ReScan: A Middleware Framework for Realistic and Robust Black-box Web Application Scanning

Kostas Drakonakis, Sotiris Ioannidis, Jason Polakis
kostasdrk@ics.forth.gr

Web Application Scanners

- ❖ Plethora of existing black-box scanners
  - ➢ App agnostic
  - ➢ Variety of testing techniques & approaches
  - ➢ Cover different flaws
- ❖ Extremely valuable for uncovering vulnerabilities

Web Application Scanners

- Plethora of existing black-box scanners
  - App agnostic
  - Variety of testing techniques & approaches
  - Cover different flaws
- Extremely valuable for uncovering vulnerabilities

- The Web keeps evolving
  - New features, APIs, client-side code
  - Scanners need to keep up

However…

- Scanners suffer from core limitations
  - Lack of full-fledged browser
  - Ignore client-side events/state
  - “Stateless” navigation
  - Naive authentication methods
  - Prone to false positives/negatives
  - Inefficient due to testing similar pages

However…

- Scanners suffer from core limitations
  - Lack of full-fledged browser
  - Ignore client-side events/state
  - “Stateless” navigation
  - Naive authentication methods
  - Prone to false positives/negatives
  - Inefficient due to testing similar pages
- Miss on coverage & vulnerability detection

However…

- Scanners suffer from core limitations
  - Lack of full-fledged browser
  - Ignore client-side events/state
  - “Stateless” navigation
  - Naive authentication methods
  - Prone to false positives/negatives
  - Inefficient due to testing similar pages
- Miss on coverage & vulnerability detection
  - Still useful with a plethora of valuable testing techniques

However…

- Scanners suffer from core limitations
  - Lack of full-fledged browser
  - Ignore client-side events/state
  - “Stateless” navigation
  - Naive authentication methods
  - Prone to false positives/negatives
  - Inefficient due to testing similar pages
- Miss on coverage & vulnerability detection
  - Still useful with a plethora of valuable testing techniques
- Implementing a single new tool
  - Prohibitive engineering effort
  - Inherently can’t incorporate all past and future techniques

However…

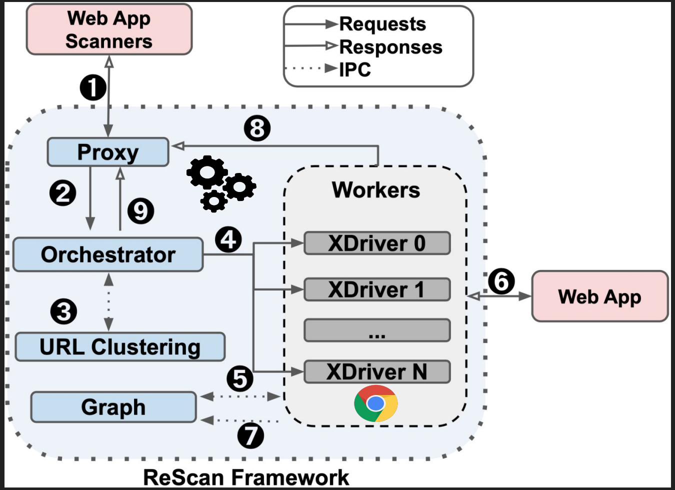
- Scanners suffer from core limitations

How can we address these core limitations,
without having to redesign everything from scratch?

- Implementing a single new tool
  - Prohibitive engineering effort
  - Inherently can’t incorporate all past and future techniques

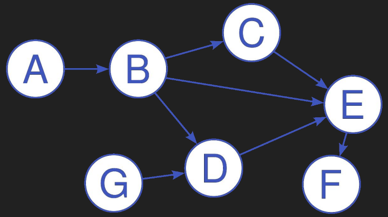
Enter **ReScan**

- ❖ Scanner-agnostic middleware framework
  - ➢ Intercepts all scanner requests
  - ➢ Executes them through a SotA browser
- ❖ Transparently addresses limitations
  - ➢ Multiple enhancement modules
  - ➢ Employed on every scanner request
- ❖ Several technical challenges to overcome
  - ➢ Careful design choices
  - ➢ Ensure robustness



Enhancement techniques

- Build navigation model
  - Links, forms, events
  - Correctly transition through app states


$$\begin{tikzcd} & & C \arrow[dr] & \\ A \arrow[r] & B \arrow[ur] \arrow[rr] \arrow[d] & & E \arrow[d] \\ G \arrow[r] & D \arrow[urr] & & F \end{tikzcd}$$

- Event discovery
  - Cover multiple JS events
  - Find dynamic DOM content & requests

- Detect *inter-state dependencies* (ISD)
  - Payloads affecting other parts of the app
  - Useful for certain vulnerabilities, e.g., stored XSS

https://www.pngegg.com/

Enhancement techniques

- Authentication helper
  - Detect credentials
  - Dynamically infer auth oracle
  - Re-establish sessions when needed
- XSS false positive/negative reduction
  - Detect payload bearing requests
  - Map page alerts/popups to injections
- API for *future* scanners
  - Access to ReScan's internal knowledge
  - Enable/disable modules at runtime

{ api }

Enhancement techniques

❖ Authentication helper

How can we transparently communicate our findings back to the scanner?

