Minimalist: Semi-automated Debloating of PHP Web Applications through Static Analysis

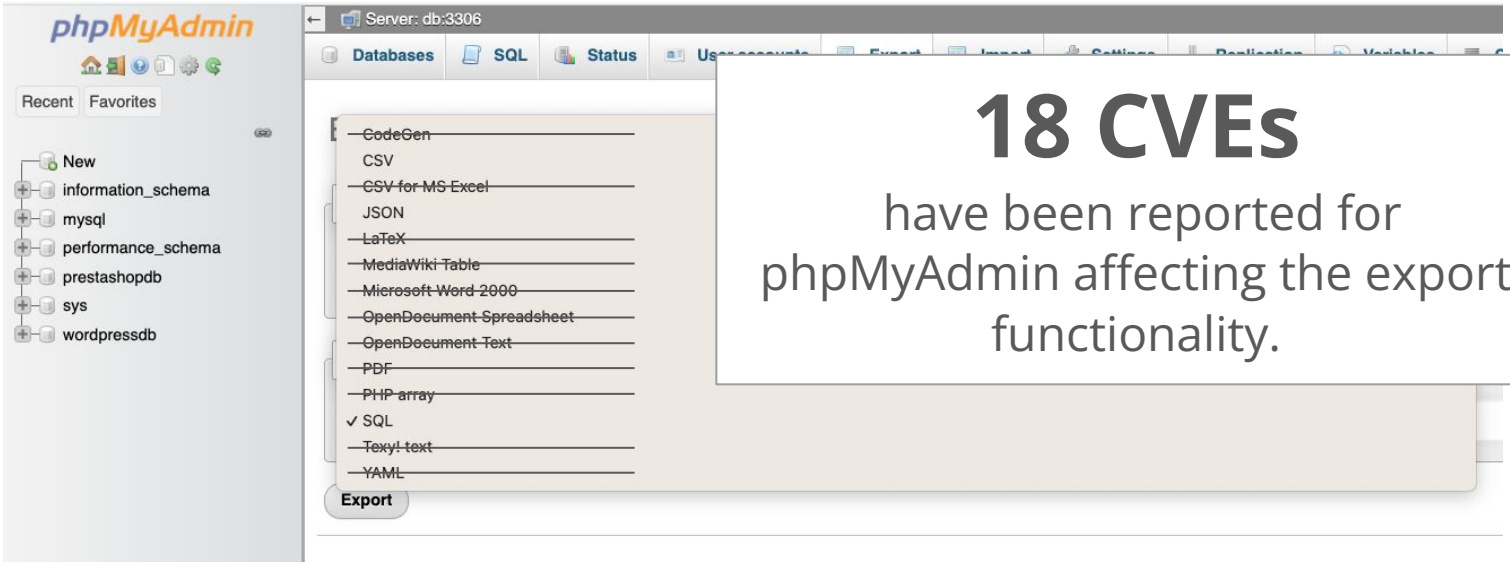
Rasoul Jahanshahi **Babak AminAzad**

Nick Nikiforakis Manuel Egele

Code bloat

- What is code bloat?
  - *It is the sum of all unused pieces of code in an application*
- Why is it bad?
- What can we do about it?

Unused code contains vulnerabilities



18 CVEs

have been reported for phpMyAdmin affecting the export functionality.

Debloating: Identifying and removing unused code

- Less is More (LIM) - Usenix Security 2019 [3]
- Simulate user behavior
- Use dynamic traces to determine file and function usage
- Debloat the unused portion of code

Results

47%	60%
Smaller Apps	Less CVEs

Sad reality: Dynamic instrumentation does not scale

- Can be miserably slow
  - 2x to 17x increase page load time
- Strictly tied to an instance of an application
  - A change in user input or state of the database can trigger an error due to removed code

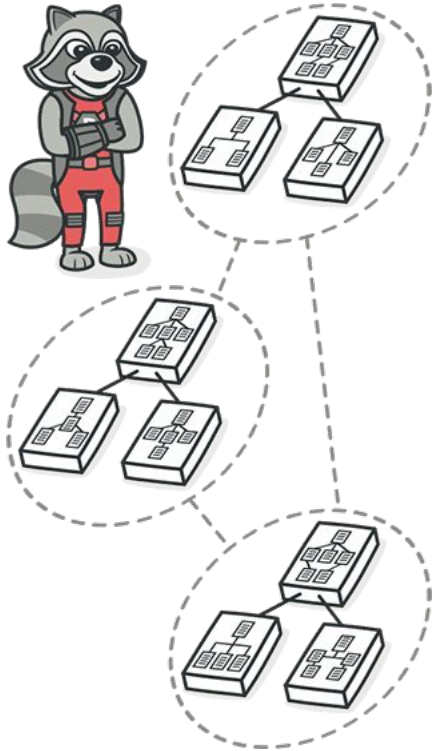
Let’s fix it!

