S&P’2021

Black Widow: Blackbox Data-driven Web Scanning

Benjamin Eriksson*, Giancarlo Pellegrino†, and Andrei Sabelfeld*

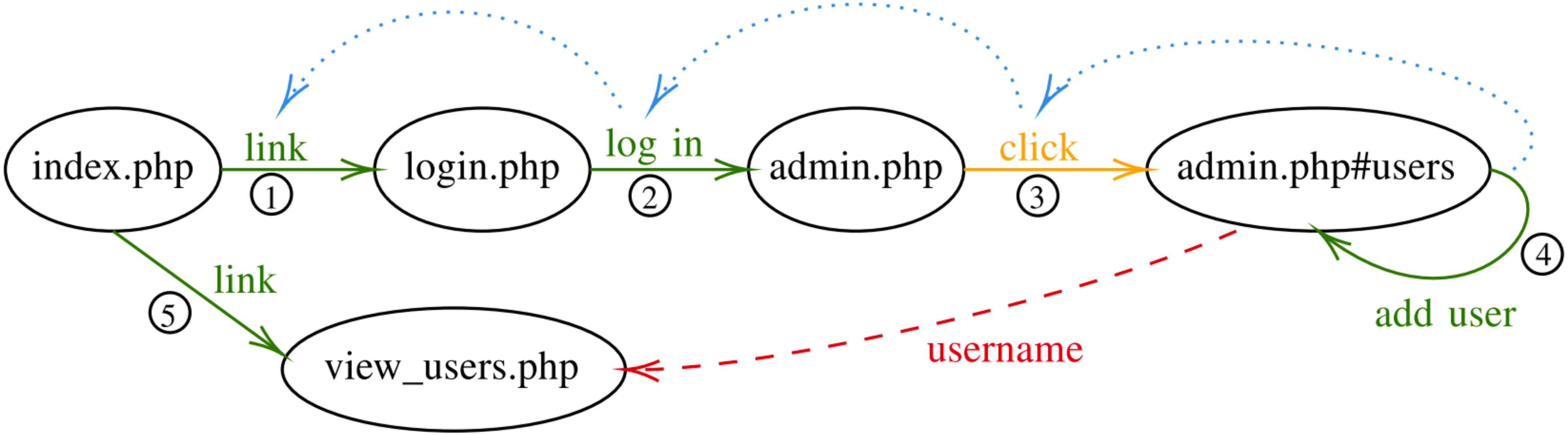
*Chalmers University of Technology
†CISPA Helmholtz Center for Information Security

Blackbox Web Scanning

- No prior knowledge about the applications.
- Using crawlers to explore and identify attack surfaces, e.g., input fields.
- Generating tests to feed the applications and trigger bugs.
- Challenges
    - Coverage of dynamic and complex web applications.
    - Control-flow and data-flow dependencies between states/pages.
        - Index.php ->Viewbooks.php ->Selectbooks.php ->Checkout.php

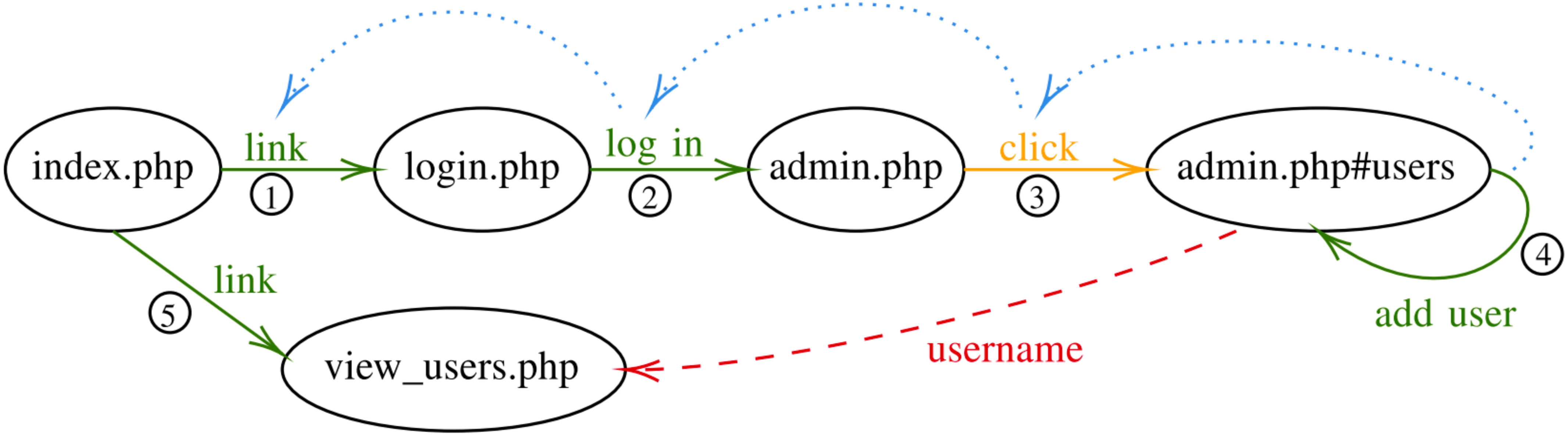
Challenges – Navigation model

- Web applications are stateful.
- The navigation model shall model
  - Links, forms, events, etc. that change states in both server-side and client-side.
  - State conversion and paths.
  - Inter-state dependencies.



