opentext™

Fortify Developer Workbook

2/13/24

SCA

Report Overview

Report Summary

On Feb 9, 2024, a source code review was performed over the Biosamiemain code base. 8,740 files, 231,517 LOC (Executable) were scanned. A total of 64 issues were uncovered during the analysis. This report provides a comprehensive description of all the types of issues found in this project. Specific examples and source code are provided for each issue type.

Issues by Fortify Priority Order	
Low	36
High	25
Critical	3

Issue Summary

Overall number of results

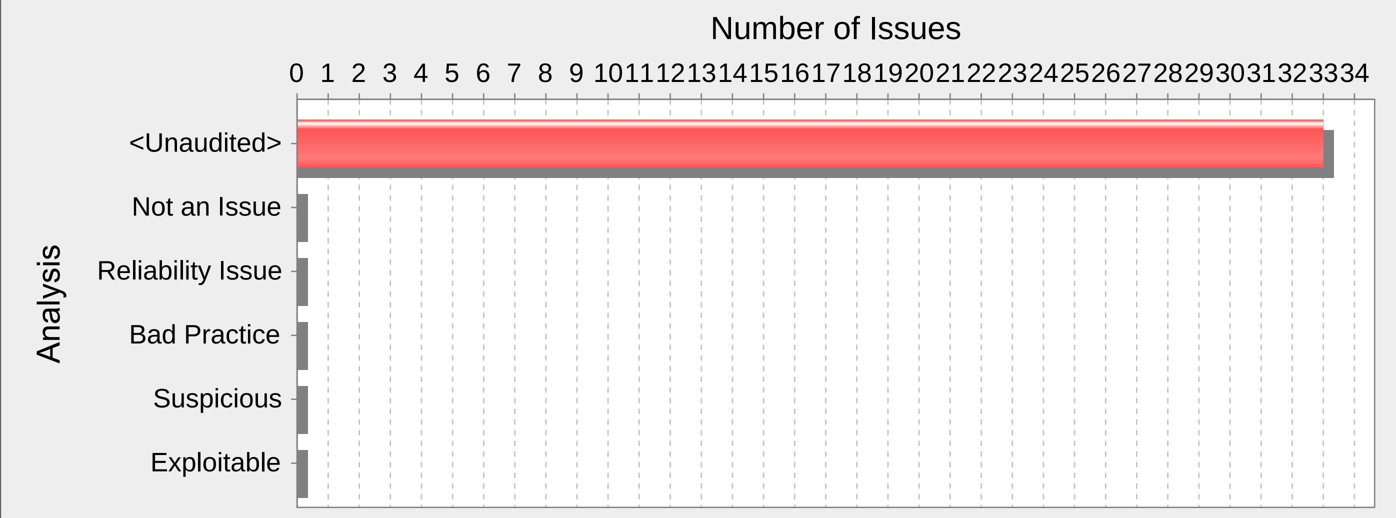
The scan found 64 issues.

Issues by Category	
Weak Cryptographic Hash	33
Path Manipulation	24
XML Injection	3
System Information Leak: External	2
Cookie Security: Cookie not Sent Over SSL	1
Cookie Security: HTTPOnly not Set	1

Results Outline

Vulnerability Examples by Category

Category: Weak Cryptographic Hash (33 Issues)



Abstract:

Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Explanation:

MD2, MD4, MD5, RIPEMD-160, and SHA-1 are popular cryptographic hash algorithms often used to verify the integrity of messages and other data. However, as recent cryptanalysis research has revealed fundamental weaknesses in these algorithms, they should no longer be used within security-critical contexts.

Effective techniques for breaking MD and RIPEMD hashes are widely available, so those algorithms should not be relied upon for security. In the case of SHA-1, current techniques still require a significant amount of computational power and are more difficult to implement. However, attackers have found the Achilles' heel for the algorithm, and techniques for breaking it will likely lead to the discovery of even faster attacks.

Recommendations:

Discontinue the use of MD2, MD4, MD5, RIPEMD-160, and SHA-1 for data-verification in security-critical contexts. Currently, SHA-224, SHA-256, SHA-384, SHA-512, and SHA-3 are good alternatives. However, these variants of the Secure Hash Algorithm have not been scrutinized as closely as SHA-1, so be mindful of future research that might impact the security of these algorithms.

Tips:

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see the Fortify Static Code Analyzer User Guide.

Request.php, line 640 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	Request.php:640 sha1()		

```
638                 }
639
640                 return sha1(implode('|', array_merge(
641                     $route->methods(),
642                     [$route->getDomain(), $route->uri(), $this->ip()]
```

ThrottlesExceptions.php, line 163 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	ThrottlesExceptions.php:163 md5()		

```
161                 }
```

```
162
163                 return $this->prefix.md5(get_class($job));
164             }
```

ThrottleRequests.php, line 172 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: ThrottleRequests.php:172 sha1()

```
170                     return sha1($user->getAuthIdentifier());
171                 } elseif ($route = $request->route()) {
172                     return sha1($route->getDomain().'|'.$request->ip());
173                 }
```

RedisTaggedCache.php, line 131 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: RedisTaggedCache.php:131 sha1()

```
129             protected function pushKeys($namespace, $key, $reference)
130             {
131                 $fullKey = $this->store->getPrefix().sha1($namespace).':'.$key;
132
133                 foreach (explode('|', $namespace) as $segment) {
```

Event.php, line 953 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: Event.php:953 sha1()

```
951                 }
952
953                 return 'framework'.DIRECTORY_SEPARATOR.'schedule-'.sha1($this-
                >expression.$this->command);
954             }
```

Compiler.php, line 84 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: Compiler.php:84 sha1()

```
82              public function getCompiledPath($path)
83              {
84                  return $this->cachePath.'/'.sha1('v2'.Str::after($path, $this-
                >basePath)).'.'.$this->compiledExtension;
85              }
```

Filesystem.php, line 531 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: Filesystem.php:531 md5_file()

```
529             public function hasSameHash($firstFile, $secondFile)
```

```
        {
            $hash = @md5_file($firstFile);

            return $hash && $hash === @md5_file($secondFile);
```

CachingFileAnalyser.php, line 176 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	CachingFileAnalyser.php:176 md5()		

```
    private function cacheFile(string $filename): string
    {
        $cacheKey = md5(
            implode(
                "\0",
```

OpenAiSolution.php, line 74 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	OpenAiSolution.php:74 sha1()		

```
    protected function getCacheKey(): string
    {
        $hash = sha1($this->prompt);

        return "ignition-solution-{$hash}";
```

