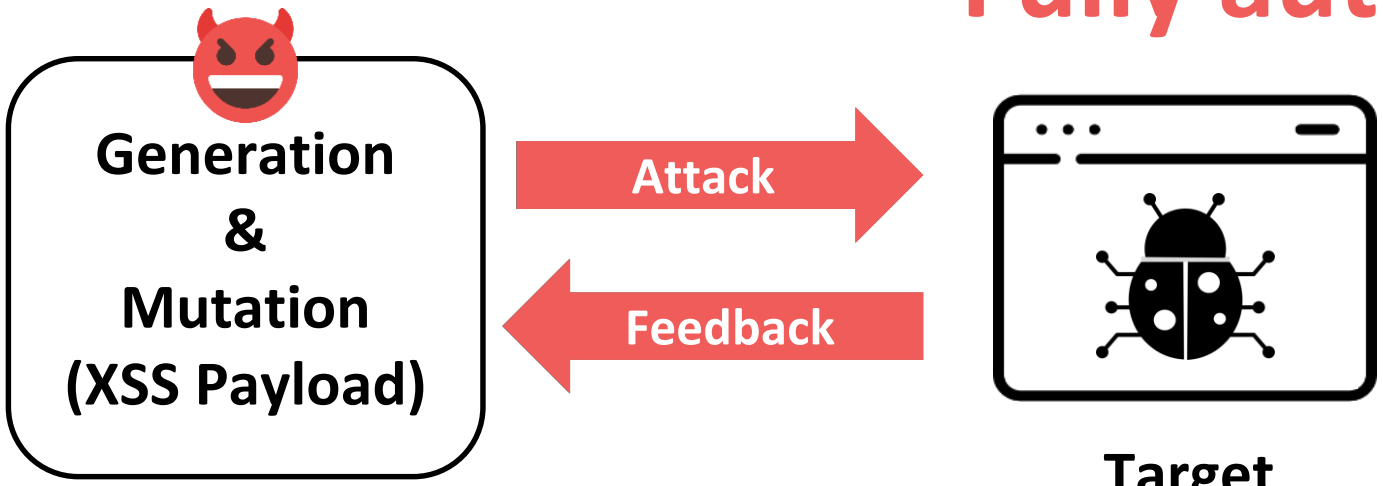


Too much effort!

We propose **Link**

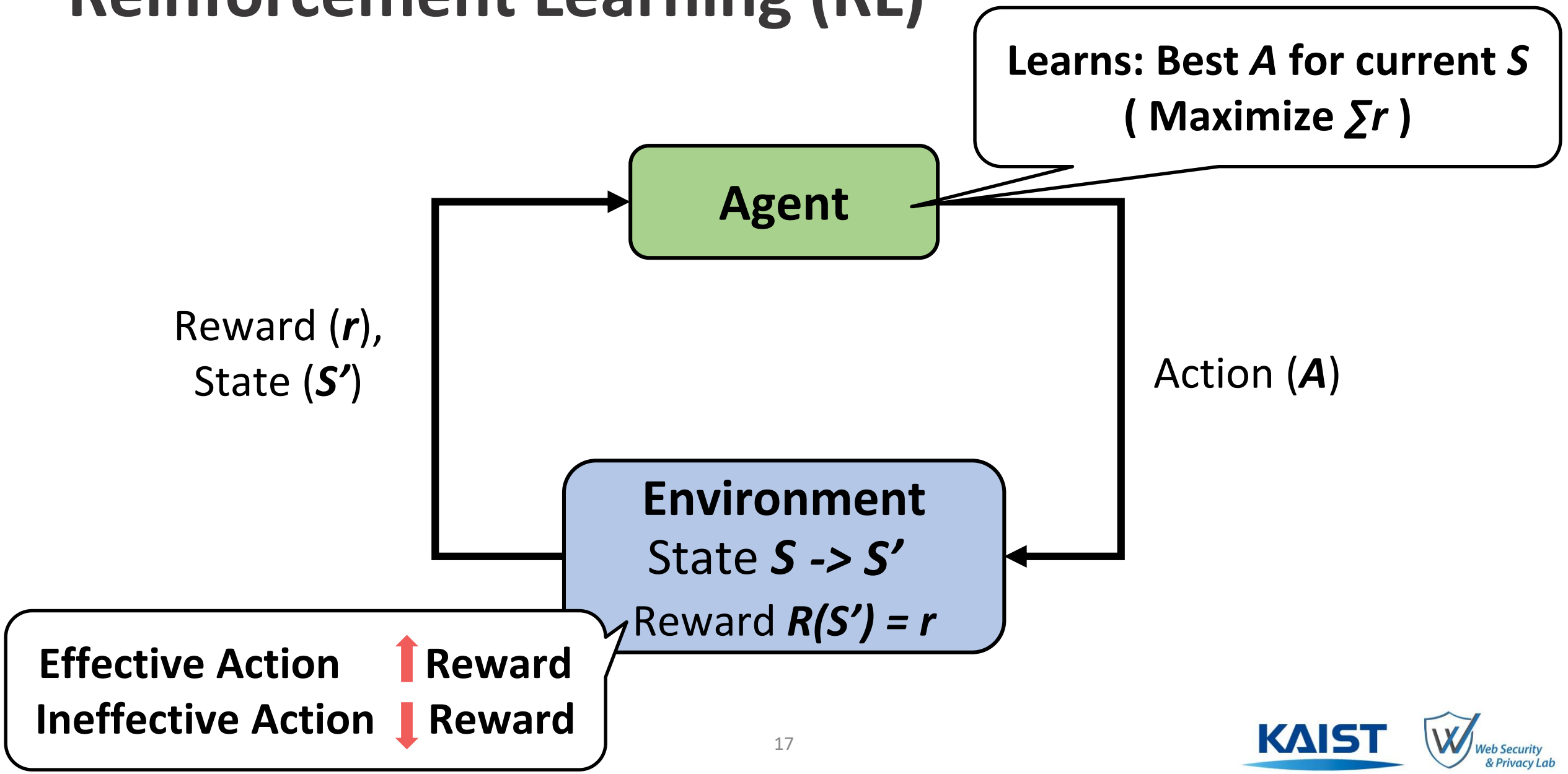
- **Reinforcement Learning**
- Context-aware payloads

Fully automated!

