leepeuker / **movary** Public

Code | Issues 71 | Pull requests 18 | Discussions | Actions | Security and

Commit d459b35

leepeuker authored last week · ✓ 1 / 1 · Verified

```
Merge pull request #751 from leepeuker/ssfr-GHSA-2m2v-v563-qqvj

Add basic SSFR check for Jellyfin and Plex
```

main (#751) · 0.71.1

2 parents 12c8a09 + 5e65df2 commit d459b35

11 files changed +329 -34 lines changed

Top

Filter files…

- .env.example
- bootstrap.php
- docs
  - configuration.md
  - features
    - jellyfin.md
    - plex.md
- public/js
  - settings-integration-jellyfin.js
- src
  - Factory.php
  - HttpController/Web
    - JellyfinController.php
    - PlexController.php
  - Util

UrlValidator.php

ValueObject/Exception

InvalidSafeUrl.php

11 files changed +329 -34 lines changed

Search within code

.env.example

```
@@ -63,10 +63,16 @@ TMDB_API_KEY=
# PLEX_IDENTIFIER=
# PLEX_APP_NAME=Movary

+ # Enable SSRF protection for Plex server URLs (blocks localhost, private IPs, internal DNS)
+ # PLEX_VALIDATE_URL_SAFE=0
+
# Required for Plex Authentication. Generate with e.g. openssl rand -base64 32
# JELLYFIN_DEVICE_ID=
# JELLYFIN_APP_NAME=Movary

+ # Enable SSRF protection for Jellyfin server URLs (blocks localhost, private IPs, internal DNS)
+ # JELLYFIN_VALIDATE_URL_SAFE=0
+
###############
### DOCKER ###
###############
```

bootstrap.php

```
@@ -17,6 +17,8 @@
            \Movary\HttpController\Api\OpenApiController::class => DI\factory([Factory::class, 'createOpenApiController']),
            \Movary\HttpController\Web\JobController::class => DI\factory([Factory::class, 'createJobController']),
            \Movary\HttpController\Web\LandingPageController::class => DI\factory([Factory::class, 'createLandingPageController']),
+           \Movary\HttpController\Web\JellyfinController::class => DI\factory([Factory::class, 'createJellyfinController']),
```

```
21  + \Movary\HttpController\Web\PlexController::class =>
    DI\factory([Factory::class, 'createPlexController']),
20  22
    \Movary\HttpController\Web\Middleware\ServerHasRegistrationEnabled::class =>
    DI\factory([Factory::class, 'createMiddlewareServerHasRegistrationEnabled']),
21  23 \Movary\ValueObject\Http\Request::class => DI\factory([Factory::class,
    'createCurrentHttpRequest']),
22  24 \Movary\Command\CreatePublicStorageLink::class =>
    DI\factory([Factory::class, 'createCreatePublicStorageLink']),
```

docs/configuration.md

```
    @@ -48,12 +48,14 @@ Required to run the application
48  48
49  49 Required for some third party integrations. Only necessary if the relevant third
    party integrations should be enabled.
50  50
51    - | NAME                | DEFAULT VALUE | INFO
      | Web UI |
52    - |:--------------------|:-------------:|:---------------------------------------
      ----------------------------------------|:------:|
53    - | `PLEX_IDENTIFIER`   |       -       | Required for Plex Authentication.
      Generate with e.g. `openssl rand -base64 32`     |        |
54    - | `PLEX_APP_NAME`     |   `Movary`    | Used for Plex Authentication
      |        |
55    - | `JELLYFIN_DEVICE_ID` |       -       | Required for Jellyfin Authentication.
      Generate with e.g. `openssl rand -base64 32` |        |
56    - | `JELLYFIN_APP_NAME`  |   `Movary`    | Used for Jellyfin Authentication
      |        |
      51 + | NAME                          | DEFAULT VALUE | INFO
         | Web UI |
      52 + |:------------------------------|:-------------:|:-------------------------------
         ----------------------------------------------------------------|:------:|
      53 + | `PLEX_IDENTIFIER`             |       -       | Required for Plex
         Authentication. Generate with e.g. `openssl rand -base64 32`
         |        |
      54 + | `PLEX_APP_NAME`               |   `Movary`    | Used for Plex Authentication
         |        |
      55 + | `PLEX_VALIDATE_URL_SAFE`      |      `0`      | Enable SSRF protection for Plex
         server URLs (blocks localhost, private IPs, internal DNS)          |        |
```

```
+ | `JELLYFIN_DEVICE_ID`         |       -       | Required for Jellyfin Authentication. Generate with e.g. `openssl rand -base64 32`                |
+ | `JELLYFIN_APP_NAME`          |   `Movary`    | Used for Jellyfin Authentication                                                                  |
+ | `JELLYFIN_VALIDATE_URL_SAFE` |      `0`      | Enable SSRF protection for Jellyfin server URLs (blocks localhost, private IPs, internal DNS)     |
```

```
  ### Email
```