❖

❖ API for *future* scanners

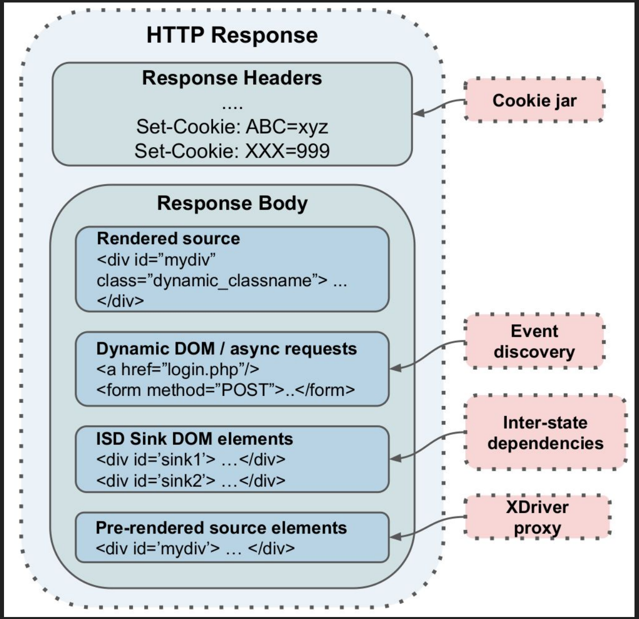
➢ Access to ReScan's internal knowledge

➢ Enable/disable modules at runtime

{ api }

Middleware enhancement

- Utilize the existing communication channel
  - HTTP response
- Discovered endpoints
  - Transcribed as links/forms in final HTTP response
- Detected ISD sinks
  - Append sink's element including payload
  - Pre-fill form inputs with unique tokens
- Authentication & app state
  - Set-Cookie headers back to scanner
- Browsers may alter payload structure
  - Append elements' pre-rendered source

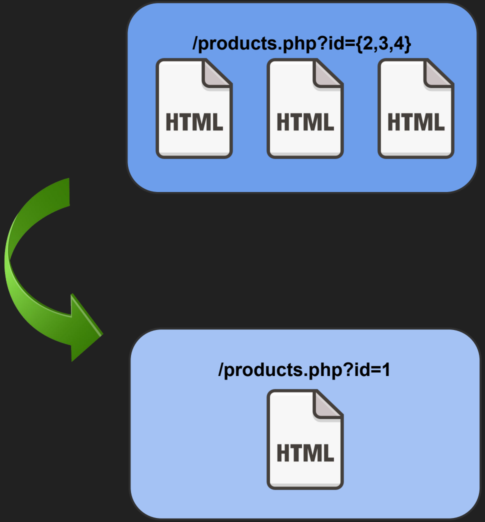


URL Clustering

- Identify functionality-similar pages
  - Common URL path, different parameters
  - Compute DOM similarity
  - Prevent scanner from learning redundant pages

URL Clustering

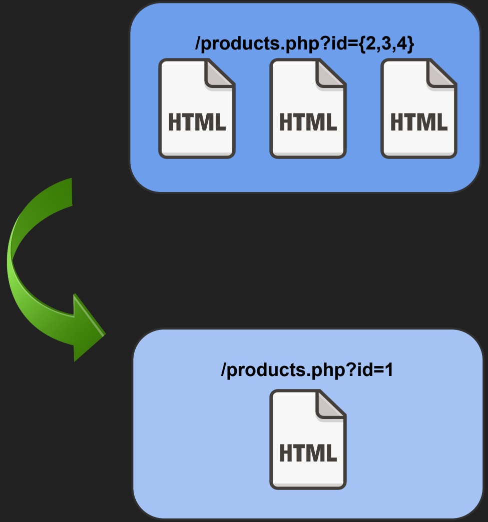
- Identify functionality-similar pages
  - Common URL path, different parameters
  - Compute DOM similarity
  - Prevent scanner from learning redundant pages
- On a new request
  - Keep track of already seen parameters
  - Iteratively swap new values with known ones
  - Compare swapped page with original request
  - Generate clustering rules
  - If rule applies, always redirect to same page