Requirements

- No instrumentation overhead
- Reusable analysis

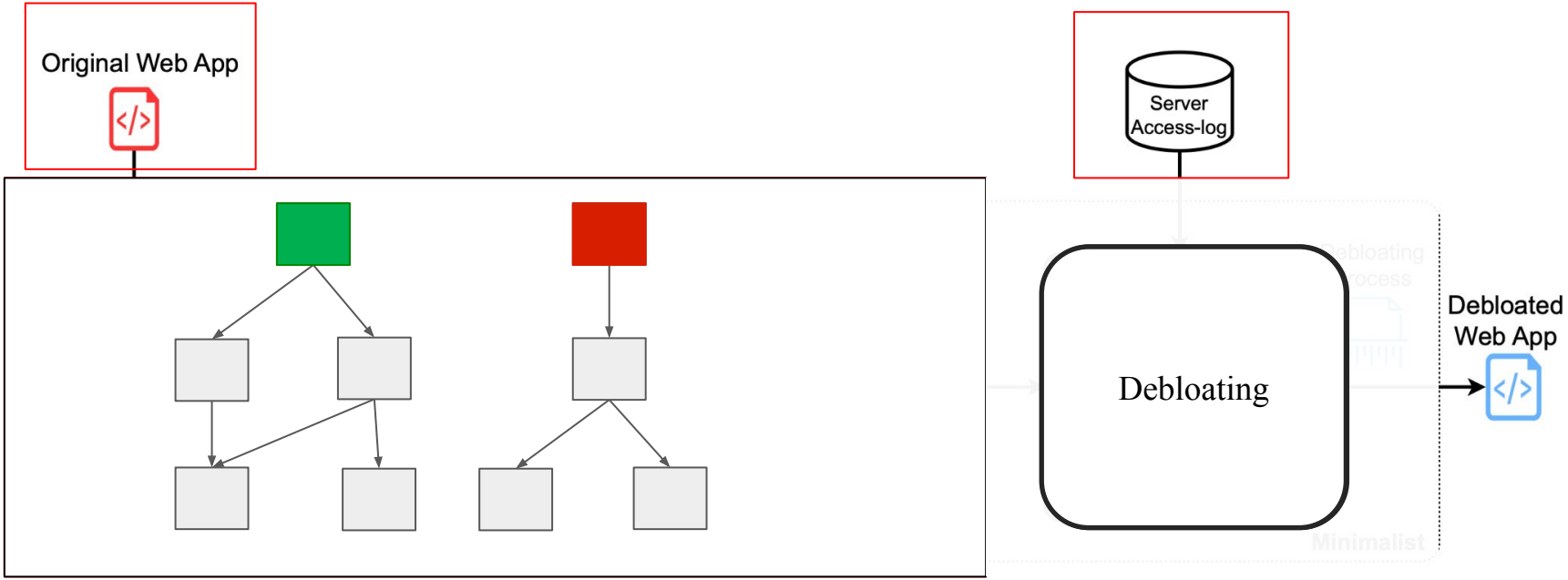
Introducing **Minimalist** (***& AnimateDead - next presentation***)

- Static reachability analysis on the web server logs

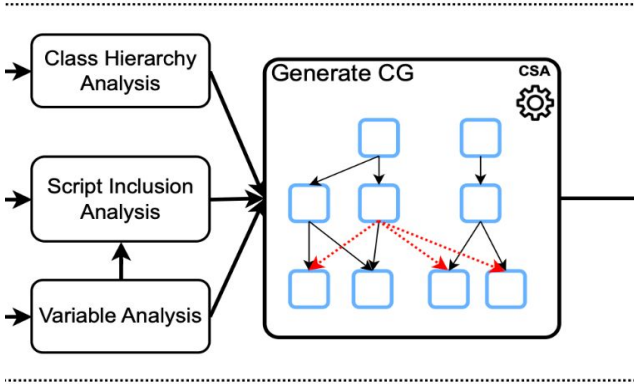


Minimalist - Overview

Minimalist proposes a semi-automated static approach to debloat web apps



Minimalist - Call Graph



Minimalist – Generate call graph

Not always easy to generate call-graph

- Variable script inclusion
- Variable function call
- Object oriented programming