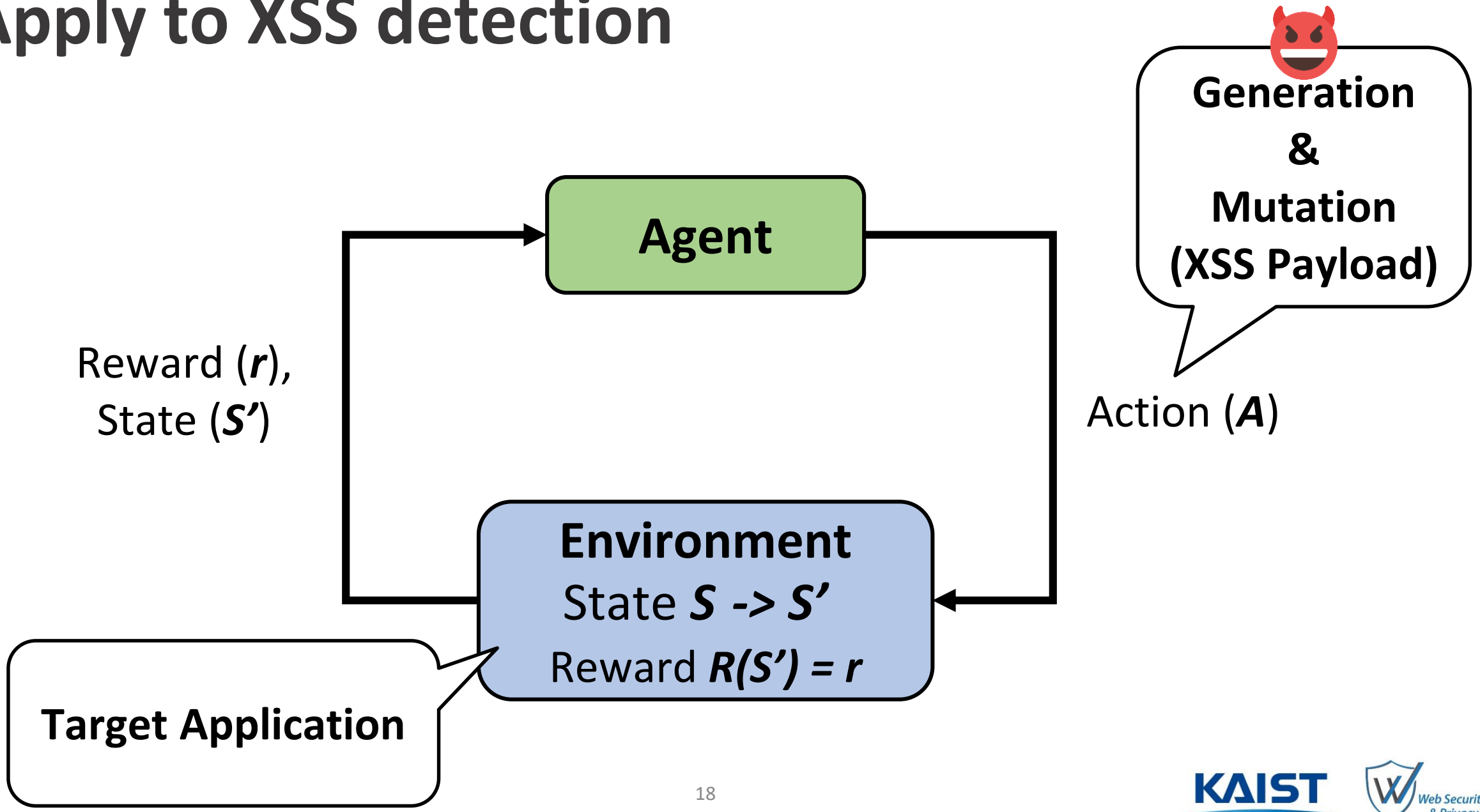


KAIST Web Security & Privacy Lab

Reinforcement Learning (RL)