docs/features/jellyfin.md

```
@@ -33,6 +33,45 @@ During the authentication process a Jellyfin access token is generated and store
  This token will be used in all further Jellyfin API requests.
  When an authentication is removed from Movary, the token will be deleted in Movary and the Jellyfin server.

+ ## URL Validation and SSRF Protection
+
+ ### Overview
+
+ Movary can validate Jellyfin server URLs to protect against Server-Side Request Forgery (SSRF) attacks.
+
+ ### Security Features
+
+ When enabled, URL validation blocks:
+
+ - **Localhost access** (localhost, 127.0.0.1, ::1)
+ - **Private IP ranges** (192.168.x.x, 10.x.x.x, 172.16-31.x.x)
+ - **Internal DNS names** (.internal, .local, .docker, .corp, .lan, .home, .priv)
+ - **Cloud metadata endpoints** (169.254.169.254, metadata.google.internal, etc.)
+ - **Suspicious ports** (only allows 80, 443, 8096, 8920)
+ - **DNS rebinding attacks**
+
+ ### Configuration
```

```
+
+ Enable SSRF protection by setting the environment variable:
+
+ ```bash
+ JELLYFIN_VALIDATE_URL_SAFE=1
+ ```
+
+ !!! warning
+
+     Enabling this feature will break existing configurations that use:
+     - Localhost Jellyfin servers
+     - Private network IP addresses
+     - Internal DNS names
+
+     Ensure your Jellyfin server is accessible via a public domain name before enabling.
+
+ ### Recommendations
+
+ - **Enable in public environments** for enhanced security
+ - **Keep disabled** for development or when using localhost/internal networks
+
  ## Sync

  General notes:
```

docs/features/plex.md

```
@@ -36,6 +36,45 @@ When an authentication is removed from Movary, the token will be deleted only in
      Removing the authentication only deletes the token stored in Movary itself. The token still exists in Plex.
      To invalidate the access token in Plex, go to your Plex settings at: Account -> Authorized devices -> Click on the red cross for the entry "Movary"

+ ## URL Validation and SSRF Protection
+
+ ### Overview
+
```

```
+ Movary can validate Plex server URLs to protect against Server-Side Request Forgery (SSRF) attacks.
+
+ ### Security Features
+
+ When enabled, URL validation blocks:
+
+ - **Localhost access** (localhost, 127.0.0.1, ::1)
+ - **Private IP ranges** (192.168.x.x, 10.x.x.x, 172.16-31.x.x)
+ - **Internal DNS names** (.internal, .local, .docker, .corp, .lan, .home, .priv)
+ - **Cloud metadata endpoints** (169.254.169.254, metadata.google.internal, etc.)
+ - **Suspicious ports** (only allows 80, 443, 8096, 8920)
+ - **DNS rebinding attacks**
+
+ ### Configuration
+
+ Enable SSRF protection by setting the environment variable:
+
+ ```bash
+ PLEX_VALIDATE_URL_SAFE=1
+ ```
+
+ !!! warning
+
+     Enabling this feature will break existing configurations that use:
+     - Localhost Plex servers
+     - Private network IP addresses
+     - Internal DNS names
+
+     Ensure your Plex server is accessible via a public domain name before enabling.
+
+ ### Recommendations
+
+ - **Enable in public environments** for enhanced security
+ - **Keep disabled** for development or when using localhost/internal networks
+
  ## Watchlist import

  ### Description
```