https://www.pngegg.com/

URL Clustering

- Identify functionality-similar pages
  - Common URL path, different parameters
  - Compute DOM similarity
  - Prevent scanner from learning redundant pages
- On a new request
  - Keep track of already seen parameters
  - Iteratively swap new values with known ones
  - Compare swapped page with original request
  - Generate clustering rules
  - If rule applies, always redirect to same page
- Ensure consistency across clusters
- Account for arbitrary URL ordering

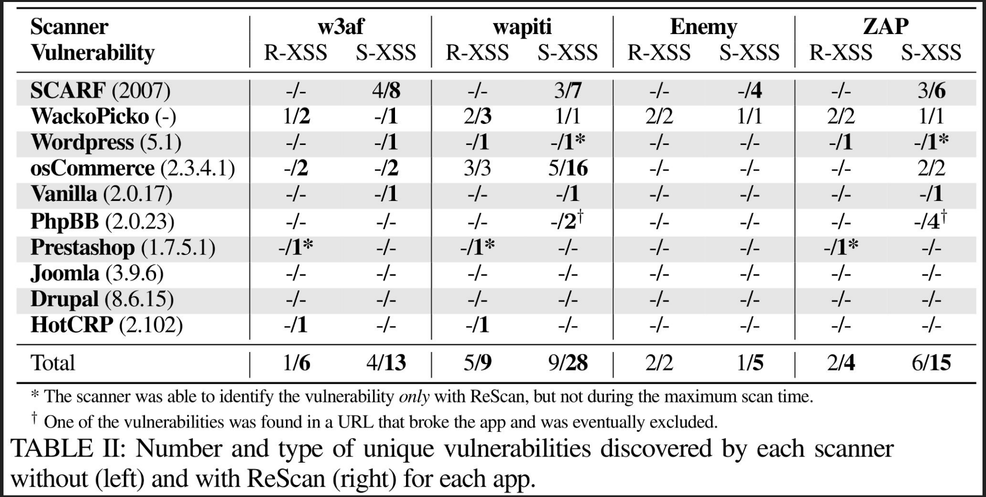


https://www.pngegg.com/

Evaluation

- ❖ Popular black-box scanners
  - ➢ w3af, wapiti, ZAP, Enemy of the State [USENIX Sec ‘12]
  - ➢ Configured to scan for XSS
  - ➢ Authenticated scans
  - ➢ Max scan time set to 24h
- ❖ Diverse application set with 10 apps
  - ➢ Wordpress, osCommerce, PhpBB, HotCRP…
  - ➢ Modern & older ones
- ❖ Ran all scanners on all apps with and without ReScan

Detection & Coverage



Scanner	w3af		wapiti		Enemy		ZAP	
Vulnerability	R-XSS	S-XSS	R-XSS	S-XSS	R-XSS	S-XSS	R-XSS	S-XSS
SCARF (2007)	-/-	4/**8**	-/-	3/**7**	-/-	-/**4**	-/-	3/**6**
WackoPicko (-)	1/**2**	-/**1**	2/**3**	1/1	2/2	1/1	2/2	1/1
Wordpress (5.1)	-/-	-/**1**	-/**1**	-/**1***	-/-	-/-	-/**1**	-/**1***
osCommerce (2.3.4.1)	-/**2**	-/**2**	3/3	5/**16**	-/-	-/-	-/-	2/2
Vanilla (2.0.17)	-/-	-/**1**	-/-	-/**1**	-/-	-/-	-/-	-/**1**
PhpBB (2.0.23)	-/-	-/-	-/-	-/**2**†	-/-	-/-	-/-	-/**4**†
Prestashop (1.7.5.1)	-/**1***	-/-	-/**1***	-/-	-/-	-/-	-/**1***	-/-
Joomla (3.9.6)	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
Drupal (8.6.15)	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
HotCRP (2.102)	-/**1**	-/-	-/**1**	-/-	-/-	-/-	-/-	-/-
Total	1/**6**	4/**13**	5/**9**	9/**28**	2/2	1/**5**	2/**4**	6/**15**