ThrottleRequests.php, line 170 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	ThrottleRequests.php:170 sha1()		

```
    {
        if ($user = $request->user()) {
            return sha1($user->getAuthIdentifier());
        } elseif ($route = $request->route()) {
            return sha1($route->getDomain().'|'.$request->ip());
```

CompilesComponents.php, line 49 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	CompilesComponents.php:49 sha1()		

```
    public static function newComponentHash(string $component)
    {
        static::$componentHashStack[] = $hash = sha1($component);

        return $hash;
```

Version.php, line 54 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		

Sink: Version.php:54 sha1()

```
        if (mt_rand(0, 1)) {
            // short git revision syntax: https://git-scm.com/book/en/v2/Git-Tools-
Revision-Selection
            return substr(sha1(Helper::lexify('??????')), 0, 7);
        }
```

Event.php, line 485 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: Event.php:485 sha1()

```
    {
        if (is_null($this->output) || $this->output == $this->getDefaultOutput()) {
            $this->sendOutputTo(storage_path('logs/schedule-'.sha1($this-
>mutexName()).'.log'));
        }
    }
```

NotPwnedVerifier.php, line 71 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: NotPwnedVerifier.php:71 sha1()

```
    protected function getHash($value)
    {
        $hash = strtoupper(sha1((string) $value));

        $hashPrefix = substr($hash, 0, 5);
```

CallbackEvent.php, line 189 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: CallbackEvent.php:189 sha1()

```
    public function mutexName()
    {
        return 'framework/schedule-'.sha1($this->description ?? '');
    }
```

CookieValuePrefix.php, line 16 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: CookieValuePrefix.php:16 hash_hmac()

```
    public static function create($cookieName, $key)
    {
        return hash_hmac('sha1', $cookieName.'v2', $key).'|';
    }
```

TaggedCache.php, line 102 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: TaggedCache.php:102 sha1()

```
100     public function taggedItemKey($key)
101     {
102         return sha1($this->tags->getNamespace()).':'.$key;
103     }
```

SetCacheHeaders.php, line 34 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: SetCacheHeaders.php:34 md5()

```
32
33          if (isset($options['etag']) && $options['etag'] === true) {
34              $options['etag'] = $response->getEtag() ?? md5($response->getContent());
35          }
```

Container.php, line 247 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: Container.php:247 md5()

```
245     {
246         $keys = array_keys($this->_mocks);
247         $match = preg_grep("/__demeter_" . md5($parent) . "_{$method}$/", $keys);
248         if (count($match) == 1) {
249             $res = array_values($match);
```

CachingFileAnalyser.php, line 205 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: CachingFileAnalyser.php:205 md5()

```
203         }
204
205         self::$cacheVersion = md5(implode("\0", $buffer));
206
207         return self::$cacheVersion;
```

FileStore.php, line 329 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: FileStore.php:329 sha1()

```
327     protected function path($key)
328     {
329         $parts = array_slice(str_split($hash = sha1($key), 2), 0, 2);
330
331         return $this->directory.'/'.implode('/', $parts).'/'.$hash;
```

RateLimited.php, line 70 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low

Kingdom:	Security Features
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.
Sink:	RateLimited.php:70 md5()

```
68              collect(Arr::wrap($limiterResponse))->map(function ($limit) {
69                  return (object) [
70                      'key' => md5($this->limiterName.$limit->key),
71                      'maxAttempts' => $limit->maxAttempts,
72                      'decayMinutes' => $limit->decayMinutes,
```

AliasLoader.php, line 103 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	AliasLoader.php:103 sha1()		

```
101     protected function ensureFacadeExists($alias)
102     {
103         if (is_file($path = storage_path('framework/cache/facade-
'.sha1($alias).'.php'))) {
104             return $path;
105         }
```

SessionGuard.php, line 799 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	SessionGuard.php:799 sha1()		

```
797     public function getName()
798     {
799         return 'login_'.$this->name.'_'.sha1(static::class);
800     }
```

EmailVerificationRequest.php, line 21 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	EmailVerificationRequest.php:21 sha1()		

```
19          }
20
21          if (! hash_equals(sha1($this->user()->getEmailForVerification()), (string)
$this->route('hash'))) {
22              return false;
23          }
```

VerifyEmail.php, line 88 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	VerifyEmail.php:88 sha1()		

```
86              [
87                  'id' => $notifiable->getKey(),
88                  'hash' => sha1($notifiable->getEmailForVerification()),
89              ]
90          );
```

ThrottleRequests.php, line 97 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	ThrottleRequests.php:97 md5()		

```
95              collect(Arr::wrap($limiterResponse))->map(function ($limit) use
   ($limiterName) {
96                  return (object) [
97                      'key' => md5($limiterName.$limit->key),
98                      'maxAttempts' => $limit->maxAttempts,
99                      'decayMinutes' => $limit->decayMinutes,
```

SessionGuard.php, line 809 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	SessionGuard.php:809 sha1()		

```
807     public function getRecallerName()
808     {
809         return 'remember_'.$this->name.'_'.sha1(static::class);
810     }
```

BladeCompiler.php, line 798 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	BladeCompiler.php:798 md5()		

```
796     public function anonymousComponentPath(string $path, string $prefix = null)
797     {
798         $prefixHash = md5($prefix ?: $path);
799
800         $this->anonymousComponentPaths[] = [
```

Component.php, line 190 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	Component.php:190 sha1()		

```
188         );
189
190         if (! is_file($viewFile = $directory.'/'.sha1($contents).'.blade.php')) {
191             if (! is_dir($directory)) {
192                 mkdir($directory, 0755, true);
```

ManagesLayouts.php, line 181 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.		
Sink:	ManagesLayouts.php:181 sha1()		

```
179             $salt = static::parentPlaceholderSalt();
180
```

```
181                 static::$parentPlaceholder[$section] = '##parent-placeholder-
          '.sha1($salt.$section).'##';
182             }
```

Vite.php, line 722 (Weak Cryptographic Hash)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: Vite.php:722 md5_file()

```
720             }
721
722             return md5_file($path) ?: null;
723         }
```

ReflectionClosure.php, line 812 (Weak Cryptographic Hash)