public/js/settings-integration-jellyfin.js

```
@@ -152,7 +152,7 @@ async function verifyJellyfinServerUrl() {
152 152
153 153         if (!response.ok) {
154 154             if (response.status === 400) {
155     -             addAlert('alertJellyfinAuthenticationModalDiv', await response.text(), 'danger')
    155 +             addAlert('alertJellyfinServerUrlDiv', await response.text(), 'danger')
156 156
157 157                 return
158 158             }
```

src/Factory.php

```
@@ -211,6 +211,35 @@ public static function createLandingPageController(ContainerInterface $container
211 211             );
212 212         }
213 213
    214 +     public static function createJellyfinController(ContainerInterface $container, Config $config) : HttpController\Web\JellyfinController
    215 +     {
    216 +         return new HttpController\Web\JellyfinController(
    217 +             $container->get(Authentication::class),
    218 +             $container->get(UserApi::class),
    219 +             $container->get(Service\Jellyfin\JellyfinScrobbler::class),
    220 +             $container->get(Service\WebhookUrlBuilder::class),
    221 +             $container->get(LoggerInterface::class),
    222 +             $container->get(Api\Jellyfin\JellyfinApi::class),
    223 +             $container->get(Util\UrlValidator::class),
    224 +             $config->getAsBool('JELLYFIN_VALIDATE_URL_SAFE', false),
    225 +         );
    226 +     }
    227 +
    228 +     public static function createPlexController(ContainerInterface $container, Config $config) : HttpController\Web\PlexController
```

```
+     {
+         return new HttpController\Web\PlexController(
+             $container->get(Authentication::class),
+             $container->get(UserApi::class),
+             $container->get(Service\Plex\PlexScrobbler::class),
+             $container->get(Api\Plex\PlexApi::class),
+             $container->get(Service\WebhookUrlBuilder::class),
+             $container->get(LoggerInterface::class),
+             $container->get(Service\ApplicationUrlService::class),
+             $container->get(Util\UrlValidator::class),
+             $config->getAsBool('PLEX_VALIDATE_URL_SAFE', false),
+         );
+     }
+
      public static function createLineFormatter(Config $config) : LineFormatter
      {
          $formatter = new LineFormatter(LineFormatter::SIMPLE_FORMAT,
LineFormatter::SIMPLE_DATE);
```