test.php

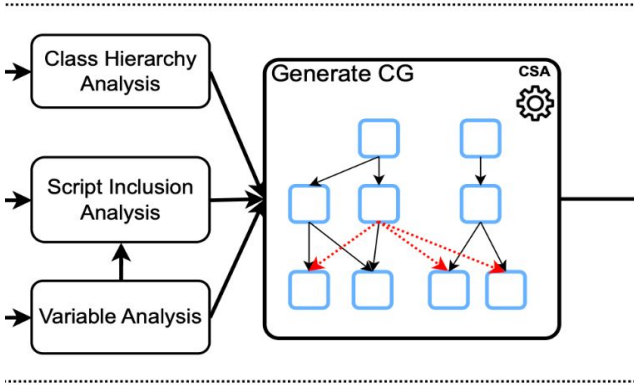
```
1.  define ( 'classpath', __DIR__ );
2.  $included = classpath ."/Class";
3.  include_once $included . '.php';
4.  $type = "ChildClass";
5.  $obj = new $type ;
6.  $method = "call";
7.  $obj->$method();
```

Class.php

```
1.  class ParentClass {
2.      public $feature = 0;
3.      public function __construct () {
4.          $this->feature = 1;
5.  }
6.      public function Cprint (){
7.          echo $this->feature."\n";
8.      }
9.  }
10. class ChildClass extends ParentClass {
11.     public function call() {
12.      call_user_func(array($this, 'Cprint'));
13.    }
14. }
```

Minimalist - Call Graph

- Minimalist performs three analyses before generating the call-graph
  - Class Hierarchy
    - Identify the inheritance relationship
  - Script Inclusion
    - Generate the script dependency graph
  - Variable Analysis
    - Determine the assigned values to variables
- Generate the call-graph of the web app
  - Use prior analysis when necessary



Minimalist – Custom Static Analysis

- Web applications could use certain dynamic code structures pose a challenge for static analysis

```
1.function test () {
2.// Retrieve the callable action from the database
3.$query = "SELECT * FROM actions WHERE " .$conds;
4.$result_db = mysql_query($query);
5.// Assign the value to the variable action
6.
7.$action = mysql_fetch_row ($result_db);
8.// Invoke the retrieved function name
9.// from the database
10.$result = $action();
11.}
```

- Minimalist provide a plugin API for analysts
  - Written in Go
  - Write analysis snippet (CSA)
  - Update the call graph

```
1.  list_actions = Get the list of function calls
2.  foreach list_actions.Next() {
3.      // grab items from the list of actions
4.      var item = list_actions.Scan(&item)
5.      // update the call-graph of function test
6.      // with the retrieved function name
7.      update_callgraph( "test", "actions.php", item)
8.  }
```

Minimalist - Evaluation

- Evaluated on 4 popular web applications
  WordPress, Joomla, Drupal, and phpMyAdmin
- Mapped 45 CVEs to their source code

Minimalist	LIM
18% size reduction	+ 53% size reduction
38% removal of vulnerabilities	+ 73% removal of vulnerabilities
+ No breakage after debloating	Likely to result in breakage

Conclusion

- Minimalist
  - Analyzes PHP application to generate the call-graph
  - Integrates information collected from web server
  - Debloating functions/file from the PHP application

- Takeaway
  - We can debloat web applications without incurring performance overhead while maintaining the usability
- Our artifacts are open-source and available at:

https://debloating.com

References

[1] Amin Azad, Babak, and Nick Nikiforakis. Role Models: Role-based Debloating for Web Applications. In Proceedings of the 13th ACM Conference on Data and Application Security and Privacy. 2023.

[2] https://www.imperva.com/blog/the-resurrection-of-phpunit-rce-vulnerability/

[3] Babak Amin Azad, Pierre Laperdrix, and Nick Nikiforakis. Less is more: Quantifying the security benefits of debloating web applications. In Proceedings of the 28th USENIX Conference on Security Symposium, 2019.

[4] https://www.liquidweb.com/kb/exporting-databases-and-tables-with-phpmyadmin/

[5] Illustrations from: https://refactoring.guru/