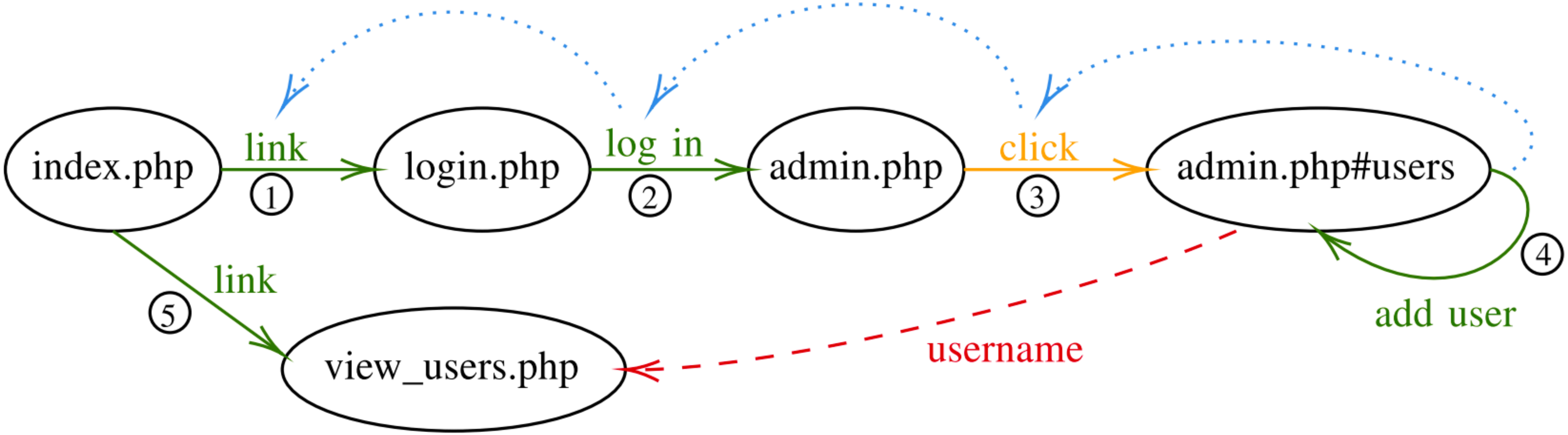
Challenges – Traversing

- Traversing a workflow requires a combination of link navigation, form submission, and event interaction.
- Decide in which order and when to perform (state-changing) actions.
- Workflow reproducing in state changing.