src/HttpController/Web/JellyfinController.php

```
@@ -4,15 +4,13 @@

 use Exception;
 use Movary\Api\Jellyfin\Exception\JellyfinInvalidAuthentication;
-use Movary\Api\Jellyfin\Exception\JellyfinNotFoundError;
-use Movary\Api\Jellyfin\Exception\JellyfinServerConnectionError;
-use Movary\Api\Jellyfin\Exception\JellyfinServerUrlMissing;
 use Movary\Api\Jellyfin\JellyfinApi;
 use Movary\Domain\User\Service\Authentication;
 use Movary\Domain\User\UserApi;
 use Movary\Service\Jellyfin\JellyfinScrobbler;
 use Movary\Service\WebhookUrlBuilder;
 use Movary\Util\Json;
+use Movary\Util\UrlValidator;
 use Movary\ValueObject\Exception\InvalidUrl;
 use Movary\ValueObject\Http\Request;
 use Movary\ValueObject\Http\Response;
@@ -28,6 +26,8 @@ public function __construct(
         private readonly WebhookUrlBuilder $webhookUrlBuilder,
```

```
29  27              private readonly LoggerInterface $logger,
30  28              private readonly JellyfinApi $jellyfinApi,
    29  +           private readonly UrlValidator $urlValidator,
    30  +           private readonly bool $validateUrlIsSafe = false,
31  31          ) {
32  32          }
33  33
    @@ -44,13 +44,8 @@ public function authenticateJellyfinAccount(Request $request) : Response
44  44
45  45              try {
46  46                  $jellyfinAuthentication = $this->jellyfinApi->createJellyfinAuthentication($userId, $username, $password);
47      -           } catch (JellyfinServerUrlMissing) {
48      -               return Response::createBadRequest('Could not authenticate: Server url missing');
49      -           } catch (JellyfinNotFoundError) {
50      -               return Response::createBadRequest('Could not authenticate: Page not found');
51      -           } catch (JellyfinServerConnectionError) {
52      -               return Response::createBadRequest('Could not authenticate: Cannot connect to server');
53      -           } catch (JellyfinInvalidAuthentication) {
    47  +           } catch (Exception $e) {
    48  +               $this->logger->warning('Jellyfin could not authenticate: ' . $e->getMessage());
54  49                  return Response::createBadRequest('Could not authenticate');
55  50              }
56  51
    @@ -82,7 +77,7 @@ public function handleJellyfinWebhook(Request $request) : Response
82  77              $requestPayload = $request->getBody();
83  78
84  79              $this->logger->debug('Jellyfin: Webhook triggered with payload: ' . $requestPayload);
85      -           $this->logger->warning('This jellyfin webhook url is deprecated and will stop to work soon, regenerate the url');
    80  +           $this->logger->warning('Jellyfin: This webhook url is deprecated and will stop to work soon, regenerate the url');
86  81
```

```
87  82          $this->jellyfinScrobbler->processJellyfinWebhook($userId,
                Json::decode($requestPayload));
88  83
    @@ -108,7 +103,7 @@ public function removeJellyfinAuthentication() :
    Response
108 103         try {
109 104             $this->jellyfinApi-
                >deleteJellyfinAccessToken($jellyfinAuthentication);
110 105         } catch (Exception) {
111     -             $this->logger->warning('Could not delete jellyfin remote access
                token for user: ' . $userId);
        106 +         $this->logger->warning('Jellyfin: Could not delete remote access
                token for user: ' . $userId);
112 107         }
113 108
114 109         $this->userApi->deleteJellyfinAuthentication($userId);
    @@ -120,21 +115,31 @@ public function removeJellyfinAuthentication() :
    Response
120 115
121 116     public function saveJellyfinServerUrl(Request $request) : Response
122 117     {
123     -         $jellyfinServerUrl = Json::decode($request->getBody())
                ['JellyfinServerUrl'];
        118 +         $jellyfinServerUrlString = Json::decode($request->getBody())
                ['JellyfinServerUrl'];
124 119         $userId = $this->authenticationService->getCurrentUserId();
125 120
126     -         if (empty($jellyfinServerUrl)) {
        121 +         if (empty($jellyfinServerUrlString)) {
127 122             $this->userApi->updateJellyfinServerUrl($userId, null);
128 123
129 124             return Response::createOk();
130 125         }
131 126
132 127         try {
133     -             $jellyfinServerUrl = Url::createFromString($jellyfinServerUrl);
        128 +             $jellyfinServerUrl =
                Url::createFromString($jellyfinServerUrlString);
134 129         } catch (InvalidUrl) {