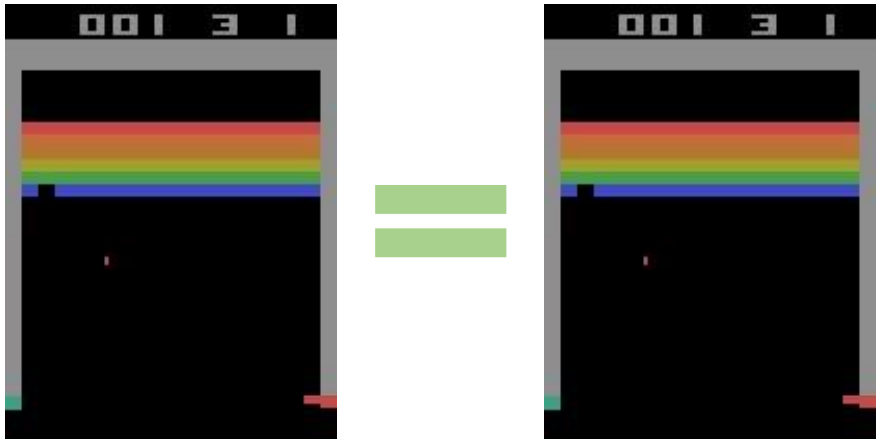
Apply to XSS detection



Challenges #1: Transferable RL agent

- **Training & Testing application**

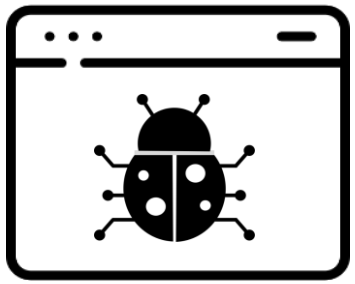
Training



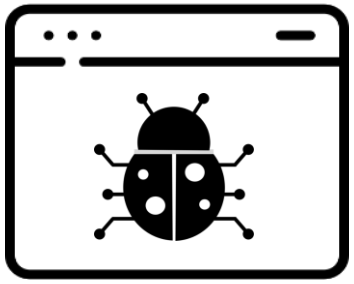
Testing

Game

Training



Testing



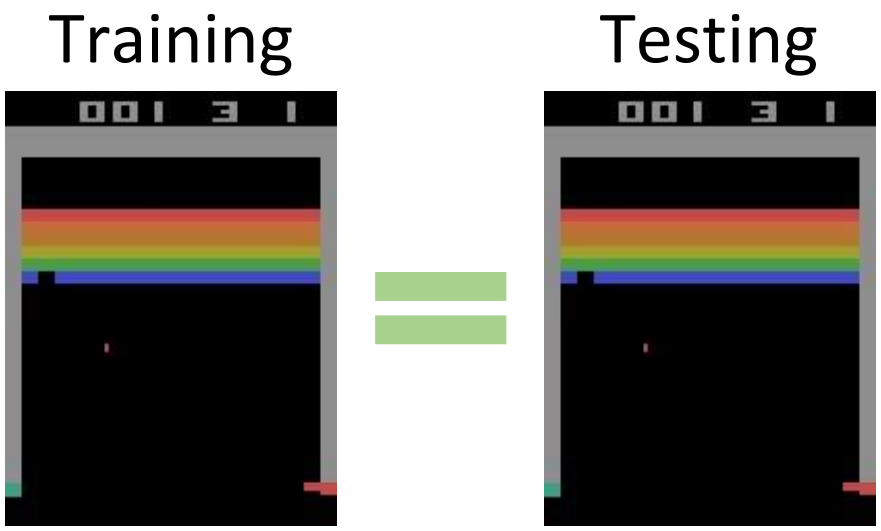
Training Application

Testing Application

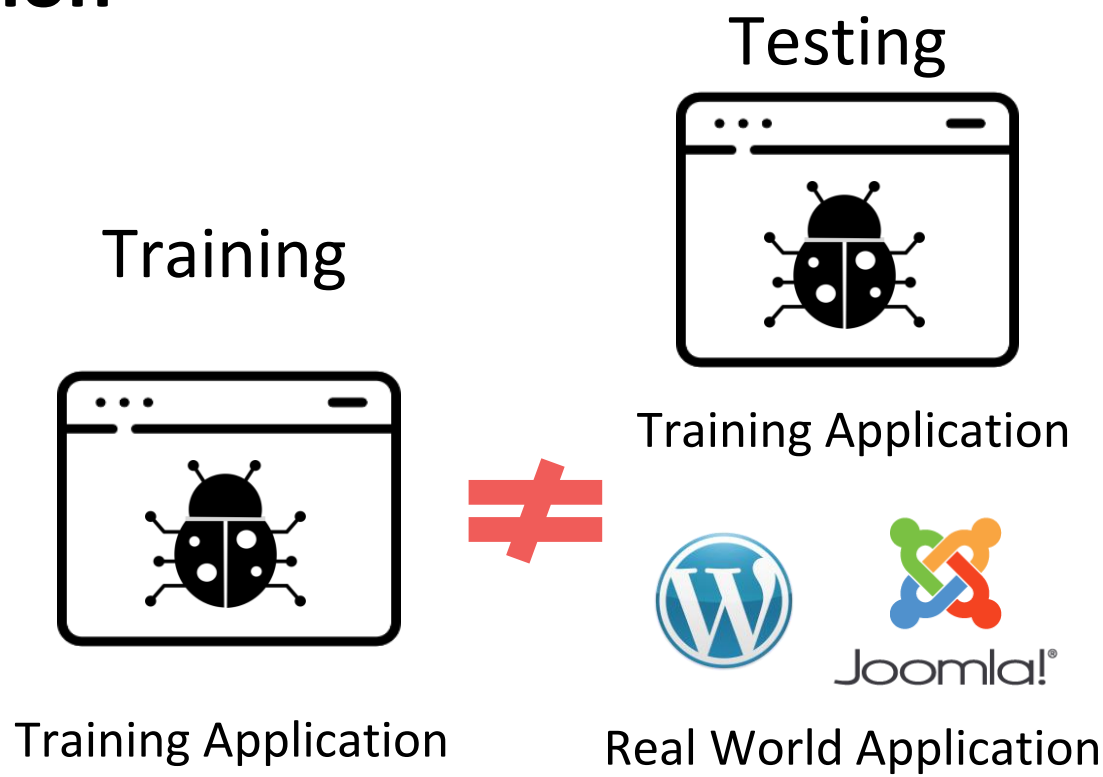
XSS Detection

Challenges #1: Transferable RL agent

- **Training & Testing application**



Game



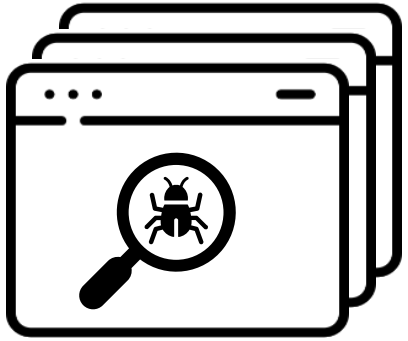
XSS Detection

KAIST Web Security & Privacy Lab

Challenges #1: Transferable RL agent

- **Our training application**
  - General enough to cover diverse cases

XSS Scanner Evaluation Application
(i.e. Firing-Range, WAVSEP)