Challenges – Inter-state Dependencies

- Model how user inputs affect web applications.
- Blackbox taint analysis

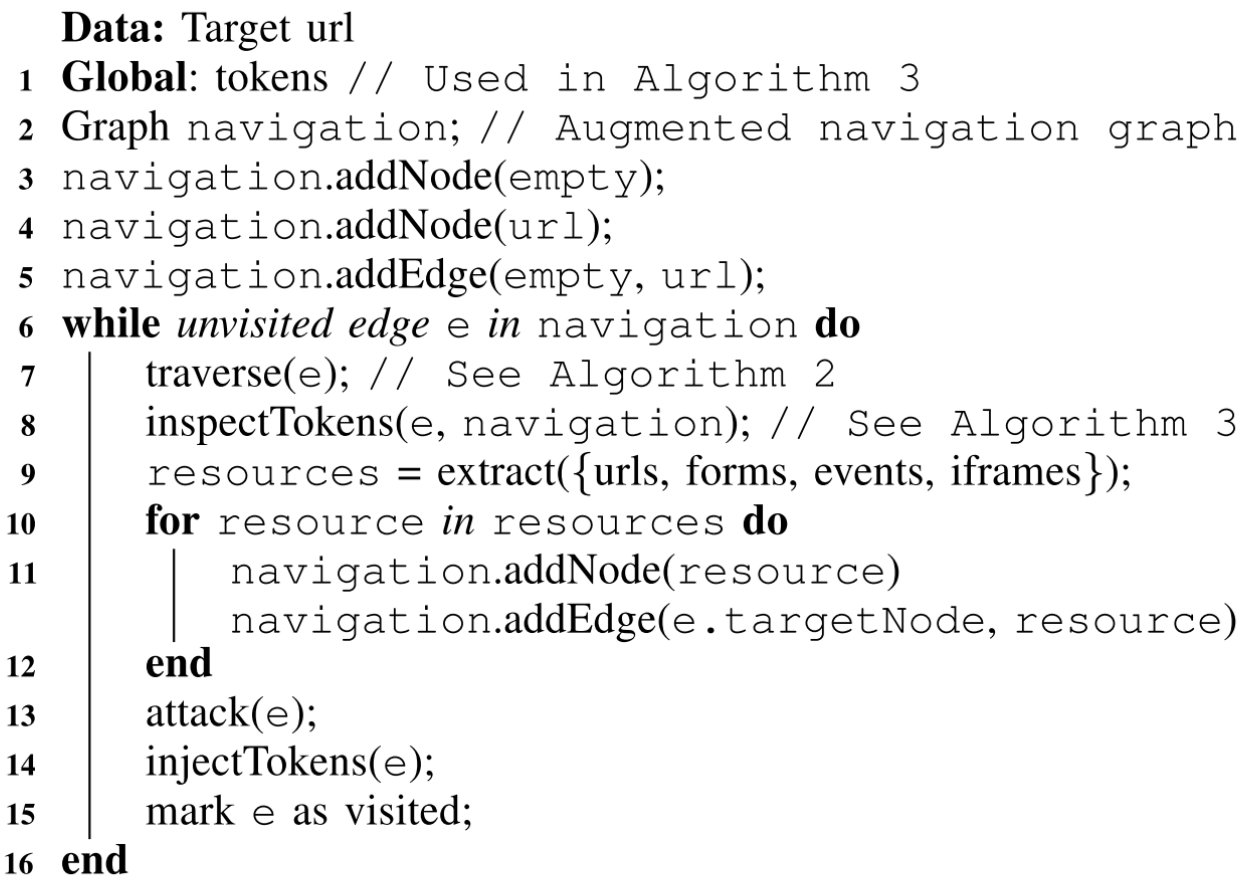


Black Widow – Navigation Modeling

- A labeled directed graph
  - Node is a state of the client-side program, including URL of the page and state of JavaScript program
  - Edge is an action to move from one state to another, considering GET requests, Form submissions, iframes, JavaScript events

Black Widow – Navigation Modeling

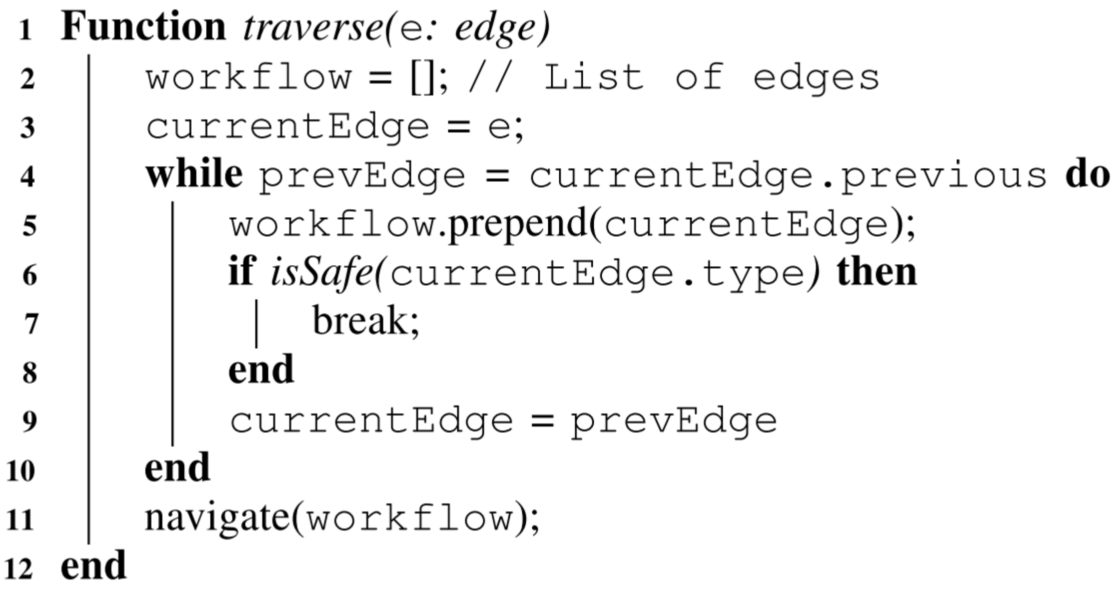
- Gradually extend the navigation graph
- Breadth-first search for unvisited edges in navigation model
- Traverse() finds a sequence of edges (actions) leading to a new page
- Gradually extend the navigation graph



```
Data: Target url
Global: tokens // Used in Algorithm 3
Graph navigation; // Augmented navigation graph
navigation.addNode(empty);
navigation.addNode(url);
navigation.addEdge(empty, url);
while unvisited edge e in navigation do
    traverse(e); // See Algorithm 2
    inspectTokens(e, navigation); // See Algorithm 3
    resources = extract({urls, forms, events, iframes});
    for resource in resources do
        navigation.addNode(resource)
        navigation.addEdge(e.targetNode, resource)
    end
    attack(e);
    injectTokens(e);
    mark e as visited;
end
```

Black Widow – Traversal

- Recursively inspect the previous edge till a *safe* edge.
    - Safe actions do not expect to change server state.
    - The purpose is to allow automated retrieval processes (spiders) and cache performance optimization (pre-fetching)
    - Idempotent actions: multiple identical actions has the same effect of a single such request.
    - *Traverse from beginning.*