```

```
                130 +             $this->logger->info('Jellyfin: Provided server url is not a valid url: ' . $jellyfinServerUrlString);
135             131               return Response::createBadRequest('Provided server url is not a valid url');
136             132           }
137             133
                134 +         if ($this->validateUrlIsSafe === true) {
                135 +             try {
                136 +                 $this->urlValidator->validateUrlIsSafe($jellyfinServerUrl);
                137 +             } catch (InvalidUrl $e) {
                138 +                 $this->logger->warning('Jellyfin: Could not safe server ur: ' . $e->getMessage());
                139 +                 return Response::createBadRequest('Could not safe server url');
                140 +             }
                141 +         }
                142 +
138             143           $this->userApi->updateJellyfinServerUrl($userId, $jellyfinServerUrl);
139             144
140             145           return Response::createOk();
                  @@ -152,7 +157,24 @@ public function saveJellyfinSyncOptions(Request $request) : Response
152             157
153             158       public function verifyJellyfinServerUrl(Request $request) : Response
154             159       {
155             -             $jellyfinServerUrl = Url::createFromString(Json::decode($request->getBody())['jellyfinServerUrl']);
                160 +         $jellyfinServerUrlString = Json::decode($request->getBody())['jellyfinServerUrl'];
                161 +
                162 +         try {
                163 +             $jellyfinServerUrl = Url::createFromString($jellyfinServerUrlString);
                164 +         } catch (InvalidUrl) {
                165 +             $this->logger->info('Jellyfin: Provided server url is not a valid url: ' . $jellyfinServerUrlString);
                166 +             return Response::createBadRequest('Provided server url is not a valid url');
                167 +         }
                168 +
                169 +         if ($this->validateUrlIsSafe === true) {
```

```
      170 +             try {
      171 +                 $this->urlValidator->validateUrlIsSafe($jellyfinServerUrl);
      172 +             } catch (InvalidUrl $e) {
      173 +                 $this->logger->warning('Jellyfin: Could not safe server ur: ' . $e->getMessage());
      174 +                 return Response::createBadRequest('Could not safe server url');
      175 +             }
      176 +         }
      177 +
156   178           $jellyfinAuthentication = $this->userApi->findJellyfinAuthentication($this->authenticationService->getCurrentUserId());
157   179
158   180           $authenticationVerified = $jellyfinAuthentication !== null;
          @@ -164,16 +186,16 @@ public function verifyJellyfinServerUrl(Request $request) : Response
164   186               } else {
165   187                   $jellyfinServerInfo = $this->jellyfinApi->fetchJellyfinServerInfo($jellyfinServerUrl, $jellyfinAuthentication->getAccessToken());
166   188               }
167       -         } catch (JellyfinNotFoundError) {
168       -             return Response::createBadRequest('Connection test failed: Page not found');
169       -         } catch (JellyfinServerConnectionError) {
170       -             return Response::createBadRequest('Connection test failed: Cannot connect to server');
171   189           } catch (JellyfinInvalidAuthentication) {
172   190               $authenticationVerified = false;
      191 +         } catch (Exception $e) {
      192 +             $this->logger->warning('Jellyfin: Connection test failed: ' . $e->getMessage());
      193 +             return Response::createBadRequest('Connection test failed');
173   194           }
174   195
175   196           if ($jellyfinServerInfo === null || empty($jellyfinServerInfo['Id']) === true) {
176       -             return Response::createBadRequest('Connection test failed: Jellyfin response invalid');
      197 +             $this->logger->warning('Jellyfin: Connection test failed: Jellyfin response invalid');
```

```
                198 +             return Response::createBadRequest('Connection test failed');
177             199               }
178             200
179             201               return Response::createJson(Json::encode(['serverUrlVerified' => true,
                    'authenticationVerified' => $authenticationVerified]));
```