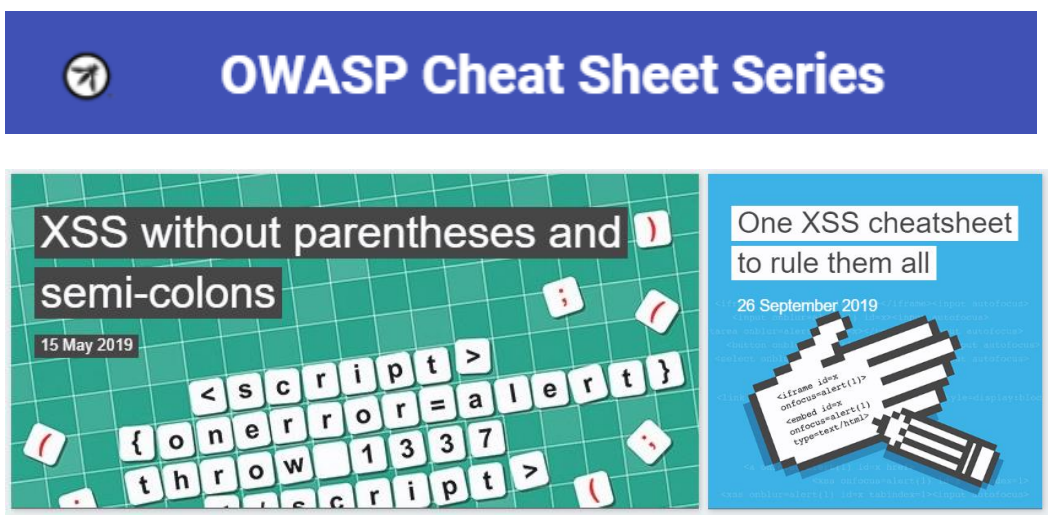


Filter Evasion Application

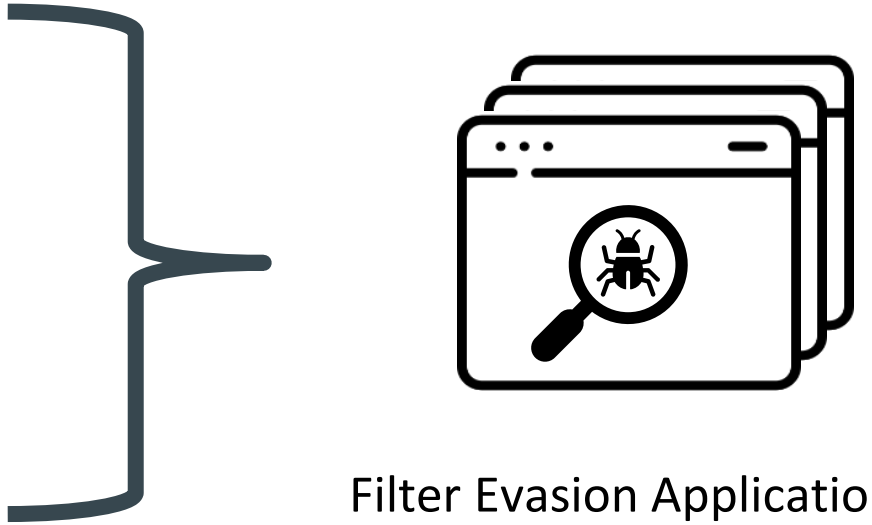
(Cannot be covered by former)

Challenges #1: Transferable RL agent

- We implement **Filter Evasion Application**



Evasion Techniques[4]