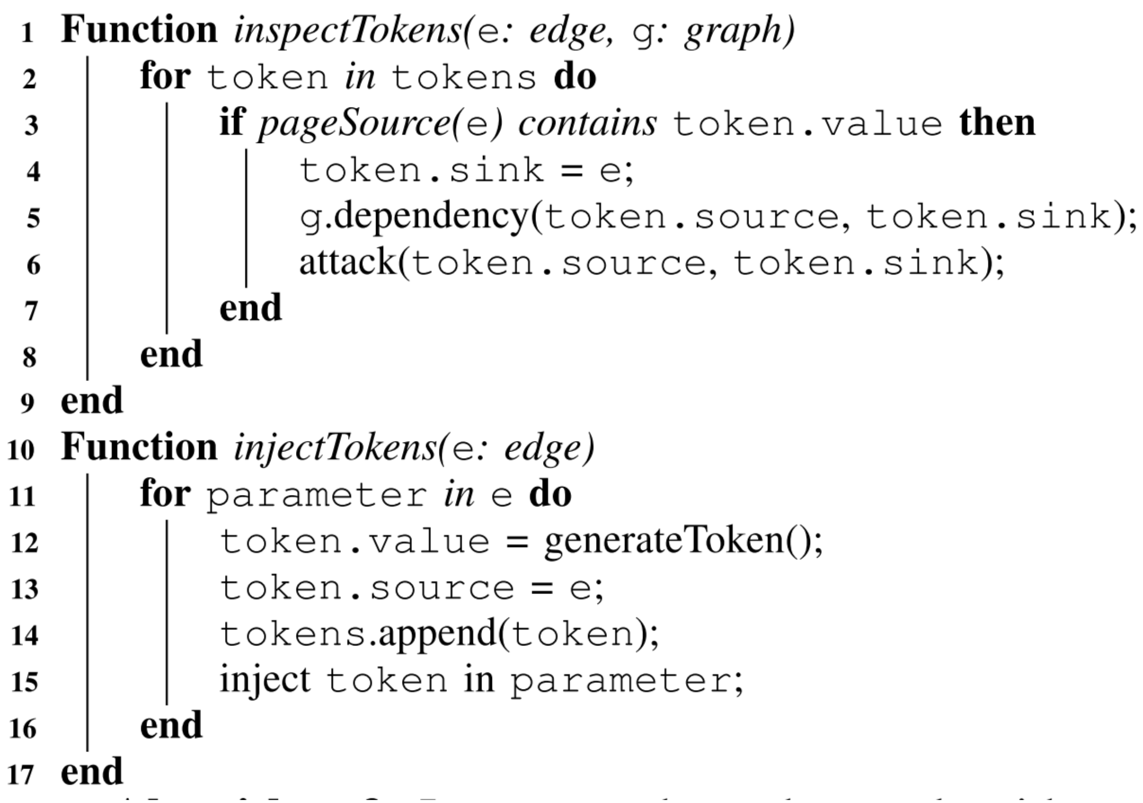


```
Function traverse(e: edge)
    workflow = []; // List of edges
    currentEdge = e;
    while prevEdge = currentEdge.previous do
        workflow.prepend(currentEdge);
        if isSafe(currentEdge.type) then
            break;
        end
        currentEdge = prevEdge
    end
    navigate(workflow);
end
```

Black Widow – Inter-state Dependencies

- Identify input fields as sources
- Inject unique tokens as taint values
- Reappear of the tokens as sinks

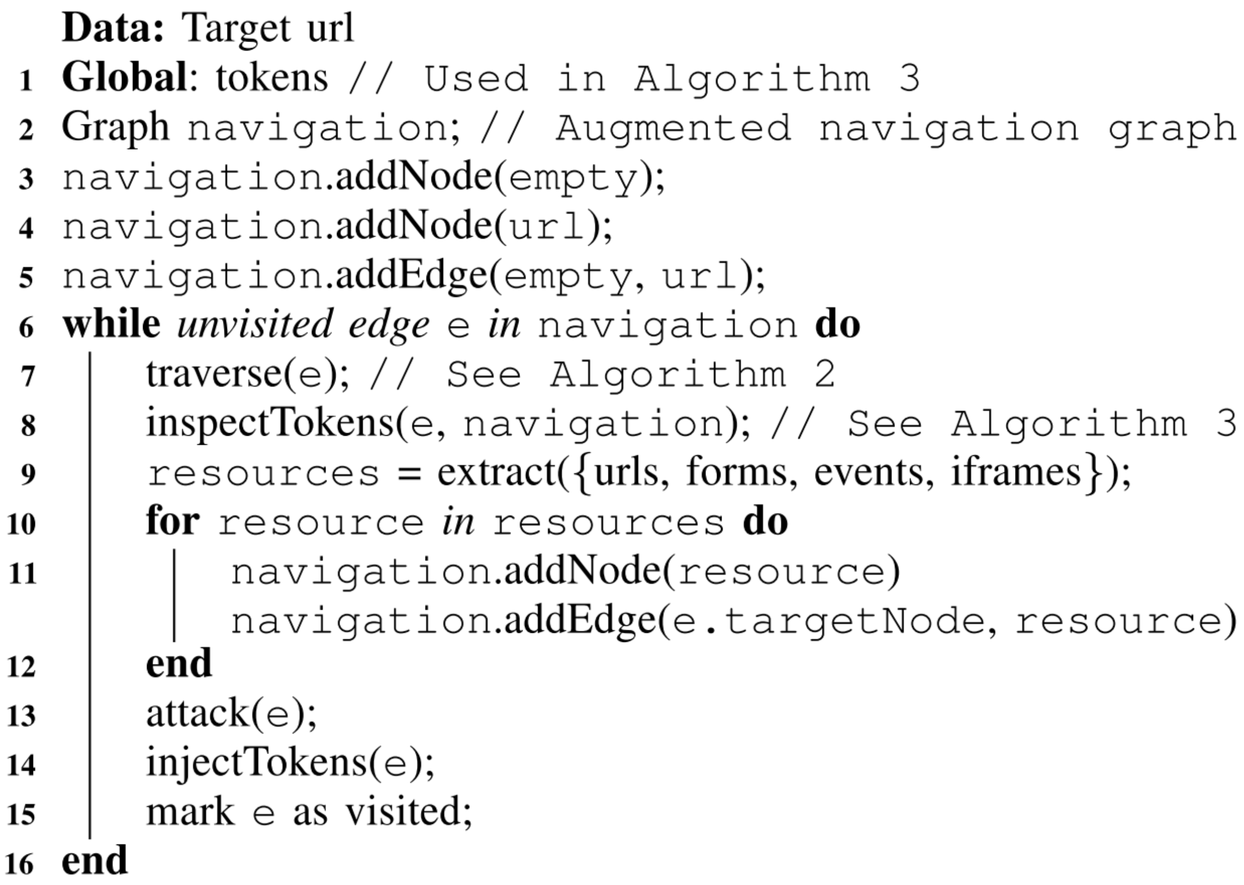
- Attack(): fuzz the source to check the sink



```
Function inspectTokens(e: edge, g: graph)
    for token in tokens do
        if pageSource(e) contains token.value then
            token.sink = e;
            g.dependency(token.source, token.sink);
            attack(token.source, token.sink);
        end
    end
end
Function injectTokens(e: edge)
    for parameter in e do
        token.value = generateToken();
        token.source = e;
        tokens.append(token);
        inject token in parameter;
    end
end
```