src/HttpController/Web/PlexController.php

```
                    @@ -10,12 +10,11 @@
10              10    use Movary\Service\Plex\PlexScrobbler;
11              11    use Movary\Service\WebhookUrlBuilder;
12              12    use Movary\Util\Json;
                13  + use Movary\Util\UrlValidator;
13              14    use Movary\ValueObject\Exception\ConfigNotSetException;
14              15    use Movary\ValueObject\Exception\InvalidUrl;
15                  - use Movary\ValueObject\Http\Header;
16              16    use Movary\ValueObject\Http\Request;
17              17    use Movary\ValueObject\Http\Response;
18                  - use Movary\ValueObject\Http\StatusCode;
19              18    use Movary\ValueObject\RelativeUrl;
20              19    use Movary\ValueObject\Url;
21              20    use Psr\Log\LoggerInterface;
                    @@ -31,6 +30,8 @@ public function __construct(
31              30            private readonly WebhookUrlBuilder $webhookUrlBuilder,
32              31            private readonly LoggerInterface $logger,
33              32            private readonly ApplicationUrlService $applicationUrlService,
                33  +         private readonly UrlValidator $urlValidator,
                34  +         private readonly bool $validateUrlIsSafe = false,
34              35        ) {
35              36        }
36              37
                    @@ -143,8 +144,12 @@ public function savePlexServerUrl(Request $request) :
                    Response
143             144
144             145           try {
145             146               $plexServerUrl = Url::createFromString($plexServerUrl);
146                 -         } catch (InvalidUrl) {
147                 -             return Response::createBadRequest('Url not properly formatted');
                147 +             if ($this->validateUrlIsSafe === true) {
                148 +                 $this->urlValidator->validateUrlIsSafe($plexServerUrl);
```

```
                149 +             }
                150 +         } catch (InvalidUrl $e) {
                151 +             $this->logger->warning('Plex: Could not safe server url: ' . $e->getMessage());
                152 +             return Response::createBadRequest('Could not safe server url');
148             153           }
149             154
150             155           $this->userApi->updatePlexServerUrl($userId, $plexServerUrl);
                    @@ -163,8 +168,12 @@ public function verifyPlexServerUrl(Request $request) : Response
163             168
164             169           try {
165             170               $plexServerUrl = Url::createFromString($plexServerUrl);
166             -         } catch (InvalidUrl) {
167             -             return Response::createBadRequest('Provided server url is not a valid url');
                171 +             if ($this->validateUrlIsSafe === true) {
                172 +                 $this->urlValidator->validateUrlIsSafe($plexServerUrl);
                173 +             }
                174 +         } catch (InvalidUrl $e) {
                175 +             $this->logger->warning('Plex: Connection test failed: ' . $e->getMessage());
                176 +             return Response::createBadRequest('Connection test failed');
168             177           }
169             178
170             179           $userClientConfiguration = PlexUserClientConfiguration::create($plexAccessToken, $plexServerUrl);
```