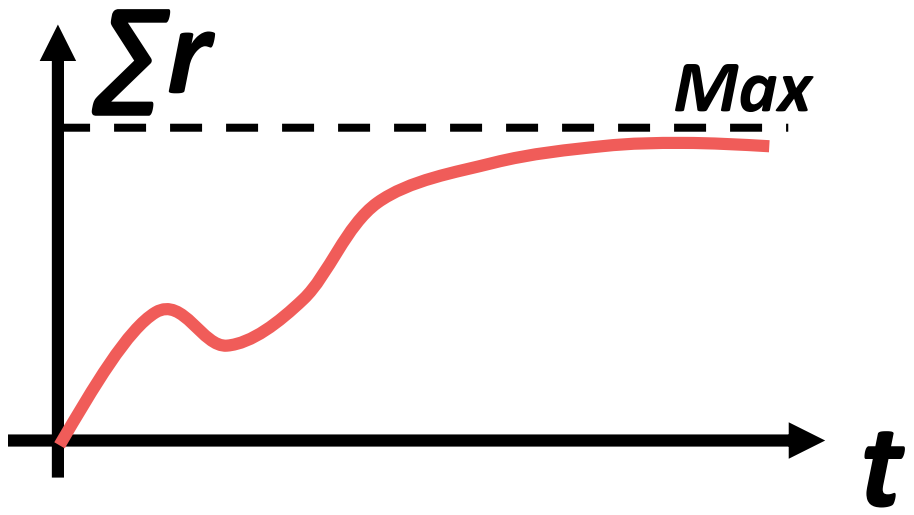


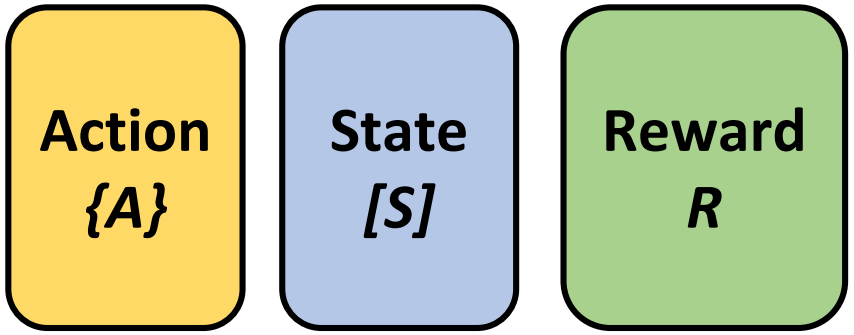
Filter Evasion Application

[4] https://portswigger.net/research/cross-site-scripting-research

Challenges #2: Non-convergence problem

- **Non-convergence problem**
  - Known problem of reinforcement learning
  - Depends on design

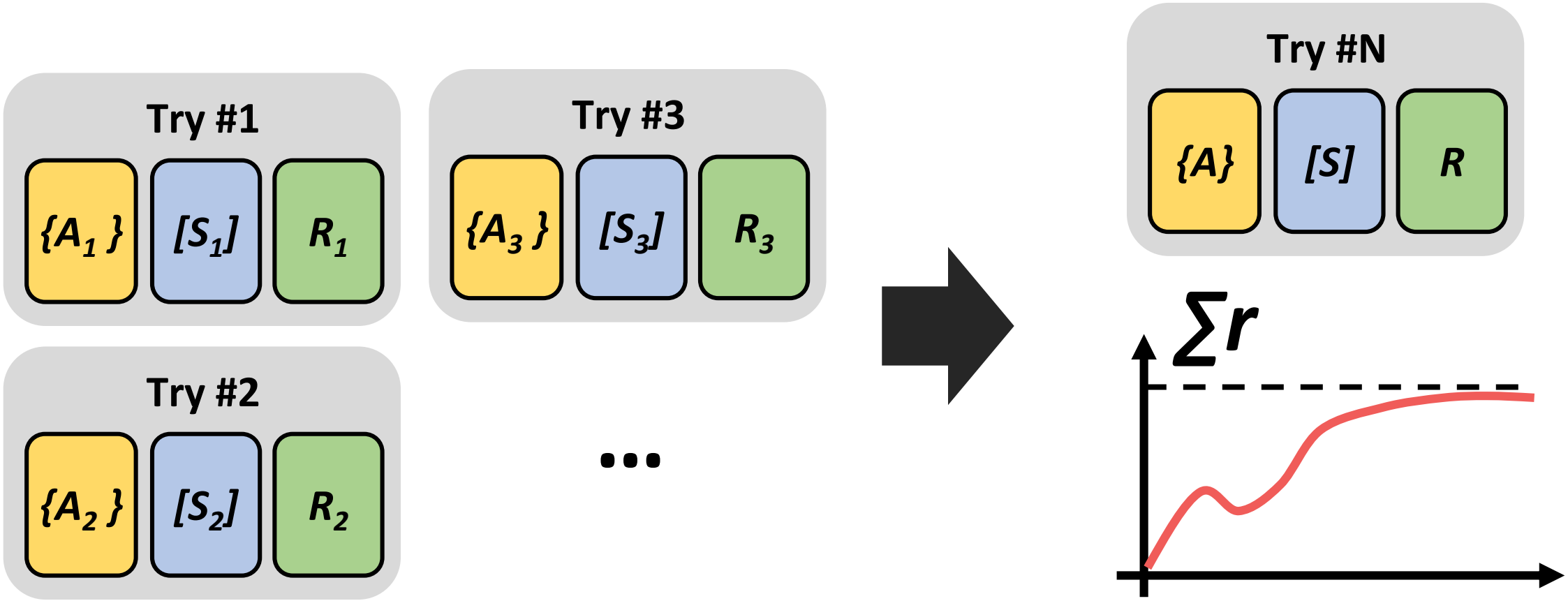




We should properly model these components!

Challenges #2: Non-convergence problem

- **Engineering Approach**



KAIST Web Security & Privacy Lab

Design - Action

Action is the attack payload **Generation** and **Mutation** rule (39 features)

- **Generation (7 rules)**
  - Basic payload for further mutation

- **Mutation (32 rules)**
  - Mutate basic payloads

Design - Action

- **Generation – Basic Payloads**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

We need first payload!

Browser (Response)

```
<textarea>

</textarea>
```

Design - Action

- **Generation – Basic Payloads**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
          <>alert(1)</>
</textarea>
```

Generation Action:

Generate payload with “script” tag

Attempt #1: <script>alert(1)</script>

“script” removed

Design - Action

- **Generation – Basic Payloads**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
          <>alert(1)</>
</textarea>
```

Generation Action:

Generate payload with “script” tag

Attempt #1: <script>alert(1)</script>

We need mutation

Design - Action

- **Mutation – Evasion technique**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
        <script>alert(1)</script>
</textarea>
```

Mutation Action:

Overlap the tag string

Attempt #1: <script>alert(1)</script>

Attempt #2:

<scrscriptipt>alert(1)</scrscriptipt>

Design - Action

- **Mutation – Evasion technique**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
        <script>alert(1)</script>
</textarea>
```

Mutation Action:

Overlap the tag string

Attempt #1: <script>alert(1)</script>

Attempt #2:

<scrscriptipt>alert(1)</scrscriptipt>

Script is not executed!