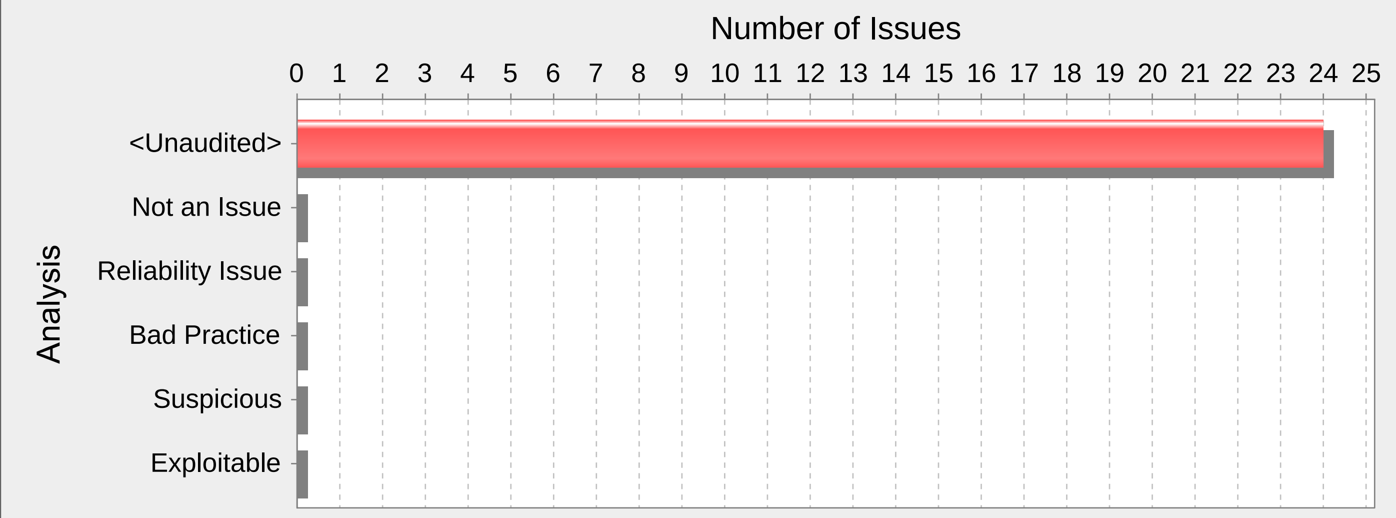
Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: Weak cryptographic hashes cannot guarantee data integrity and should not be used in security-critical contexts.

Sink: ReflectionClosure.php:812 sha1()

```
810         {
811             if ($this->hashedName === null) {
812                 $this->hashedName = sha1($this->getFileName());
813             }
```

Category: Path Manipulation (24 Issues)



Abstract:

Attackers can control the file system path argument to file_get_contents() at Cpdf.php line 1169, which allows them to access or modify otherwise protected files.

Explanation:

Path manipulation errors occur when the following two conditions are met:

1. An attacker can specify a path used in an operation on the file system.

2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program might give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

Example 1: The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files.

```
$rName = $_GET['reportName'];
$rFile = fopen("/usr/local/apfr/reports/" . rName,"a+");
...
unlink($rFile);
```

Example 2: The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with adequate privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

```
...
$filename = $CONFIG_TXT['sub'] . ".txt";
$handle = fopen($filename,"r");
$amt = fread($handle, filesize($filename));
echo $amt;
...
```

Recommendations:

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate values from which the user must select. With this approach, the user-provided input is never used directly to specify the resource name.

In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to maintain. Programmers often resort to implementing a deny list in these situations. A deny list is used to selectively reject or escape potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a list of characters that are permitted to appear in the resource name and accept input composed exclusively of characters in the approved set.

Tips:

1. If the program performs custom input validation to your satisfaction, use the Fortify Custom Rules Editor to create a cleanse rule for the validation routine.

2. Implementation of an effective deny list is notoriously difficult. One should be skeptical if validation logic requires implementing a deny list. Consider different types of input encoding and different sets of metacharacters that might have special meaning when interpreted by different operating systems, databases, or other resources. Determine whether or not the deny list can be updated easily, correctly, and completely if these requirements ever change.

3. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see the Fortify Static Code Analyzer User Guide.

FileProfilerStorage.php, line 290 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to fopen() at FileProfilerStorage.php line 290, which allows them to access or modify otherwise protected files.

Source: FileProfilerStorage.php:292 stream_get_contents()

```
        $h = fopen($file, 'r');
        flock($h, \LOCK_SH);
        $data = stream_get_contents($h);
        flock($h, \LOCK_UN);
        fclose($h);
```

Sink: FileProfilerStorage.php:290 fopen()

```
        }

        $h = fopen($file, 'r');
        flock($h, \LOCK_SH);
        $data = stream_get_contents($h);
```

Font.php, line 55 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at Font.php line 55, which allows them to access or modify otherwise protected files.

Source: Cpdf.php:3354 file()

```
            }

            $file = file("$dir/$metrics_name");

            foreach ($file as $rowA) {
```

Sink: Font.php:55 file_get_contents()

```
      // Unknown type or EOT
      default:
        $magicNumber = file_get_contents($file, false, null, 34, 2);

        if ($magicNumber === "LP") {
```

FileProfilerStorage.php, line 332 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to unlink() at FileProfilerStorage.php line 332, which allows them to access or modify otherwise protected files.

Source: FileProfilerStorage.php:317 fgets()

```
        }

        while ($line = fgets($handle)) {
            $values = str_getcsv($line);
```

Sink: FileProfilerStorage.php:332 unlink()

```
            }

            @unlink($this->getFilename($csvToken));
```

```
333                     $offset += \strlen($line);
334                 }
```

PhptTestCase.php, line 528 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at PhptTestCase.php line 528, which allows them to access or modify otherwise protected files.

Source: PhptTestCase.php:460 file()

```
458             $lineNr = 0;
459
460             foreach (file($this->filename) as $line) {
461                 $lineNr++;
```

Sink: PhptTestCase.php:528 file_get_contents()

```
526                     }
527
528                     $sections[$section] = file_get_contents($testDirectory .
$externalFilename);
529                 }
530             }
```

EditCommand.php, line 166 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at EditCommand.php line 166, which allows them to access or modify otherwise protected files.

Source: Transient.php:117 fgets()

```
115             echo $prompt;
116
117             return \rtrim(\fgets($this->getStdin()), "\n\r");
118         }
```

Sink: EditCommand.php:166 file_get_contents()

```
164             \proc_close($proc);
165
166             $editedContent = @\file_get_contents($filePath);
167
168             if ($shouldRemoveFile) {
```

Configuration.php, line 631 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to mkdir() at Configuration.php line 631, which allows them to access or modify otherwise protected files.