Black Widow – Overall

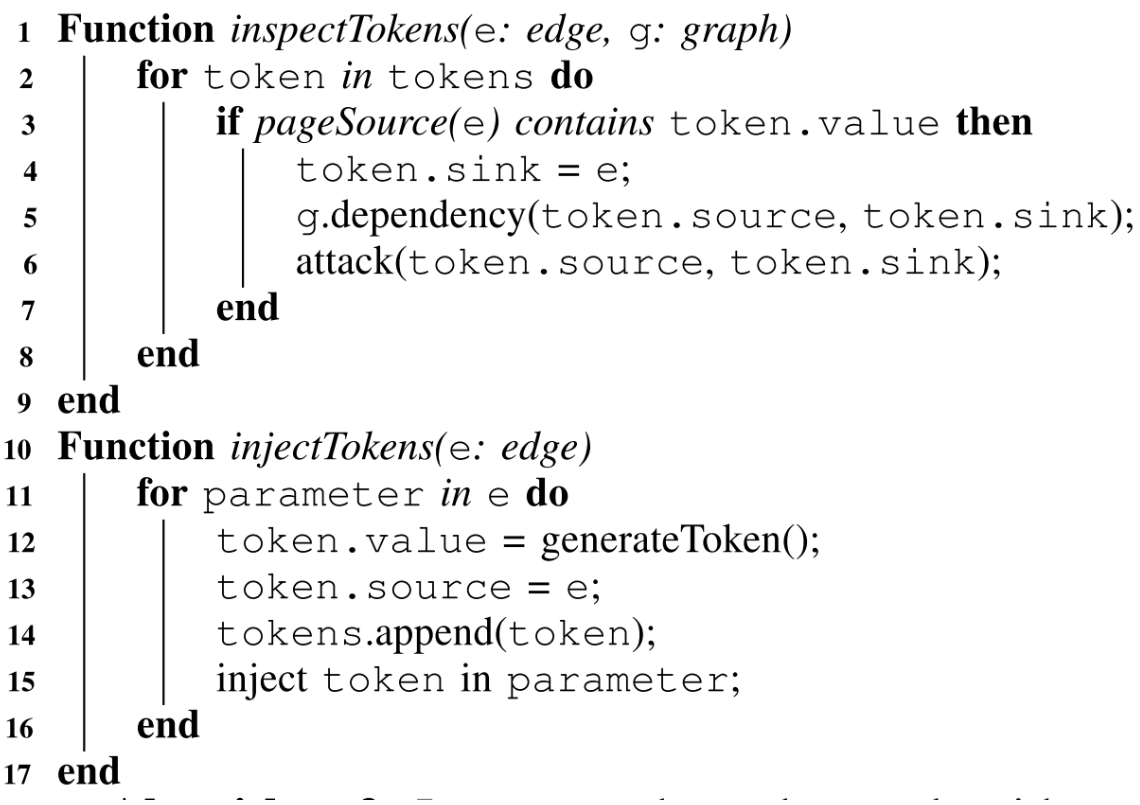
- InspectTokens()
- Attack(): fuzz parameters for vulnerability detection. Parameters might include URL parameters, form values, etc.



```
Data: Target url
Global: tokens // Used in Algorithm 3
Graph navigation; // Augmented navigation graph
navigation.addNode(empty);
navigation.addNode(url);
navigation.addEdge(empty, url);
while unvisited edge e in navigation do
    traverse(e); // See Algorithm 2
    inspectTokens(e, navigation); // See Algorithm 3
    resources = extract({urls, forms, events, iframes});
    for resource in resources do
        navigation.addNode(resource)
        navigation.addEdge(e.targetNode, resource)
    end
    attack(e);
    injectTokens(e);
    mark e as visited;
end
```

Black Widow – Dynamic XSS Detection

- Inject JavaScript code xss(ID) on every page that insert ID to a result array
- Generate JavaScript payload that tries to call xss(ID)
- Monitor and inspect the result array for XSS vulnerabilities.



```
Function inspectTokens(e: edge, g: graph)
    for token in tokens do
        if pageSource(e) contains token.value then
            token.sink = e;
            g.dependency(token.source, token.sink);
            attack(token.source, token.sink);
        end
    end
end
Function injectTokens(e: edge)
    for parameter in e do
        token.value = generateToken();
        token.source = e;
        tokens.append(token);
        inject token in parameter;
    end
end
```