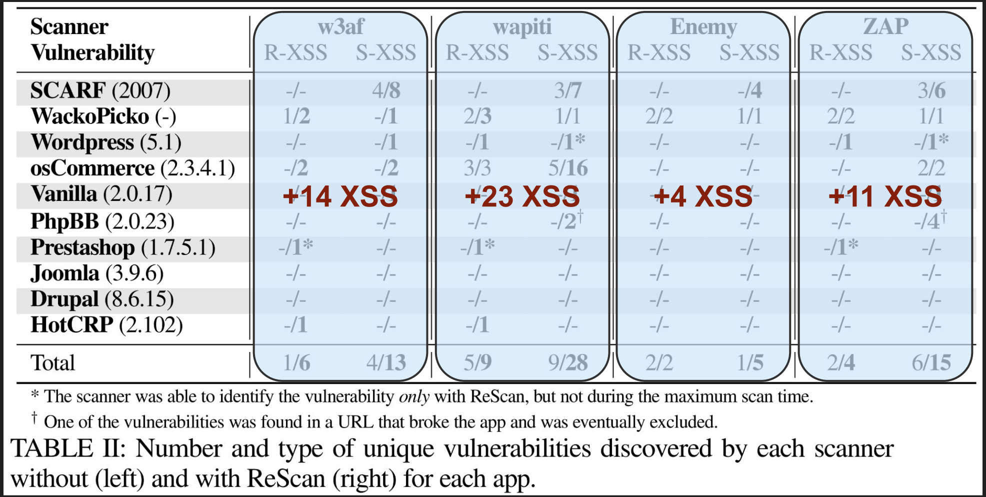
* The scanner was able to identify the vulnerability *only* with ReScan, but not during the maximum scan time.

† One of the vulnerabilities was found in a URL that broke the app and was eventually excluded.

TABLE II: Number and type of unique vulnerabilities discovered by each scanner without (left) and with ReScan (right) for each app.

- Detection
  - ReScan improves all scanners for most apps
  - Eliminated wapiti's and ZAP's FPs

Detection & Coverage



Scanner / Vulnerability	w3af R-XSS	w3af S-XSS	wapiti R-XSS	wapiti S-XSS	Enemy R-XSS	Enemy S-XSS	ZAP R-XSS	ZAP S-XSS
SCARF (2007)	-/-	4/**8**	-/-	3/**7**	-/-	-/**4**	-/-	3/**6**
WackoPicko (-)	1/**2**	-/**1**	2/**3**	1/1	2/2	1/1	2/2	1/1
Wordpress (5.1)	-/-	-/**1**	-/**1**	-/**1***	-/-	-/-	-/**1**	-/**1***
osCommerce (2.3.4.1)	-/**2**	-/**2**	3/3	5/**16**	-/-	-/-	-/-	2/2
Vanilla (2.0.17)	+14 XSS		+23 XSS		+4 XSS		+11 XSS	
PhpBB (2.0.23)	-/-	-/-	-/-	-/**2**†	-/-	-/-	-/-	-/**4**†
Prestashop (1.7.5.1)	-/**1***	-/-	-/**1***	-/-	-/-	-/-	-/**1***	-/-
Joomla (3.9.6)	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
Drupal (8.6.15)	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
HotCRP (2.102)	-/**1**	-/-	-/**1**	-/-	-/-	-/-	-/-	-/-
Total	1/**6**	4/**13**	5/**9**	9/**28**	2/2	1/**5**	2/**4**	6/**15**

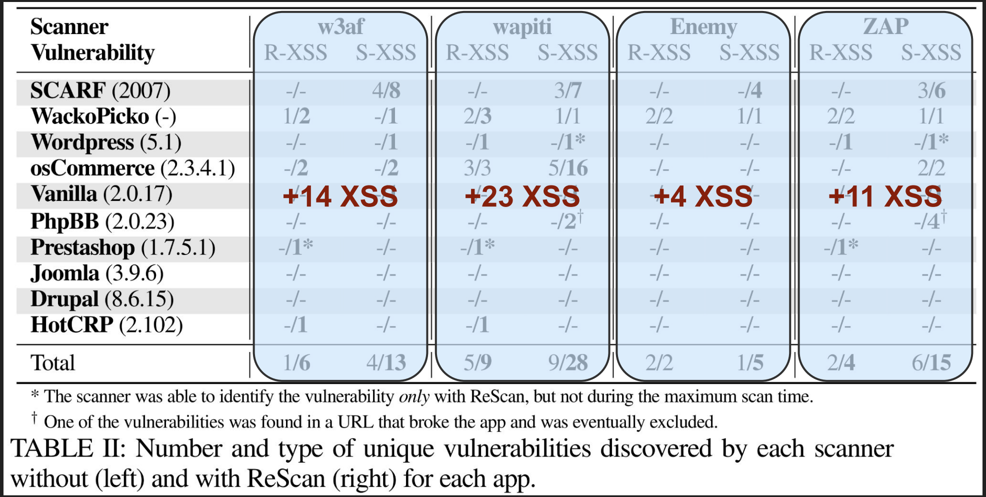
* The scanner was able to identify the vulnerability *only* with ReScan, but not during the maximum scan time.
† One of the vulnerabilities was found in a URL that broke the app and was eventually excluded.