Source: SystemEnv.php:30 getenv()

```
28              }
29
30              $result = \getenv($key);
31
32              return $result === false ? null : $result;
```

Sink: Configuration.php:631 mkdir()

```
629
630             if (!\is_dir($runtimeDir)) {
631                 if (!@\mkdir($runtimeDir, 0700, true)) {
632                     throw new RuntimeException(\sprintf('Unable to create PsySH runtime
directory. Make sure PHP is able to write to %s in order to continue.',
\dirname($runtimeDir)));
633                 }
```

AbstractDumper.php, line 71 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to fopen() at AbstractDumper.php line 71, which allows them to access or modify otherwise protected files.

Source: FileLinkFormatter.php:38 Read $_ENV['SYMFONY_IDE']()

```
36        public function __construct(string|array $fileLinkFormat = null, RequestStack
          $requestStack = null, string $baseDir = null, string|\Closure $urlFormat = null)
37        {
38            $fileLinkFormat ??= $_ENV['SYMFONY_IDE'] ?? $_SERVER['SYMFONY_IDE'] ?? '';
39
40            if (!\is_array($f = $fileLinkFormat)) {
```

Sink: AbstractDumper.php:71 fopen()

```
69            } else {
70                if (\is_string($output)) {
71                    $output = fopen($output, 'w');
72                }
73                $this->outputStream = $output;
```

AbstractDumper.php, line 71 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to fopen() at AbstractDumper.php line 71, which allows them to access or modify otherwise protected files.

Source: Request.php:302 Read $_POST()

```
300       public static function createFromGlobals(): static
301       {
302           $request = self::createRequestFromFactory($_GET, $_POST, [], $_COOKIE,
          $_FILES, $_SERVER);
303
304           if (str_starts_with($request->headers->get('CONTENT_TYPE', ''),
          'application/x-www-form-urlencoded')
```

Sink: AbstractDumper.php:71 fopen()

```
69            } else {
70                if (\is_string($output)) {
71                    $output = fopen($output, 'w');
72                }
73                $this->outputStream = $output;
```

Cpdf.php, line 1169 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at Cpdf.php line 1169, which allows them to access or modify otherwise protected files.

Source: Cpdf.php:3354 file()

```
3352              }
3353
3354              $file = file("$dir/$metrics_name");
3355
3356              foreach ($file as $rowA) {
```

Sink: Cpdf.php:1169 file_get_contents()

```
1167              // simple utility to convert them from pfa to pfb.
1168              if (!$font['isSubsetting']) {
1169                  $data = file_get_contents($fbfile);
1170              } else {
1171                  $adobeFontName = $this->getFontSubsettingTag($font) . '+' .
          $adobeFontName;
```

DefaultTestResultCache.php, line 109 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at DefaultTestResultCache.php line 109, which allows them to access or modify otherwise protected files.

Source: DefaultTestResultCache.php:75 Read $_ENV['PHPUNIT_RESULT_CACHE']()

```
73              }
74
75              $this->cacheFilename = $filepath ?? $_ENV['PHPUNIT_RESULT_CACHE'] ??
    self::DEFAULT_RESULT_CACHE_FILENAME;
76          }
```

Sink: DefaultTestResultCache.php:109 file_get_contents()

```
107
108             $data = json_decode(
109                 file_get_contents($this->cacheFilename),
110                 true,
111             );
```

EditCommand.php, line 169 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to unlink() at EditCommand.php line 169, which allows them to access or modify otherwise protected files.

Source: Transient.php:117 fgets()

```
115             echo $prompt;
116
117             return \rtrim(\fgets($this->getStdin()), "\n\r");
118         }
```

Sink: EditCommand.php:169 unlink()

```
167
168             if ($shouldRemoveFile) {
169                 @\unlink($filePath);
170             }
```

AbstractDumper.php, line 71 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to fopen() at AbstractDumper.php line 71, which allows them to access or modify otherwise protected files.

Source: Request.php:302 Read $_GET()

```
300         public static function createFromGlobals(): static
301         {
302             $request = self::createRequestFromFactory($_GET, $_POST, [], $_COOKIE,
    $_FILES, $_SERVER);
303
304             if (str_starts_with($request->headers->get('CONTENT_TYPE', ''),
    'application/x-www-form-urlencoded')
```

Sink: AbstractDumper.php:71 fopen()

```
69              } else {
70                  if (\is_string($output)) {
71                      $output = fopen($output, 'w');
72                  }
73                  $this->outputStream = $output;
```

CPdf.php, line 1173 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract:	Attackers can control the file system path argument to file_get_contents() at CPdf.php line 1173, which allows them to access or modify otherwise protected files.

Source: CPdf.php:3309 file()

```
3307                }
3308
3309                $file = file($dir . $metrics_name);
3310
3311                foreach ($file as $rowA) {
```

Sink: CPdf.php:1173 file_get_contents()

```
1171                // note that pdf supports only binary format type 1 font files, though
        there is a
1172                // simple utility to convert them from pfa to pfb.
1173                $data = file_get_contents($fbfile);
1174
1175                // create the font descriptor
```

extract-tentative-return-types.php, line 48 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract:	Attackers can control the file system path argument to file_get_contents() at extract-tentative-return-types.php line 48, which allows them to access or modify otherwise protected files.

Source: extract-tentative-return-types.php:47 fgets()

```
45      EOPHP;
46
47      while (false !== $file = fgets(\STDIN)) {
48          $code = file_get_contents(substr($file, 0, -1));
```

Sink: extract-tentative-return-types.php:48 file_get_contents()

```
46
47      while (false !== $file = fgets(\STDIN)) {
48          $code = file_get_contents(substr($file, 0, -1));
49
50          if (!str_contains($code, '@tentative-return-type')) {
```

Validator.php, line 27 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract:	Attackers can control the file system path argument to file_get_contents() at Validator.php line 27, which allows them to access or modify otherwise protected files.

Source: Version.php:84 stream_get_contents()

```
82              }
83
84              $result = \trim(\stream_get_contents($pipes[1]));
85
86              \fclose($pipes[1]);
```

Sink: Validator.php:27 file_get_contents()

```
25              $originalErrorHandling = libxml_use_internal_errors(true);
26
27              $document->schemaValidateSource(file_get_contents($xsdFilename));
28
29              $errors = libxml_get_errors();
```

AbstractDumper.php, line 71 (Path Manipulation)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Input Validation and Representation		

Abstract:	Attackers can control the file system path argument to fopen() at AbstractDumper.php line 71, which allows them to access or modify otherwise protected files.