Implementation

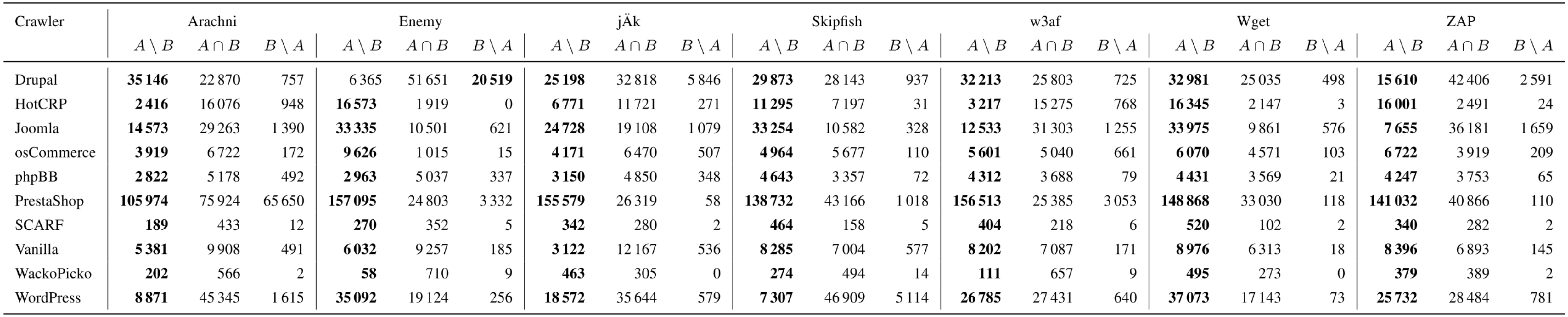
- Use Python and Selenium to control a browser
- A custom JavaScript library to extract actions

Evaluation

- Code coverage
- Vulnerability detection
- Comparison:
  - State-of-the-art academic blackbox scanners: Enemy of the State, jAk
  - Scanner used in related works: Skipfish, Wget, w3af, Arachni and ZAP
  - Commercial scanners are NOT included.
- Application dataset:
  - Applications with known vulnerabilities
  - Modern production-grade applications.

Evaluation – Coverage

- Has highest coverage on 9/10 applications
- Finds many unique code that no others find (Table II)
- Finds more unique code than other tools (Table III)



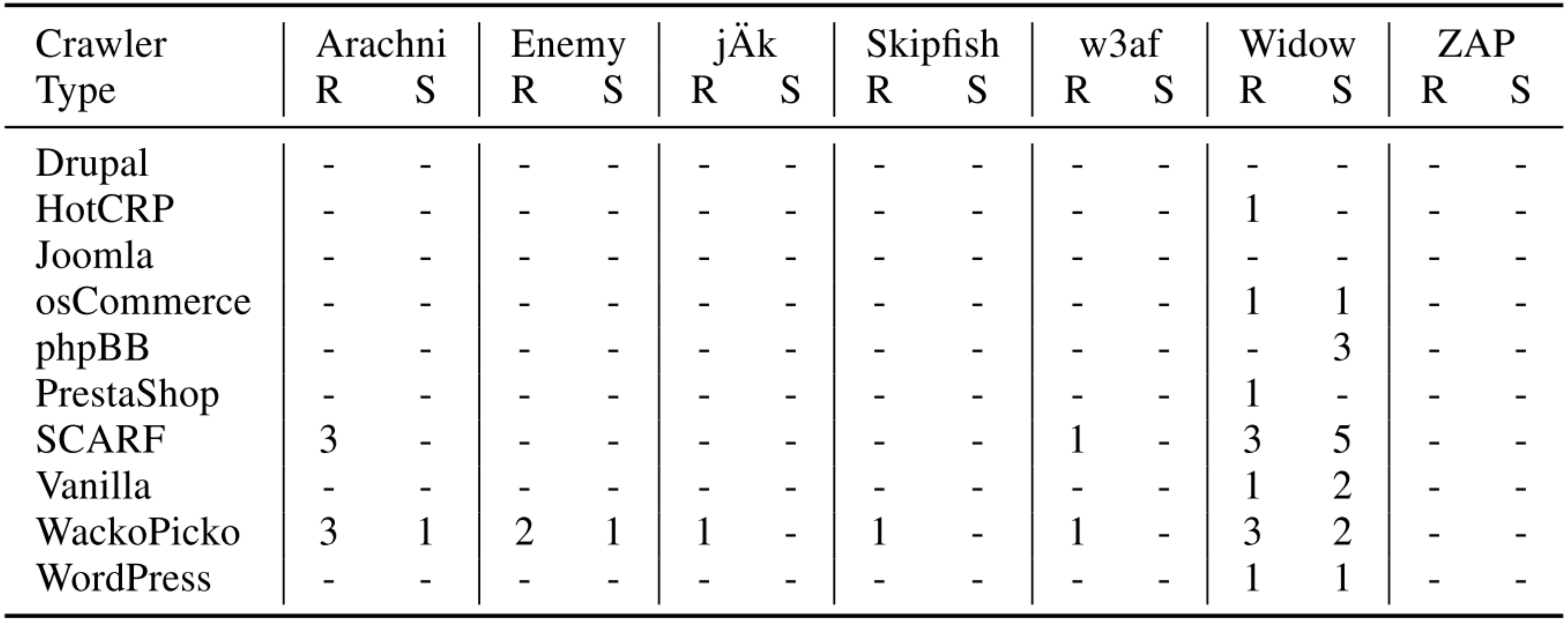
Crawler	Arachni			Enemy			jÄk			Skipfish			w3af			Wget			ZAP		
	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$
Drupal	**35 146**	22 870	757	6 365	51 651	**20 519**	**25 198**	32 818	5 846	**29 873**	28 143	937	**32 213**	25 803	725	**32 981**	25 035	498	**15 610**	42 406	2 591
HotCRP	**2 416**	16 076	948	**16 573**	1 919	0	**6 771**	11 721	271	**11 295**	7 197	31	**3 217**	15 275	768	**16 345**	2 147	3	**16 001**	2 491	24
Joomla	**14 573**	29 263	1 390	**33 335**	10 501	621	**24 728**	19 108	1 079	**33 254**	10 582	328	**12 533**	31 303	1 255	**33 975**	9 861	576	**7 655**	36 181	1 659
osCommerce	**3 919**	6 722	172	**9 626**	1 015	15	**4 171**	6 470	507	**4 964**	5 677	110	**5 601**	5 040	661	**6 070**	4 571	103	**6 722**	3 919	209
phpBB	**2 822**	5 178	492	**2 963**	5 037	337	**3 150**	4 850	348	**4 643**	3 357	72	**4 312**	3 688	79	**4 431**	3 569	21	**4 247**	3 753	65
PrestaShop	**105 974**	75 924	65 650	**157 095**	24 803	3 332	**155 579**	26 319	58	**138 732**	43 166	1 018	**156 513**	25 385	3 053	**148 868**	33 030	118	**141 032**	40 866	110
SCARF	**189**	433	12	**270**	352	5	**342**	280	2	**464**	158	5	**404**	218	6	**520**	102	2	**340**	282	2
Vanilla	**5 381**	9 908	491	**6 032**	9 257	185	**3 122**	12 167	536	**8 285**	7 004	577	**8 202**	7 087	171	**8 976**	6 313	18	**8 396**	6 893	145
WackoPicko	**202**	566	2	**58**	710	9	**463**	305	0	**274**	494	14	**111**	657	9	**495**	273	0	**379**	389	2
WordPress	**8 871**	45 345	1 615	**35 092**	19 124	256	**18 572**	35 644	579	**7 307**	46 909	5 114	**26 785**	27 431	640	**37 073**	17 143	73	**25 732**	28 484	781