src/Util/UrlValidator.php

```
                    @@ -0,0 +1,136 @@
                1 + <?php declare(strict_types=1);
                2 +
                3 + namespace Movary\Util;
                4 +
                5 + use Movary\ValueObject\Exception\InvalidSafeUrl;
                6 + use Movary\ValueObject\Url;
                7 +
                8 + class UrlValidator
                9 + {
```

```
10  +     private const array BLOCKED_HOSTS
11  +         = [
12  +             'metadata.google.internal',
13  +             '169.254.169.254',  // AWS metadata
14  +             'metadata.azure.internal',
15  +             'metadata.rackspace.com',
16  +             '169.254.169.254/latest/meta-data/',
17  +             'metadata.service.consul',
18  +             '169.254.169.254/latest',
19  +         ];
20  +
21  +     private const array ALLOWED_LOCALHOST
22  +         = [
23  +             'localhost',
24  +             '127.0.0.1',
25  +             '::1',
26  +         ];
27  +
28  +     private const array INTERNAL_DNS_PATTERNS
29  +         = [
30  +             '.internal',
31  +             '.local',
32  +             '.docker',
33  +             '.corp',
34  +             '.lan',
35  +             '.home',
36  +             '.priv',
37  +         ];
38  +
39  +     private const array ALLOWED_PORTS = [80, 443, 8096, 8920]; // 8096 and 8920
      are Jellyfin defaults
40  +
41  +     public function validateUrlIsSafe(Url $url): void
42  +     {
43  +         $parsedUrl = parse_url((string)$url);
44  +
45  +         if ($parsedUrl === false || !isset($parsedUrl['host'])) {
46  +             throw InvalidSafeUrl::create((string)$url);
47  +         }
48  +
```

```
49  +         $host = strtolower($parsedUrl['host']);
50  +         $port = $parsedUrl['port'] ?? (($parsedUrl['scheme'] ?? null) ===
      'https' ? 443 : 80);
51  +
52  +         $this->validateHost($host);
53  +         $this->validatePort($port);
54  +         $this->validateResolvedIp($host);
55  +     }
56  +
57  +     private function validateHost(string $host): void
58  +     {
59  +         // Block known internal/metadata endpoints
60  +         foreach (self::BLOCKED_HOSTS as $blocked) {
61  +             if (str_contains($host, $blocked)) {
62  +                 throw new InvalidSafeUrl('Blocked internal host: ' . $host);
63  +             }
64  +         }
65  +
66  +         // Block localhost
67  +         if ($this->isAllowedLocalhost($host)) {
68  +             throw new InvalidSafeUrl('Localhost access not allowed: ' . $host);
69  +         }
70  +
71  +         // Block private IP ranges
72  +         if ($this->isPrivateIp($host)) {
73  +             throw new InvalidSafeUrl('Private IP address not allowed: ' .
      $host);
74  +         }
75  +
76  +         // Block internal DNS patterns
77  +         if ($this->isInternalDns($host)) {
78  +             throw new InvalidSafeUrl('Internal DNS name not allowed: ' .
      $host);
79  +         }
80  +     }
81  +
82  +     private function validatePort(int $port): void
83  +     {
84  +         // Block suspicious ports (except common web ports)
85  +         if (!in_array($port, self::ALLOWED_PORTS, true)) {
```

```
86  +             throw new InvalidSafeUrl('Port not allowed: ' . $port);
87  +         }
88  +     }
89  +
90  +     private function validateResolvedIp(string $host): void
91  +     {
92  +         if (filter_var($host, FILTER_VALIDATE_IP)) {
93  +             return;
94  +         }
95  +
96  +         // Resolve hostname to IP addresses
97  +         $ips = gethostbynamel($host);
98  +
99  +         if ($ips === false) {
100 +             throw new InvalidSafeUrl('Could not resolve hostname: ' . $host);
101 +         }
102 +
103 +         foreach ($ips as $ip) {
104 +             // Check if resolved IP is private (but allow localhost)
105 +             if ($this->isPrivateIp($ip) && !$this->isAllowedLocalhost($ip)) {
106 +                 throw new InvalidSafeUrl('Hostname resolves to private IP: ' .
      $host . ' -> ' . $ip);
107 +             }
108 +         }
109 +     }
110 +
111 +     private function isAllowedLocalhost(string $host): bool
112 +     {
113 +         return in_array($host, self::ALLOWED_LOCALHOST, true);
114 +     }
115 +
116 +     private function isPrivateIp(string $host): bool
117 +     {
118 +         if (!filter_var($host, FILTER_VALIDATE_IP)) {
119 +             return false;
120 +         }
121 +
122 +         // Use FILTER_FLAG_NO_PRIV_RANGE and FILTER_FLAG_NO_RES_RANGE to block
      private and reserved IPs
```

```
123 +         return !filter_var($host, FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE
      | FILTER_FLAG_NO_RES_RANGE);
124 +     }
125 +
126 +     private function isInternalDns(string $host): bool
127 +     {
128 +         foreach (self::INTERNAL_DNS_PATTERNS as $pattern) {
129 +             if (str_ends_with($host, $pattern)) {
130 +                 return true;
131 +             }
132 +         }
133 +
134 +         return false;
135 +     }
136 + }
```

src/ValueObject/Exception/InvalidSafeUrl.php

```
... @@ -0,0 +1,11 @@
1  + <?php declare(strict_types=1);
2  +
3  + namespace Movary\ValueObject\Exception;
4  +
5  + class InvalidSafeUrl extends InvalidUrl
6  + {
7  +     public static function create(string $url) : self
8  +     {
9  +         return new self('Not a valid safe url: ' . $url);
10 +     }
11 + }
```

Comments 0

Please sign in to comment.