Source: FileLinkFormatter.php:38 Read $_ENV['SYMFONY_IDE']()

```
36          public function __construct(string|array $fileLinkFormat = null, RequestStack
    $requestStack = null, string $baseDir = null, string|\Closure $urlFormat = null)
37          {
38              $fileLinkFormat ??= $_ENV['SYMFONY_IDE'] ?? $_SERVER['SYMFONY_IDE'] ?? '';
39
40              if (!\is_array($f = $fileLinkFormat)) {
```

Sink: AbstractDumper.php:71 fopen()

```
69              } else {
70                  if (\is_string($output)) {
71                      $output = fopen($output, 'w');
72                  }
73                  $this->outputStream = $output;
```

TextPart.php, line 153 (Path Manipulation)

Fortify Priority: High **Folder** High

Kingdom: Input Validation and Representation

Abstract: Attackers can control the file system path argument to fopen() at TextPart.php line 153, which allows them to access or modify otherwise protected files.

Source: TextPart.php:141 stream_get_contents()

```
139             }
140
141             return stream_get_contents($this->body) ?: '';
142         }
```

Sink: TextPart.php:153 fopen()

```
151             if ($this->body instanceof File) {
152                 $path = $this->body->getPath();
153                 if (false === $handle = @fopen($path, 'r', false)) {
154                     throw new InvalidArgumentException(sprintf('Unable to open path
    "%s".', $path));
155                 }
```

AbstractDumper.php, line 71 (Path Manipulation)

Fortify Priority: High **Folder** High

Kingdom: Input Validation and Representation

Abstract: Attackers can control the file system path argument to fopen() at AbstractDumper.php line 71, which allows them to access or modify otherwise protected files.

Source: Request.php:302 Read $_COOKIE()

```
300         public static function createFromGlobals(): static
301         {
302             $request = self::createRequestFromFactory($_GET, $_POST, [], $_COOKIE,
    $_FILES, $_SERVER);
303
304             if (str_starts_with($request->headers->get('CONTENT_TYPE', ''),
    'application/x-www-form-urlencoded')
```

Sink: AbstractDumper.php:71 fopen()

```
69              } else {
70                  if (\is_string($output)) {
71                      $output = fopen($output, 'w');
72                  }
73                  $this->outputStream = $output;
```

TextPart.php, line 130 (Path Manipulation)

Fortify Priority: High **Folder** High

Kingdom: Input Validation and Representation

Abstract: Attackers can control the file system path argument to file_get_contents() at TextPart.php line 130, which allows them to access or modify otherwise protected files.

Source: TextPart.php:141 stream_get_contents()

```
139             }
140
```

```
                return stream_get_contents($this->body) ?: '';
            }
```

Sink: TextPart.php:130 file_get_contents()

```
            {
                if ($this->body instanceof File) {
                    return file_get_contents($this->body->getPath());
                }
```

DefaultTestResultCache.php, line 109 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at DefaultTestResultCache.php line 109, which allows them to access or modify otherwise protected files.

Source: PhpHandler.php:114 getenv()

```
                    }

                    $value = getenv($name);

                    if ($force || !isset($_ENV[$name])) {
```

Sink: DefaultTestResultCache.php:109 file_get_contents()

```

                $data = json_decode(
                    file_get_contents($this->cacheFilename),
                    true,
                );
```

TextPart.php, line 153 (Path Manipulation)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to fopen() at TextPart.php line 153, which allows them to access or modify otherwise protected files.

Source: Request.php:1506 file_get_contents()

```

                if (null === $this->content || false === $this->content) {
                    $this->content = file_get_contents('php://input');
                }
```

Sink: TextPart.php:153 fopen()

```
                if ($this->body instanceof File) {
                    $path = $this->body->getPath();
                    if (false === $handle = @fopen($path, 'r', false)) {
                        throw new InvalidArgumentException(sprintf('Unable to open path
"%s".', $path));
                    }
```

TextPart.php, line 153 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to fopen() at TextPart.php line 153, which allows them to access or modify otherwise protected files.

Source: Request.php:1502 stream_get_contents()

```
                    rewind($this->content);

                    return stream_get_contents($this->content);
                }
```

Sink: TextPart.php:153 fopen()

```
                if ($this->body instanceof File) {
                    $path = $this->body->getPath();
                    if (false === $handle = @fopen($path, 'r', false)) {
```

```
                    throw new InvalidArgumentException(sprintf('Unable to open path
"%s".', $path));
                }
```

Font.php, line 31 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at Font.php line 31, which allows them to access or modify otherwise protected files.

Source: Cpdf.php:3354 file()

```
            }

            $file = file("$dir/$metrics_name");

            foreach ($file as $rowA) {
```

Sink: Font.php:31 file_get_contents()

```
        }

    $header = file_get_contents($file, false, null, 0, 4);
    $class  = null;
```

Cpdf.php, line 1226 (Path Manipulation)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: Attackers can control the file system path argument to file_get_contents() at Cpdf.php line 1226, which allows them to access or modify otherwise protected files.

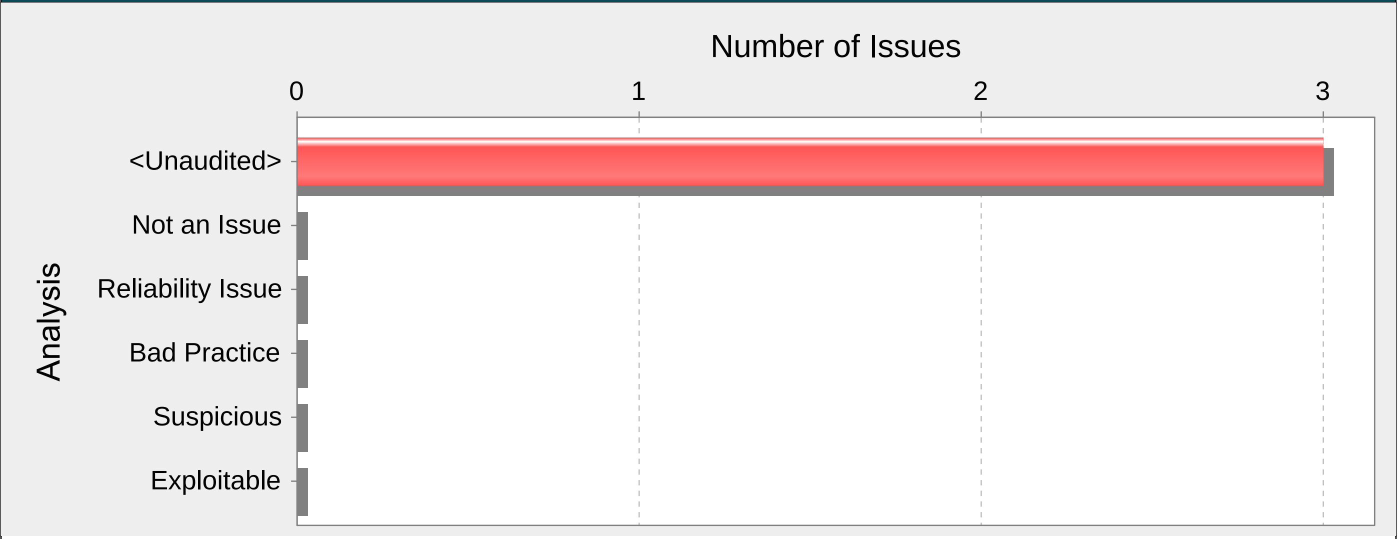
Source: Cpdf.php:3354 file()

```
            }

            $file = file("$dir/$metrics_name");

            foreach ($file as $rowA) {
```