TABLE II: Number and type of unique vulnerabilities discovered by each scanner without (left) and with ReScan (right) for each app.

- ❖ Detection
  - ➢ ReScan improves all scanners for most apps
  - ➢ Eliminated wapiti's and ZAP's FPs
  - ➢ Overall, +4 reflected, +21 stored XSS

kostasdrk@ics.forth.gr

Detection & Coverage



Scanner / Vulnerability	w3af R-XSS	w3af S-XSS	wapiti R-XSS	wapiti S-XSS	Enemy R-XSS	Enemy S-XSS	ZAP R-XSS	ZAP S-XSS
SCARF (2007)	-/-	4/8	-/-	3/7	-/-	-/4	-/-	3/6
WackoPicko (-)	1/2	-/1	2/3	1/1	2/2	1/1	2/2	1/1
Wordpress (5.1)	-/-	-/1	-/1	-/1*	-/-	-/-	-/1	-/1*
osCommerce (2.3.4.1)	-/2	-/2	3/3	5/16	-/-	-/-	-/-	2/2
Vanilla (2.0.17)	+14 XSS		+23 XSS		+4 XSS		+11 XSS	
PhpBB (2.0.23)	-/-	-/-	-/-	-/2†	-/-	-/-	-/-	-/4†
Prestashop (1.7.5.1)	-/1*	-/-	-/1*	-/-	-/-	-/-	-/1*	-/-
Joomla (3.9.6)	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
Drupal (8.6.15)	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
HotCRP (2.102)	-/1	-/-	-/1	-/-	-/-	-/-	-/-	-/-
Total	1/6	4/13	5/9	9/28	2/2	1/5	2/4	6/15

* The scanner was able to identify the vulnerability *only* with ReScan, but not during the maximum scan time.
† One of the vulnerabilities was found in a URL that broke the app and was eventually excluded.

TABLE II: Number and type of unique vulnerabilities discovered by each scanner without (left) and with ReScan (right) for each app.

- Detection
  - ReScan improves all scanners for most apps
  - Eliminated wapiti's and ZAP's FPs
  - Overall, +4 reflected, +21 stored XSS
- Coverage
  - Improved in all cases between 3% - 935%
  - On average 168% improvement

Further evaluation

- Successfully detects other types of vulnerabilities
  - Unrestricted file upload
  - Login brute-forcing
  - Blind SQL injection

- Outperforms current SotA [Black Widow - S&P '21]
  - Partially addresses some of the limitations
  - +8 reflected, +15 stored XSS
  - +46% code coverage on average

Performance

- ❖ Non-negligible overhead
  - ➢ When compared to standalone scanners
  - ➢ Numerous techniques, full-fledged browser
  - ➢ Each request completed in < 5 seconds on average
  - ➢ Max scan time reached for 15 / 40 scans
- ❖ URL clustering improves performance
  - ➢ ~6.7x speedup
- ❖ Outperforms current SotA in most cases
  - ➢ BW reached time limit in 8/10 apps

Performance

- Non-negligible overhead
  - When compared to standalone scanners
  - Numerous techniques, full-fledged browser
  - Each request completed in < 5 seconds on average
  - Max scan time reached for 15 / 40 scans
- URL clustering improves performance
  - ~6.7x speedup
- Outperforms current SotA in most cases
  - BW reached time limit in 8/10 apps
- **Acceptable trade-off, given the significant improvements**

Conclusion

- Designed scanner-agnostic middleware framework
  - Transparently addresses scanners' limitations
  - Numerous enhancement techniques
  - Can aid existing *and* future scanners

- Comprehensive evaluation on diverse scanners and apps
  - Facilitates vulnerability detection (XSS + more)
  - Significantly increases code coverage
  - Outperforms current state-of-the-art

- Code & apps' docker images publicly available
  - https://gitlab.com/kostasdrk/rescan/

kostasdrk@ics.forth.gr

Scanners' limitations

- Only ZAP uses a real browser
- Only Enemy
  - Creates a navigation model
  - Clusters pages (based on link structure)
- No ISD detection + FP/FN elimination
- w3af + wapiti use naive authentication
- **At least 4 aspects neglected by each scanner**