Design - Action

- **Mutation – Escaping Technique**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
 </textarea><script>alert(1)</script>
</textarea>
```

Mutation Action:

Prepend HTML tag

Attempt #1: <script>alert(1)</script>

Attempt #2:

<scrscriptipt>alert(1)</scrscriptipt>

Attempt #3:

</textarea>

<scrscriptipt>alert(1)</scrscriptipt>

Design - Action

- **Mutation – Escaping Technique**

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Mutation Action:

Prepend HTML tag

Attempt #1: <script>alert(1)</script>

Attempt #2:

Browser (Response)

```
<textarea>
 </textarea><script>alert(1)</script>
</textarea>
```

Script is executed! (Vulnerability)

KAIST Web Security & Privacy Lab

Design - State

- **Information for the agent (47 features)**

$$[\ s_0\ ,\ s_1\ ,\ s_2\ ,\ \dots\ ,\ s_{45}\ ,\ s_{46}\ ]$$

Feature: Information feature about the environment (human knowledge)
Value: Numeric values (what agent see)

1. Information about the input payload
2. Information about the Injection point
3. Information about the attack result

Design - State

1. Payload information

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Attempt #1: <script>alert(1)</script>

Feature 3: Use <script> HTML Tag
Value: 1 (True)

Browser (Response)

```
<textarea>
          <>alert(1)</>
</textarea>
```

Design - State

1. Payload information

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
          <>alert(1)</>
</textarea>
```

Attempt #1: <script>alert(1)</script>

Feature 3: Use <script> HTML Tag
Value: 1 (True)

Feature 4: Use script alert(1)
Value: 1 (True)

Design - State

2. Injection point information

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Attempt #1: <script>alert(1)</script>

Browser (Response)

```
<textarea>
        <>alert(1)</>
</textarea>
```

Injection Point

Design - State

2. Injection point information

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
        <>alert(1)</>
</textarea>
```

Attempt #1: <script>alert(1)</script>

Feature1 : What is before injection point?
Value: 3 (bracket “>”)

Feature2: Where is the attack payload injected?
Value: 6 (Between the <textarea> Tags)

Design - State

3. Information about the attack result

Vulnerable Target Application

```
<textarea>
<?php
   $input = removeHTMLTagName($_GET["go"]);
   echo $input;
?>
</textarea>
```

Browser (Response)