Sink: Cpdf.php:1226 file_get_contents()

```
                    $data = file_get_contents($tmp_name);
                } else {
                    $data = file_get_contents($fbfile);
                }
```

Category: XML Injection (3 Issues)



Abstract:

On line 176 of Cache.php, the method resolve_url() processes unvalidated XML input. This call could allow an attacker to inject arbitrary elements or attributes into the body of an XML document, cause a denial of service, or leak sensitive information. XML injection is different from XML external entity (XXE) injection because the attacker usually controls input inserted into the middle or end of an XML document.

Explanation:

XML injection occurs when:

1. Data enters a program from an untrusted source.

2. The data is written to an XML document or parsed as XML.

Applications typically use XML to store data or send messages. When used to store data, XML documents are often treated like databases and can potentially contain sensitive information. XML messages are often used in web services and can also be used to transmit sensitive information. XML messages can even be used to send authentication credentials.

The semantics of XML documents and messages can be altered if an attacker has the ability to write raw XML. In the most benign case, an attacker may be able to insert extraneous tags and cause an XML parser to throw an exception. In more nefarious cases of XML injection, an attacker may be able to add XML elements that change authentication credentials or modify prices in an XML e-commerce database. In some cases, XML injection can lead to cross-site scripting or dynamic code evaluation.

Example 1:

Assume an attacker is able to control shoes in following XML.

```
<order>
<price>100.00</price>
<item>shoes</item>
</order>
```

Now suppose this XML is included in a back end web service request to place an order for a pair of shoes. Suppose the attacker modifies his request and replaces shoes with shoes</item><price>1.00</price><item>shoes. The new XML would look like:

```
<order>
<price>100.00</price>
<item>shoes</item><price>1.00</price><item>shoes</item>
</order>
```

When using XML parsers, the value from the second <price> overrides the value from the first <price> tag. This allows the attacker to purchase a pair of $100 shoes for $1.

A more serious form of this attack called XML External Entity (XXE) injection can occur when the attacker controls the front or all of the parsed XML document.

Example 2: Here is some code that is vulnerable to XXE attacks:

Assume an attacker is able to control the input XML to the following code:

```
...
<?php
```

```
$goodXML = $_GET["key"];
$doc = simplexml_load_string($goodXml);
echo $doc->testing;
?>
...
```

Now suppose that the following XML is passed by the attacker to the code in Example 2:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>
```

When the XML is processed, the content of the <foo> element is populated with the contents of the system's boot.ini file. The attacker may utilize XML elements which are returned to the client to exfiltrate data or obtain information as to the existence of network resources.

Recommendations:

When writing user-supplied data to XML, follow these guidelines:

1. Do not create tags or attributes with names that are derived from user input.

2. XML entity encode user input before writing to XML.

3. Wrap user input in CDATA tags.

When writing user-supplied data to XML, or parsing unvalidated XML, in order to mitigate XML external entity injection (XXE), you have the following options:

1. Disable entity expansion by the XML parser in use.

```
libxml_disable_entity_loader(true);
```

or

```
$dom->resolveExternals=false;
```

2. XML entity encode user input before writing to XML.

3. If you need to allow entity expansion then:
a. Validate the input to make sure the expanded entities are appropriate and allowed.
b. Limit what the parser can do while loading the XML:

```
$doc = XMLReader::xml($badXml,'UTF-8',LIBXML_NONET);
```

Tips:

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see the Fortify Static Code Analyzer User Guide.

Cache.php, line 176 (XML Injection)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: On line 176 of Cache.php, the method resolve_url() processes unvalidated XML input. This call could allow an attacker to inject arbitrary elements or attributes into the body of an XML document, cause a denial of service, or leak sensitive information. XML injection is different from XML external entity (XXE) injection because the attacker usually controls input inserted into the middle or end of an XML document.

Source: Cache.php:175 fread()

```
173
174                                     if (($fp = fopen($resolved_url, "r")) !== false) {
175                                         while ($line = fread($fp, 8192)) {
176                                             xml_parse($parser, $line, false);
177                                         }
```

Sink: Cache.php:176 xml_parse()

```
174                                     if (($fp = fopen($resolved_url, "r")) !== false) {
175                                         while ($line = fread($fp, 8192)) {
```

```
176                                             xml_parse($parser, $line, false);
177                                         }
178                                         fclose($fp);
```

Document.php, line 227 (XML Injection)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: On line 227 of Document.php, the method render() processes unvalidated XML input. This call could allow an attacker to inject arbitrary elements or attributes into the body of an XML document, cause a denial of service, or leak sensitive information. XML injection is different from XML external entity (XXE) injection because the attacker usually controls input inserted into the middle or end of an XML document.

Source: Document.php:226 fread()

```
224
225             $fp = fopen($this->filename, "r");
226             while ($line = fread($fp, 8192)) {
227                 xml_parse($parser, $line, false);
228             }
```

Sink: Document.php:227 xml_parse()