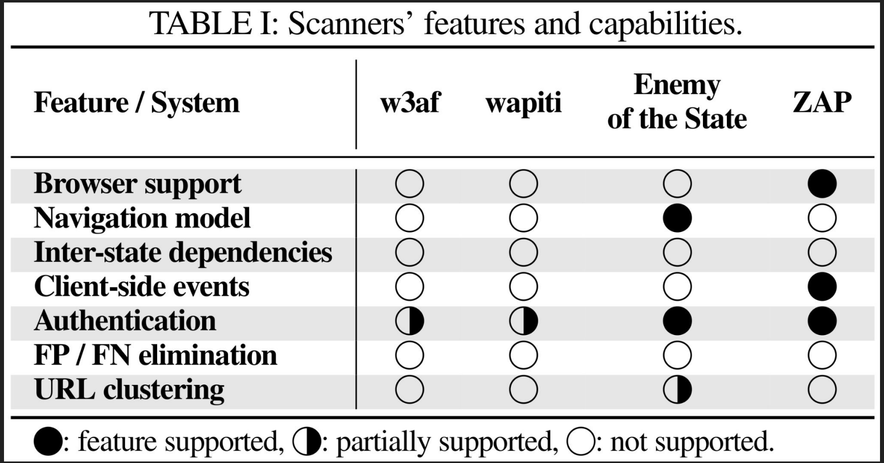


TABLE I: Scanners' features and capabilities.

Feature / System	w3af	wapiti	Enemy of the State	ZAP
Browser support	○	○	○	●
Navigation model	○	○	●	○
Inter-state dependencies	○	○	○	○
Client-side events	○	○	○	●
Authentication	◑	◑	●	●
FP / FN elimination	○	○	○	○
URL clustering	○	○	◑	○

●: feature supported, ◑: partially supported, ○: not supported.

Navigation model

- Directed graph
  - Nodes: Unique URLs
  - Edges: GET, FORM, EVENT, IFRAME, REDIRECT
- Collect all such edges from each URL
- Subsequent requests are mapped to their edge
- Recursively construct their workflow
  - Follow parent edges until first GET and execute from there

Event discovery

- Used jAk's lib to capture elements with events
- Trigger each event
  - MutationObserver to capture new links/forms/iframes
  - Capture requests & block them to avoid state changes
- BFS approach to capture nested events
  - Event dependency chains
- All events and dependency chains are included in the navigation model

Inter-state dependencies

- Background worker
  - Keep track of submitted values (ISD sources)
  - Detect if they appear in other pages (ISD sinks)
  - Notify browser workers of detected ISD links

- Browser workers on POST requests
  - Detect parameters that may include scanner payload
  - Fetch candidate ISD sinks for each parameter
  - If payload appears in sink, embed it in final HTTP response

Authentication helper

- Capture first auth request and detect credentials
  - All scanners initially submit the valid username/password
- Infer authentication oracle
  - New request without cookies (unauthenticated)
  - Check if username/email/logout/login form only appears on one of the pages
- Run oracle after every request
  - In new tab to maintain initial request's state
- Re-login if logged out and retry request

XSS FP/FN elimination

- Identify scanner payloads
  - Keyword-based (*alert, prompt, javascript:)*
  - Most scanners try to trigger an alert popup
- For any alert that occurs
  - Map its text to detected injections
  - Verified via code execution
- Effectiveness depends on underlying scanner
  - Does not reuse payloads -> Alerts are mapped to exactly one injection; FP/FN elimination
  - Reuses payloads -> Alerts are mapped to all injections; reduced confidence