Evaluation – Coverage

- Enemy of the State outperforms Black Widow on Drupal
  - Enemy keeps authenticated state while Black Widow loses the state too early.
  - Logout action is chosen early in Black Widow.
  - Drupal does not present a login form when trying to perform an unauthorized operation.
- Skipfish performs well on WordPress because some pages do not include JavaScript.

Evaluation – XSS Vulnerability Detection

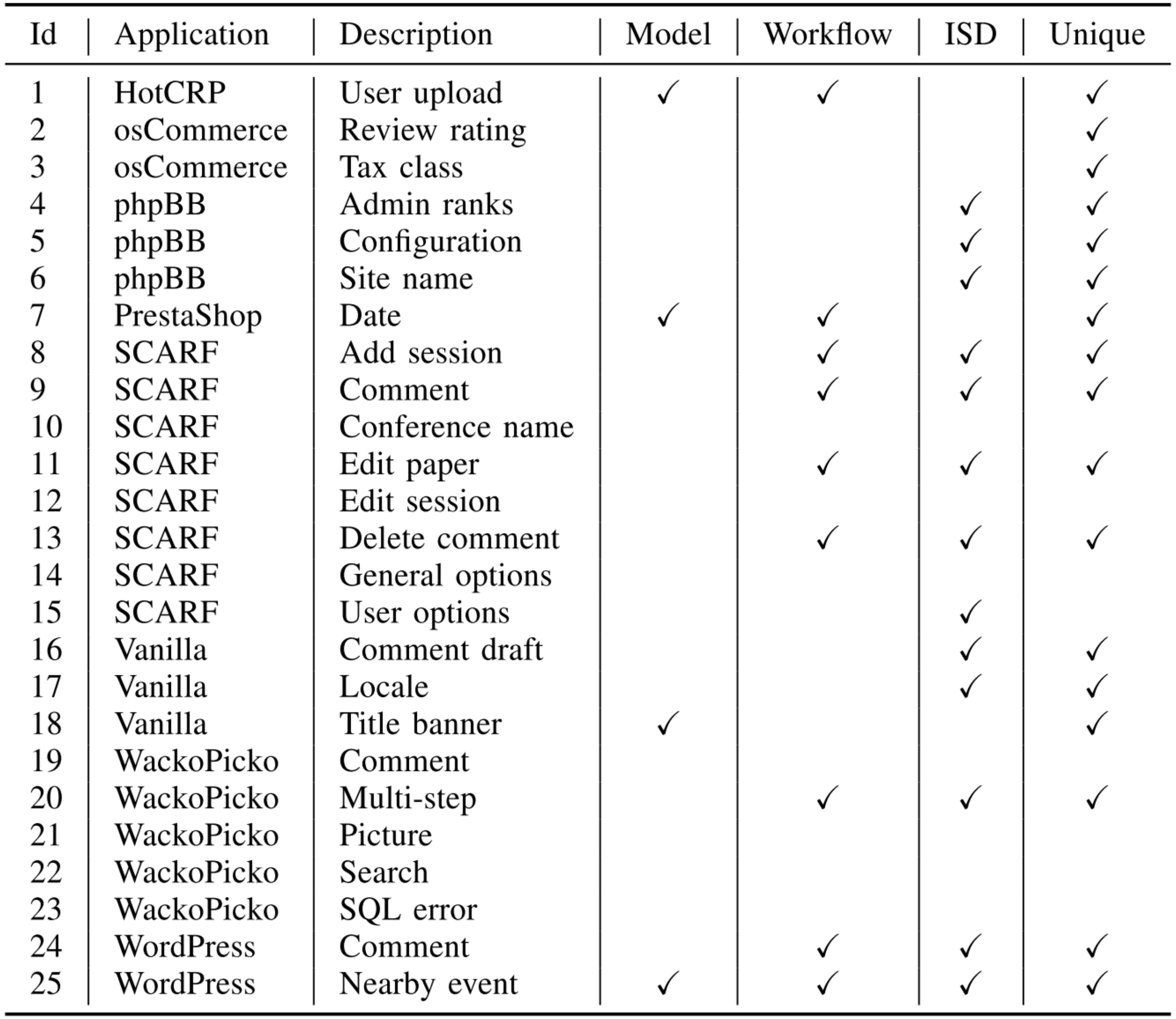
- Black Widow finds 25 unique vulnerabilities, six of which are previous unknown, including ones in modern complicated applications.
- Black Widow detects all vulnerabilities found by other tools.
- No false positives