```
225             $fp = fopen($this->filename, "r");
226             while ($line = fread($fp, 8192)) {
227                 xml_parse($parser, $line, false);
228             }
```

Document.php, line 130 (XML Injection)

Fortify Priority:	High	**Folder**	High
Kingdom:	Input Validation and Representation		

Abstract: On line 130 of Document.php, the method getdimensions() processes unvalidated XML input. This call could allow an attacker to inject arbitrary elements or attributes into the body of an XML document, cause a denial of service, or leak sensitive information. XML injection is different from XML external entity (XXE) injection because the attacker usually controls input inserted into the middle or end of an XML document.

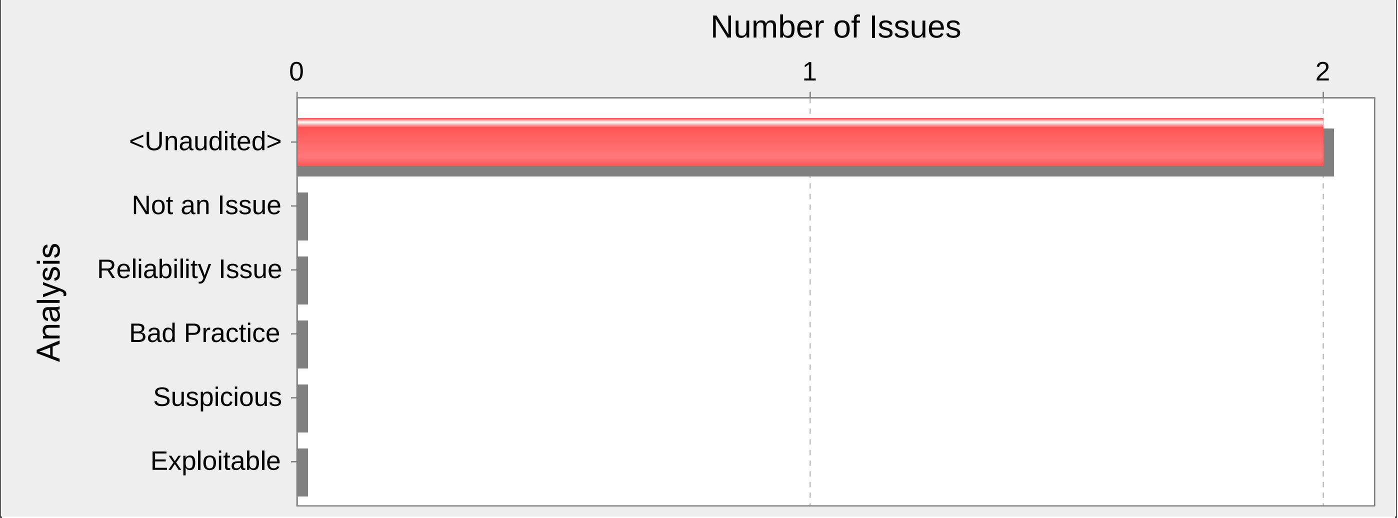
Source: Document.php:129 fread()

```
127
128             $fp = fopen($this->filename, "r");
129             while ($line = fread($fp, 8192)) {
130                 xml_parse($parser, $line, false);
```

Sink: Document.php:130 xml_parse()

```
128             $fp = fopen($this->filename, "r");
129             while ($line = fread($fp, 8192)) {
130                 xml_parse($parser, $line, false);
131
132                 if ($rootAttributes !== null) {
```

Category: System Information Leak: External (2 Issues)



Abstract:

The program might reveal system data or debugging information in HtmlErrorRenderer.php with a call to highlight_file() on line 264. The information revealed by highlight_file() could help an adversary form a plan of attack.

Explanation:

An external information leak occurs when system data or debugging information leaves the program to a remote machine via a socket or network connection.

Example 1: The following code writes an exception to the HTTP response:

```
<?php
...
echo "Server error! Printing the backtrace";
debug_print_backtrace();
...
?>
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. For example, with scripting mechanisms it is trivial to redirect output information from "Standard error" or "Standard output" into a file or another program. Alternatively, the system that the program runs on could have a remote logging mechanism such as a "syslog" server that sends the logs to a remote device. During development, you have no way of knowing where this information might end up being displayed.

In some cases, the error message provides the attacker with the precise type of attack to which the system is vulnerable. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In Example 1, the leaked information could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Recommendations:

Write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Debug traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example).

Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

Tips:

1. Do not rely on wrapper scripts, corporate IT policy, or quick-thinking system administrators to prevent system information leaks. Write software that is secure on its own.

2. This category of vulnerability does not apply to all types of programs. For example, if your application executes on a client machine where system information is already available to an attacker, or if you print system information only to a trusted log file, you can use Audit Guide to filter out this category from your scan results.

3. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see the Fortify Static Code Analyzer User Guide.

HtmlErrorRenderer.php, line 264 (System Information Leak: External)

Fortify Priority:	Critical	**Folder**	Critical

Kingdom:	Encapsulation
Abstract:	The program might reveal system data or debugging information in HtmlErrorRenderer.php with a call to highlight_file() on line 264. The information revealed by highlight_file() could help an adversary form a plan of attack.
Sink:	HtmlErrorRenderer.php:264 highlight_file()

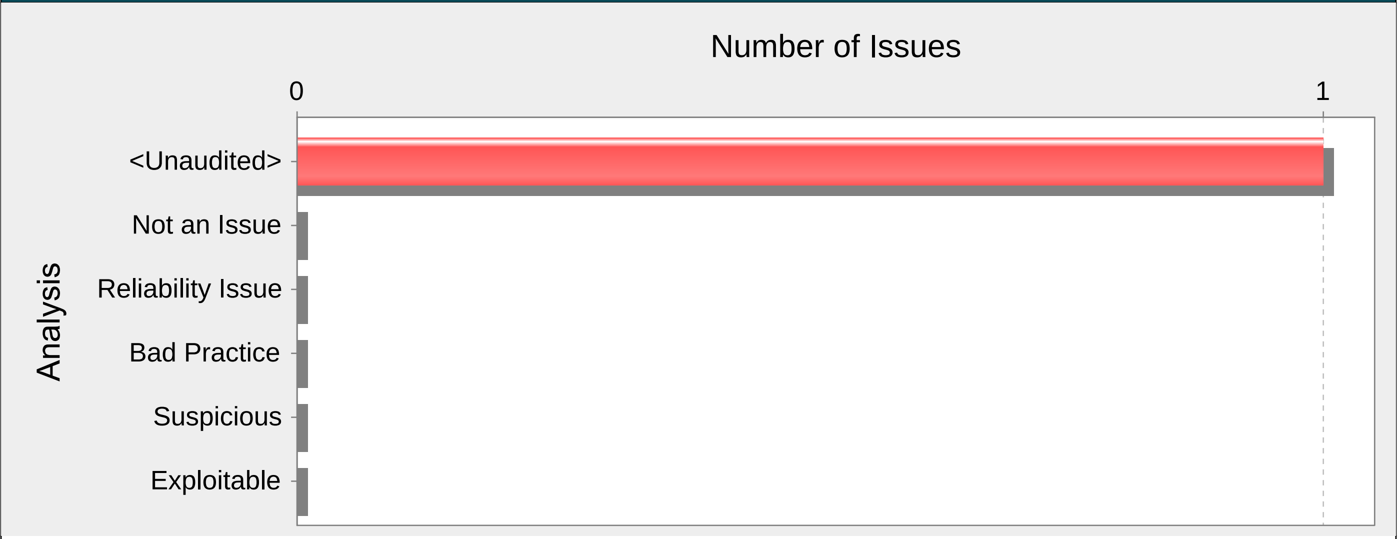
```
262                         // highlight_file could throw warnings
263                         // see https://bugs.php.net/25725
264                         $code = @highlight_file($file, true);
265                         // remove main code/span tags
266                         $code = preg_replace('#^<code.*?>\s*<span.*?>(.*)</span>\s*</code>#s',
'\\1', $code);
```