Coverage

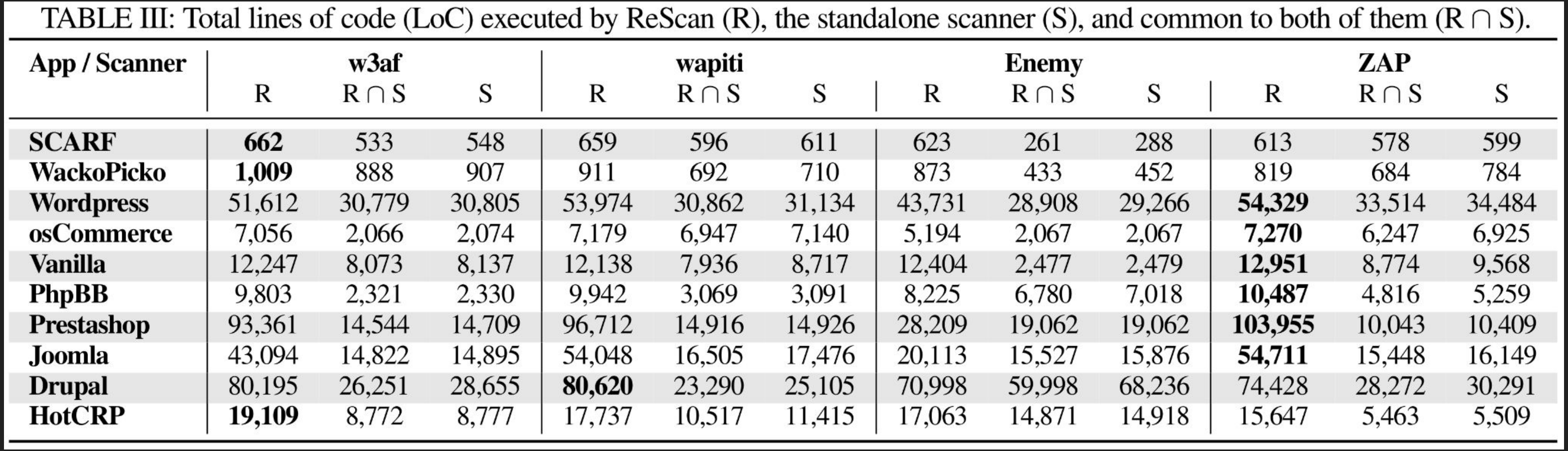


TABLE III: Total lines of code (LoC) executed by ReScan (R), the standalone scanner (S), and common to both of them (R ∩ S).

App / Scanner	w3af			wapiti			Enemy			ZAP		
	R	R ∩ S	S	R	R ∩ S	S	R	R ∩ S	S	R	R ∩ S	S
SCARF	**662**	533	548	659	596	611	623	261	288	613	578	599
WackoPicko	**1,009**	888	907	911	692	710	873	433	452	819	684	784
Wordpress	51,612	30,779	30,805	53,974	30,862	31,134	43,731	28,908	29,266	**54,329**	33,514	34,484
osCommerce	7,056	2,066	2,074	7,179	6,947	7,140	5,194	2,067	2,067	**7,270**	6,247	6,925
Vanilla	12,247	8,073	8,137	12,138	7,936	8,717	12,404	2,477	2,479	**12,951**	8,774	9,568
PhpBB	9,803	2,321	2,330	9,942	3,069	3,091	8,225	6,780	7,018	**10,487**	4,816	5,259
Prestashop	93,361	14,544	14,709	96,712	14,916	14,926	28,209	19,062	19,062	**103,955**	10,043	10,409
Joomla	43,094	14,822	14,895	54,048	16,505	17,476	20,113	15,527	15,876	**54,711**	15,448	16,149
Drupal	80,195	26,251	28,655	**80,620**	23,290	25,105	70,998	59,998	68,236	74,428	28,272	30,291
HotCRP	**19,109**	8,772	8,777	17,737	10,517	11,415	17,063	14,871	14,918	15,647	5,463	5,509

- ❖ Unique LoC during each scan
  - ➢ Improved in all cases
- ❖ Sampled & inspected
  - ➢ Several cases which directly led to missed vulnerabilities

Total scanning times

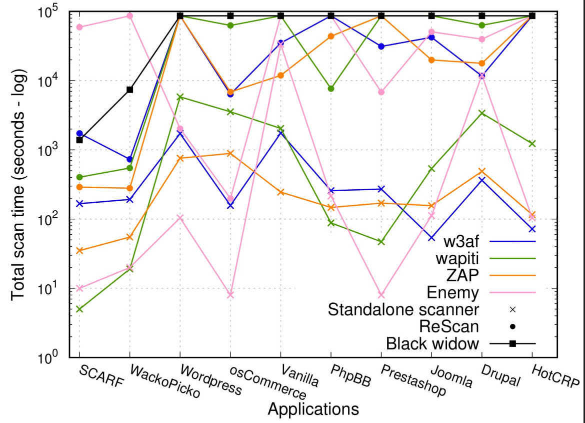


Fig. 3: Total scan time in seconds for each app/ scanner pair with and without ReScan.