Crawler	Arachni		Enemy		jÄk		Skipfish		w3af		Widow		ZAP	
Type	R	S	R	S	R	S	R	S	R	S	R	S	R	S
Drupal	-	-	-	-	-	-	-	-	-	-	-	-	-	-
HotCRP	-	-	-	-	-	-	-	-	-	-	1	-	-	-
Joomla	-	-	-	-	-	-	-	-	-	-	-	-	-	-
osCommerce	-	-	-	-	-	-	-	-	-	-	1	1	-	-
phpBB	-	-	-	-	-	-	-	-	-	-	-	3	-	-
PrestaShop	-	-	-	-	-	-	-	-	-	-	1	-	-	-
SCARF	3	-	-	-	-	-	-	-	1	-	3	5	-	-
Vanilla	-	-	-	-	-	-	-	-	-	-	1	2	-	-
WackoPicko	3	1	2	1	1	-	1	-	1	-	3	2	-	-
WordPress	-	-	-	-	-	-	-	-	-	-	1	1	-	-

Evaluation – Feature Attribution

- Impact of individual techniques
  - Navigation modeling: combination of actions to find the XSS
  - Traversing: Injection point depends on previous states
  - Inter-state dependencies: Different reflection and injection points.



Id	Application	Description	Model	Workflow	ISD	Unique
1	HotCRP	User upload	✓	✓		✓
2	osCommerce	Review rating				✓
3	osCommerce	Tax class				✓
4	phpBB	Admin ranks			✓	✓
5	phpBB	Configuration			✓	✓
6	phpBB	Site name			✓	✓
7	PrestaShop	Date	✓	✓		✓
8	SCARF	Add session		✓	✓	✓
9	SCARF	Comment		✓	✓	✓
10	SCARF	Conference name				
11	SCARF	Edit paper		✓	✓	✓
12	SCARF	Edit session				
13	SCARF	Delete comment		✓	✓	✓
14	SCARF	General options				
15	SCARF	User options			✓	
16	Vanilla	Comment draft			✓	✓
17	Vanilla	Locale			✓	✓
18	Vanilla	Title banner	✓			✓
19	WackoPicko	Comment				
20	WackoPicko	Multi-step		✓	✓	✓
21	WackoPicko	Picture				
22	WackoPicko	Search				
23	WackoPicko	SQL error				
24	WordPress	Comment		✓	✓	✓
25	WordPress	Nearby event	✓	✓	✓	✓

Evaluation –Not Covered

- False positive analysis
- Back-to-back comparison

Conclusion

- Techniques to identify inter-state dependencies with support of multiple user actions.
- High code coverage and more vulnerabilities.
- New XSS vulnerabilities in modern applications.

Reasons to present this paper

- Learn how do crawlers/scanners work internally.
- Learn how the “state” and “chain” problems are solved in such a blackbox work.

Comments on this work

- Technically more like a combination of existing works.
- Reasonably good results because of the techniques, e.g., high coverage.
- Interestingly, new XSS bugs can be found in modern web applications.
- Writing can be improved (maybe)?
    - Repeated and redundant texts/tables in code coverage. Some text description is not consistent with the table. Unclear decriptions. (It is just a preprint)
- Can an edge have multiple previous edges?
- Taint token can be aware of constraints, e.g., client-side constraints for input validation.

Potential future work

- Modeling states is always required in fuzzing complex systems, e.g., kernel (components).
    - NDSS'20: HFL: Hybrid Fuzzing on the Linux Kernel
        - Obtain potential dependency pairs (read/write on the same memory), write operation has to be invoked before read.
    - S&P'20: IJON: Exploring Deep State Spaces via Fuzzing
        - Add human annotation to guide fuzzer to particularly study certain location or data structure. It can play and solve *Super Mario Bros* game!
    - S&P'19: Fuzzing File Systems via Two-Dimensional Input Space Exploration
        - Context-aware workloads (FS system calls).
- Complex system fuzzing tries to use clean/fresh targeted program, e.g., OS, for reproducing problem.
- Balance of reproduction and state exploration.

Furthermore

- Hybrid Fuzzing
  - S&P’20: SAVIOR: Towards Bug-Driven Hybrid Testing
    - Converting coverage-oriented to bug-driven.
  - S&P’20: PANGOLIN: Incremental Hybrid Fuzzing with Polyhedral Path Abstraction
    - Preserve the explored states for more effective mutation and constraint solving.