Process.php, line 1321 (System Information Leak: External)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Encapsulation		
Abstract:	The program might reveal system data or debugging information in Process.php with a call to phpinfo() on line 1321. The information revealed by phpinfo() could help an adversary form a plan of attack.		
Sink:	Process.php:1321 phpinfo()		

```
1319
1320                ob_start();
1321                phpinfo(\INFO_GENERAL);
1322
1323                return self::$sigchild = str_contains(ob_get_clean(), '--enable-sigchild');
```

Category: Cookie Security: Cookie not Sent Over SSL (1 Issues)



Abstract:

The program creates a cookie without setting the Secure flag to true

Explanation:

Modern web browsers support a Secure flag for each cookie. If the flag is set, the browser will only send the cookie over HTTPS. Sending cookies over an unencrypted channel can expose them to network sniffing attacks, so the secure flag helps keep a cookie's value confidential. This is especially important if the cookie contains private data or carries a session identifier.

Example 1: The following code adds a cookie to the response without setting the Secure flag.

```
...
setcookie("emailCookie", $email, 0, "/", "www.example.com");
...
```

If an application uses both HTTPS and HTTP, but does not set the Secure flag, cookies sent during an HTTPS request will also be sent during subsequent HTTP requests. Attackers may then compromise the cookie by sniffing the unencrypted network traffic, which is particularly easy over wireless networks.

Recommendations:

Set the Secure flag on all new cookies in order to instruct browsers not to send these cookies in the clear. This can be accomplished by passing true as the sixth argument to setcookie().

Example 2: The following code corrects the mistake in Example 1 by setting the Secure flag to true.

```
setcookie("emailCookie", $email, 0, "/", "www.example.com", TRUE);
```

Tips:

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see the Fortify Static Code Analyzer User Guide.

AbstractSessionHandler.php, line 105 (Cookie Security: Cookie not Sent Over SSL)

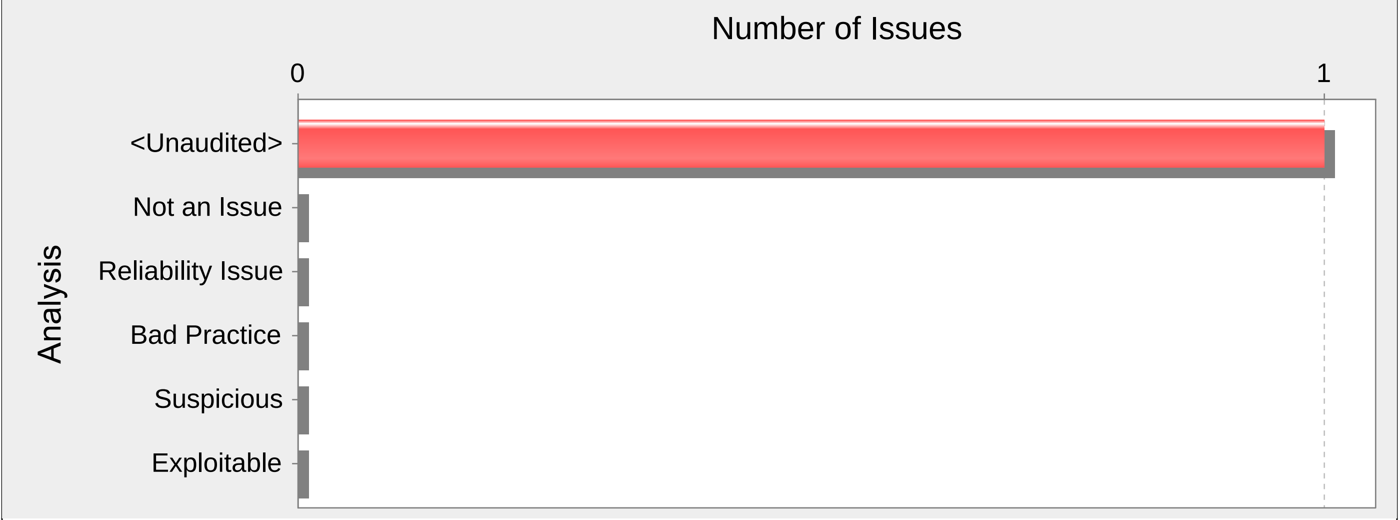
Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: The program creates a cookie without setting the Secure flag to true

Sink: AbstractSessionHandler.php:105 setcookie()

```
103                     $params = session_get_cookie_params();
104                     unset($params['lifetime']);
105                     setcookie($this->sessionName, '', $params);
106                 }
107             }
```

Category: Cookie Security: HTTPOnly not Set (1 Issues)



Abstract:

The program creates a cookie in AbstractSessionHandler.php on line 105, but fails to set the HttpOnly flag to true.

Explanation:

All major browsers support the HttpOnly cookie property that prevents client-side scripts from accessing the cookie. Cross-site scripting attacks often access cookies in an attempt to steal session identifiers or authentication tokens. Without HttpOnly enabled, attackers have easier access to user cookies.

Example 1: The following code creates a cookie without setting the HttpOnly property.

```
setcookie("emailCookie", $email, 0, "/", "www.example.com", TRUE);  //Missing 7th parameter to set HttpOnly
```

Recommendations:

Enable the HttpOnly property when you create cookies. You can do this by setting the HttpOnly parameter in the setcookie() call to true.

Example 2: The following code creates the same cookie as the code in Example 1, but this time sets the HttpOnly parameter to true.

```
setcookie("emailCookie", $email, 0, "/", "www.example.com", TRUE, TRUE);
```

Several mechanisms to bypass setting HttpOnly to true have been developed, and therefore it is not completely effective.

Tips:

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see the Fortify Static Code Analyzer User Guide.

AbstractSessionHandler.php, line 105 (Cookie Security: HTTPOnly not Set)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		

Abstract: The program creates a cookie in AbstractSessionHandler.php on line 105, but fails to set the HttpOnly flag to true.

Sink: AbstractSessionHandler.php:105 setcookie()

```
103                 $params = session_get_cookie_params();
104                 unset($params['lifetime']);
105                 setcookie($this->sessionName, '', $params);
106             }
107         }
```