```
<textarea>
        <>alert(1)</>
</textarea>
```

Attempt #1: <script>alert(1)</script>

Feature 5: HTML Tag is reflected
Value: 0 (False)

Feature 6: Script part is reflected
Value: 1 (True)

Feature 7: Success of Attack
Value: 0 (Fail)

Design - Reward

Main goal:

Vulnerability detection in shortest time!

1. Positive Reward

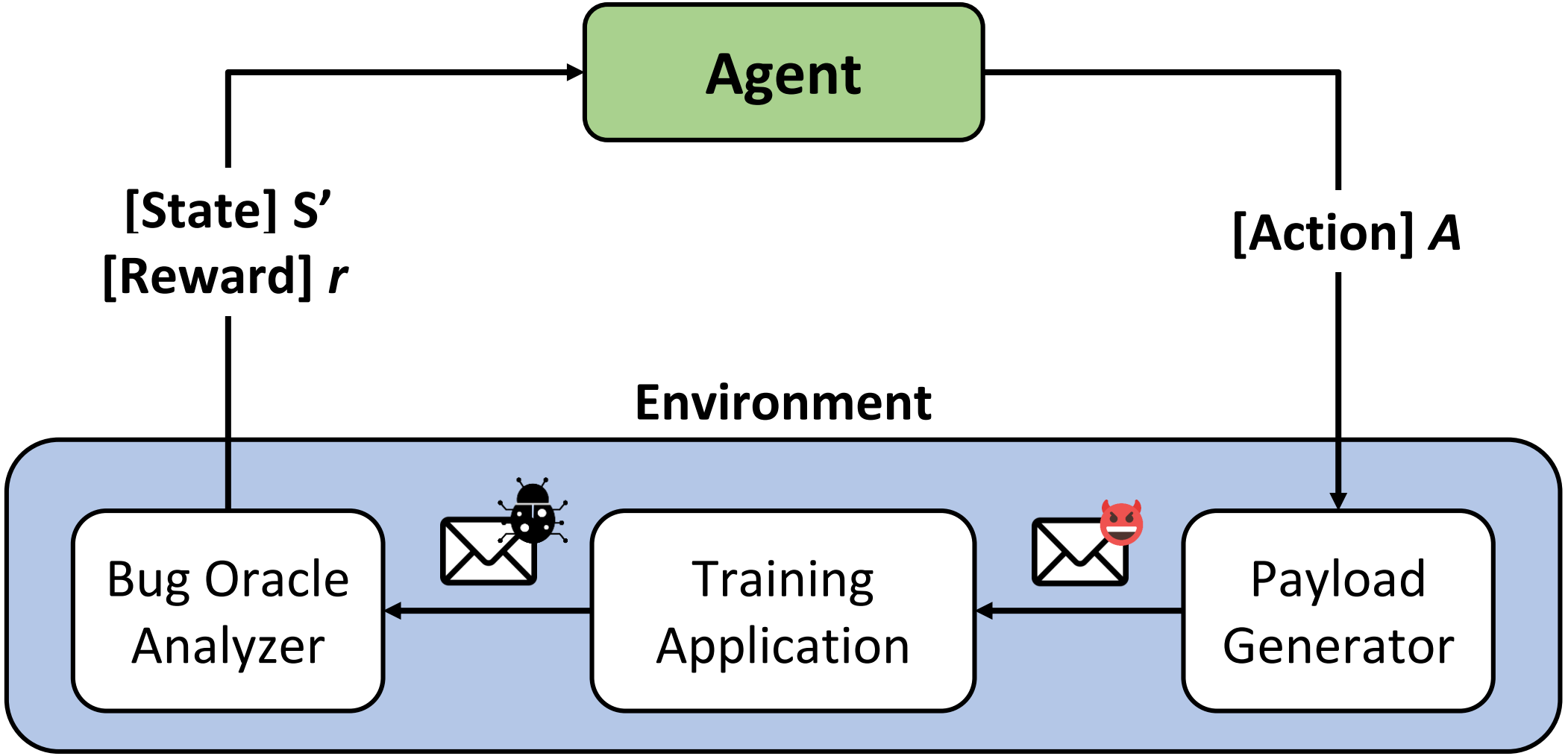
   If attack succeed: *MAX_TRY* - # of attempts

   # of attempts ⬇ Reward ⬆

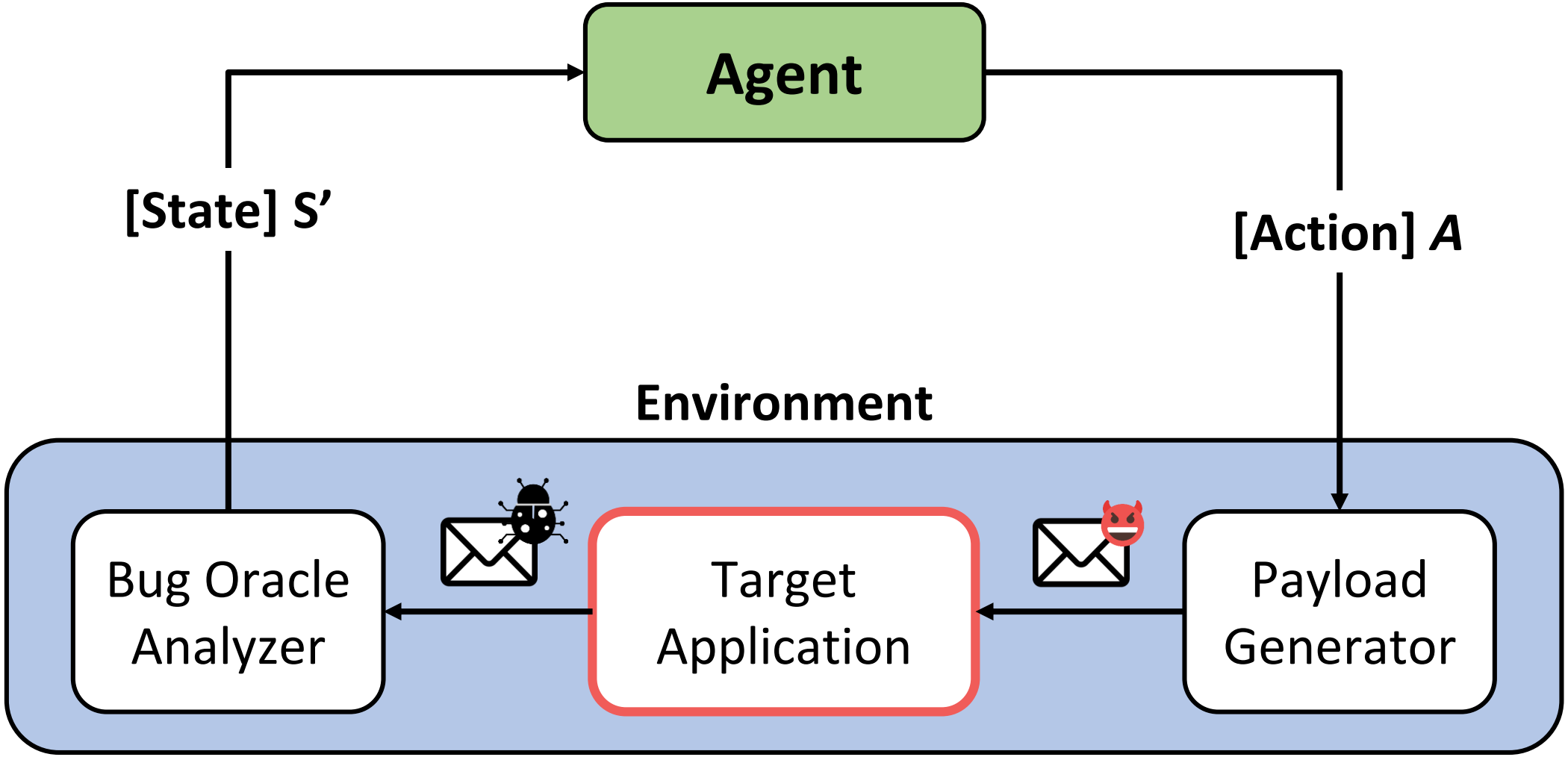
2. Negative Reward

   Attempt #1: <script>alert(1)</script>
   Attempt #2: <script>alert(1)</script>
   } Same payload -> Give -1

Link- Training



Link- Testing (Detection)



Experiment Setup

Training Application

- Firing-Range from Google
- Web Application Vulnerability Scanner Evaluation Project (WAVSEP)
- XSS filter evasion application

Reinforcement Learning Algorithm

- Advanced Actor Critic (A2C)

Comparison Tools

- Open source tools: Wapiti, ZAP, BlackWidow (BW) (*IEEE S&P '21*)
- Commercial Tool: BurpSuite Pro from PortSwigger

Evaluation

Our purpose:

Vulnerability detection in shortest time!

Our evaluation goal:

# of found bugs 

# of attempts 

vs. State-of-the-art Tools

Target Application: **Training Application + OWASP Benchmark**

# of total vulnerabilities: **376**

Tools	TP	FP	FN	# of Requests
Link	**343**	0	**33**	13,423
Burp	300	0	76	141,366
ZAP	287	0	89	36,587
Wapiti	221	0	155	9,185
BW	205	0	171	12,156

vs. State-of-the-art Tools

Target Application: **Training Application + OWASP Benchmark**

# of total vulnerabilities: **376**

Highest # of found bugs! (True positives)

Tools	TP	FP	FN	# of Requests
Link	**343**	0	**33**	13,423
Burp	300	0	76	141,366
ZAP	287	0	89	36,587
Wapiti	221	0	155	9,185
BW	205	0	171	12,156

vs. State-of-the-art Tools

Target Application: **Training Application + OWASP Benchmark**

# of total vulnerabilities: **376**

Lower # of requests

Tools	TP	FP	FN	# of Requests
Link	**343**	0	**33**	13,423
Burp	300	0	76	141,366
ZAP	287	0	89	36,587
Wapiti	221	0	155	9,185
BW	205	0	171	12,156

vs. State-of-the-art Tools

Target Application**: Filter evasion application**
# of total vulnerabilities: **25**

- Requirement of complicated evasion techniques

Tools	TP	FP	FN	# of Requests
Link	**25**	0	**0**	334
Burp	22	0	3	1,852