- ❖ Overhead can be between minutes or even several hours
  - ➢ Depends on underlying scanner and target app
- ❖ In most cases, total scan time < 24 hours

Request processing performance

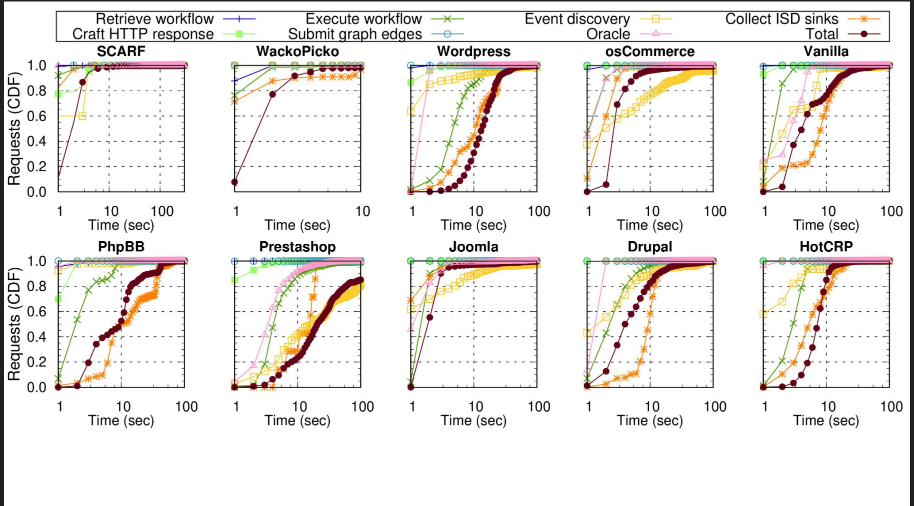


Fig. 4: Requests' CDF per application, in terms of total as well as individual components' processing time.

- Workflow and event discovery < 3 sec for most apps
- Fetching ISD sinks < 2 sec for 4 apps 6 - 16 sec for the rest
- Oracle takes < 2 sec for 99% of requests

DOM similarity threshold

- Compiled 3 sets of pages for each app
  - 1st: different URLs & functionalities
  - 2nd: similar URLs & functionalities (should be clustered)
  - 3rd: similar URLs & different functionalities

- For each pair within each set
  - Calculated modified normalized DOM-edit distance (*mNDD*)
- Different pages (1st, 3rd): min mNDD = 0.014
- Similar pages (2nd): max mNDD = 0.009

- Threshold = 0.009 to avoid possible FPs

State-of-the-art comparison

- ❖ Cannot handle asynchronous requests' payloads
- ❖ Authentication
  - ➢ No oracle
  - ➢ Only re-logins when presented with a login form
  - ➢ Does not retry failed edges
- ❖ Clustering
  - ➢ Hard limit on number of similar pages
  - ➢ Does not consider parameters' values when clustering similar pages (FPs)
- ❖ Sequential execution

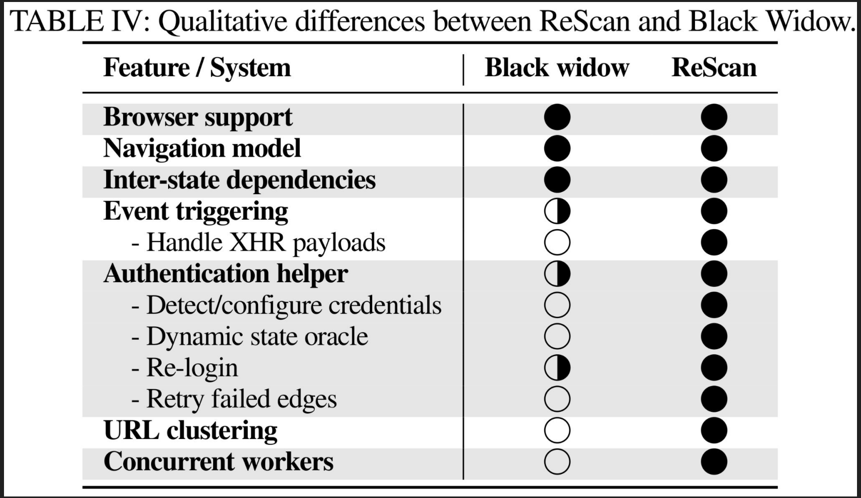


TABLE IV: Qualitative differences between ReScan and Black Widow.

Feature / System	**Black widow**	**ReScan**
Browser support	●	●
Navigation model	●	●
Inter-state dependencies	●	●
Event triggering	◑	●
- Handle XHR payloads	○	●
Authentication helper	◑	●
- Detect/configure credentials	○	●
- Dynamic state oracle	○	●
- Re-login	◑	●
- Retry failed edges	○	●
URL clustering	○	●
Concurrent workers	○	●