Wapiti	17	0	8	505
BW	12	0	13	602
ZAP	6	0	19	857

vs. State-of-the-art Tools

Target Application**: Filter evasion application**
# of total vulnerabilities: **25**

- Requirement of complicated evasion techniques

Only tool that find all bugs!

Tools	TP	FP	FN	# of Requests
Link	**25**	0	**0**	334
Burp	22	0	3	1,852
Wapiti	17	0	8	505
BW	12	0	13	602
ZAP	6	0	19	857

vs. State-of-the-art Tools

Target Application: **OWASP benchmark**
# of total vulnerabilities: **246**

- Not included in the training benchmark

Tools	TP	FP	FN	# of Requests
Link	**213**	0	**33**	11,912
Burp	186	0	60	121,311
ZAP	186	0	60	29,483
BW	157	0	89	10,759
Wapiti	137	0	109	6,451

vs. State-of-the-art Tools

Target Application: **OWASP benchmark**
# of total vulnerabilities: **246**

- Not included in the training benchmark

Still best result!

Transferable RL Agent

Tools	TP	FP	FN	# of Requests
Link	**213**	0	**33**	11,912
Burp	186	0	60	121,311
ZAP	186	0	60	29,483
BW	157	0	89	10,759
Wapiti	137	0	109	6,451

Known Real-world Vulnerabilities

Target Application: **12 Real world vulnerable PHP Applications**
# of total vulnerabilities: **49**

Lower than similar result tools

Tools	TP	FP	FN	# of Requests
Link	**43**	0	**6**	4,105
Burp	46	3	3	38,622
ZAP	36	19	13	147,595
Wapiti	33	0	16	7,175
BW	25	0	24	879

Known Real-world Vulnerabilities

Target Application: **12 Real world vulnerable PHP Applications**
# of total vulnerabilities: **49**

Reasonable testing for real-world apps

Tools	TP	FP	FN	# of Requests
Link	**43**	0	**6**	4,105
Burp	46	3	3	38,622
ZAP	36	19	13	147,595
Wapiti	33	0	16	7,175
BW	25	0	24	879

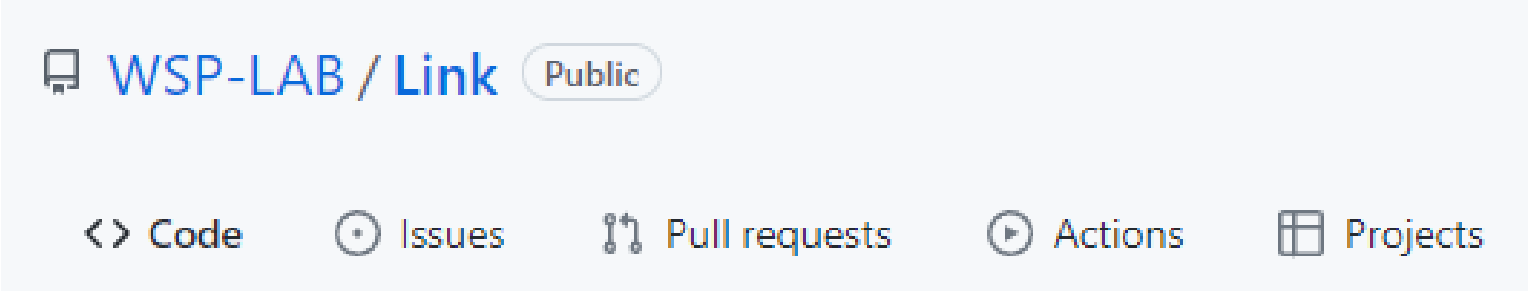
Unknown Real-world Vulnerabilities

- **Target Application**
  - Recent versions of vulnerable 12 applications
  - Recently reported vulnerable apps
- **Zero-day vulnerabilities**
  - Three new vulnerabilities in Geeklog (v2.2.1sr)
  - One in PESCMS (v2.3.3) (CVE-2021-44884)

Conclusion

- We proposed the first **fully-automated** black-box reflected XSS scanner using an **reinforcement learning** agent.
- Link demonstrates its efficiency in **decreasing the number of attack requests**
- Link found **43 vulnerabilities with no false positives** from 12 real world web applications

Open Science



https://github.com/WSP-LAB/Link

Q & A