NationalSecurityAgency / **emissary** Public

Code | Issues 1 | Pull requests 7 | Actions | Projects | Security and quality

Commit `1faf33f`

blueandhack authored last week · 12 / 12 · Verified

```
Merge commit from fork

* fix: sanitize file ending values

Add validation and cleaning for file ending values to prevent shell/meta-character injection.
Introduce INVALID_FILE_ENDING_CHARS and cleanFileEnding() which preserves leading dot and
allows alphanumerics, underscore, hyphen and dot while replacing other chars with '_'. Apply
cleaning to constructor config reads and to setInFileEnding/setOutFileEnding. Extend
ExecutrixTest with unit tests covering cleanFileEnding behavior, setter sanitization, and
configuration-time sanitization.

* chore: deduplicate and strengthen cleanFileEnding tests
```

main · 8.43.0

1 parent 082b263 commit 1faf33f

2 files changed +117 -4 lines changed

Top

Filter files…

- src
  - main/java/emissary/util/shell
    - Executrix.java
  - test/java/emissary/util/shell
    - ExecutrixTest.java

2 files changed +117 -4 lines changed

Search within code

...ain/java/emissary/util/shell/Executrix.java

```
@@ -77,6 +77,12 @@ public enum OUTPUT_TYPE {
77  77
```

```
78  78          protected static final Pattern INVALID_PLACE_NAME_CHARS =
                Pattern.compile("[^a-zA-Z0-9_-]");
79  79
        80  +     /**
        81  +      * Allowlist for file ending values. A leading dot is permitted so that
                normal extensions like {@code .out} are accepted
        82  +      * unchanged; every other character must be alphanumeric, underscore, or
                hyphen.
        83  +      */
        84  +     protected static final Pattern INVALID_FILE_ENDING_CHARS =
                Pattern.compile("[^a-zA-Z0-9_.\\-]");
        85  +
80  86          /**
81  87           * Create using all defaults
82  88           */
        @@ -118,8 +124,8 @@ protected void configure(@Nullable final Configurator
        configGArg) {
118 124             final Configurator configG = (configGArg != null) ? configGArg : new
                ServiceConfigGuide();
119 125
120 126             this.command = configG.findStringEntry("EXEC_COMMAND", "echo
                'YouForGotToSetEXEC_COMMAND' | tee bla.txt");
121     -           this.inFileEnding = configG.findStringEntry("IN_FILE_ENDING", "");
122     -           this.outFileEnding = configG.findStringEntry("OUT_FILE_ENDING",
                this.inFileEnding.isEmpty() ? ".out" : "");
        127 +           this.inFileEnding =
                cleanFileEnding(configG.findStringEntry("IN_FILE_ENDING", ""));
        128 +           this.outFileEnding =
                cleanFileEnding(configG.findStringEntry("OUT_FILE_ENDING",
                this.inFileEnding.isEmpty() ? ".out" : ""));
123 129             this.output = configG.findStringEntry("OUTPUT_TYPE", "STD");
124 130             this.order = configG.findStringEntry("ORDER", "NORMAL");
125 131             this.numArgs = configG.findStringEntry("NUM_ARGS", "");
        @@ -149,6 +155,16 @@ protected static String cleanPlaceName(final String
        placeName) {
149 155             return INVALID_PLACE_NAME_CHARS.matcher(placeName).replaceAll("_");
150 156         }
151 157
        158 +     /**
```

```
                159  +      * Remove invalid characters from a file ending value, see {@link
                       Executrix#INVALID_FILE_ENDING_CHARS}
                160  +      *
                161  +      * @param fileEnding the raw file ending (e.g. {@code .out})
                162  +      * @return a cleaned string with shell-unsafe characters replaced by
                       {@code _}
                163  +      */
                164  +     protected static String cleanFileEnding(final String fileEnding) {
                165  +         return INVALID_FILE_ENDING_CHARS.matcher(fileEnding).replaceAll("_");
                166  +     }
                167  +
 152            168          /**
 153            169           * Creates a set of temp file names (does not do any disk activity)
 154            170           *
                   @@ -1174,7 +1190,7 @@ public String getInFileEnding() {
1174           1190           * @param argInFileEnding Value to assign to this.inFileEnding
1175           1191           */
1176           1192          public void setInFileEnding(final String argInFileEnding) {
1177                  -          this.inFileEnding = argInFileEnding;
                1193  +          this.inFileEnding = cleanFileEnding(argInFileEnding);
1178           1194          }
1179           1195
1180           1196          /**
                   @@ -1192,7 +1208,7 @@ public String getOutFileEnding() {
1192           1208           * @param argOutFileEnding Value to assign to this.outFileEnding
1193           1209           */
1194           1210          public void setOutFileEnding(final String argOutFileEnding) {
1195                  -          this.outFileEnding = argOutFileEnding;
                1211  +          this.outFileEnding = cleanFileEnding(argOutFileEnding);
1196           1212          }
1197           1213
1198           1214          /**
```

…java/emissary/util/shell/ExecutrixTest.java

```
                ...  @@ -1,5 +1,7 @@
 1              1    package emissary.util.shell;
 2              2
                3  + import emissary.config.Configurator;
```

```
     4 + import emissary.config.ServiceConfigGuide;
3    5   import emissary.test.core.junit5.UnitTest;
4    6
5    7   import jakarta.annotation.Nullable;
       ↓ ↑  @@ -603,6 +605,101 @@ void testCleanPlaceName() {
603  605           assertEquals("PLACE-NAME-1", Executrix.cleanPlaceName("PLACE-NAME-1"));
604  606       }
605  607
     608 +     // ------------------------------------------------------------------------
     609 +     // cleanFileEnding – covers INVALID_FILE_ENDING_CHARS and cleanFileEnding()
     610 +     // ------------------------------------------------------------------------
     611 +
     612 +     @Test
     613 +     void testCleanFileEndingAllowedCharsPassThrough() {
     614 +         // Normal extensions must survive unchanged
     615 +         assertEquals(".out", Executrix.cleanFileEnding(".out"));
     616 +         assertEquals(".txt", Executrix.cleanFileEnding(".txt"));
     617 +         assertEquals("", Executrix.cleanFileEnding(""));
     618 +         assertEquals(".out-1_a", Executrix.cleanFileEnding(".out-1_a"));
     619 +         assertEquals("ABC123._-", Executrix.cleanFileEnding("ABC123._-"));
     620 +     }
     621 +
     622 +     @Test
     623 +     void testCleanFileEndingShellMetacharacters() {
     624 +         // Every shell-special character must be individually replaced with '_'
     625 +         assertEquals("_", Executrix.cleanFileEnding("`"));
     626 +         assertEquals("_", Executrix.cleanFileEnding("$"));
     627 +         assertEquals("_", Executrix.cleanFileEnding(";"));
     628 +         assertEquals("_", Executrix.cleanFileEnding("&"));
     629 +         assertEquals("_", Executrix.cleanFileEnding("|"));
     630 +         assertEquals("_", Executrix.cleanFileEnding(">"));
     631 +         assertEquals("_", Executrix.cleanFileEnding("<"));
     632 +         assertEquals("_", Executrix.cleanFileEnding("!"));
     633 +         assertEquals("_", Executrix.cleanFileEnding(" "));
     634 +         assertEquals("_", Executrix.cleanFileEnding("("));
     635 +         assertEquals("_", Executrix.cleanFileEnding(")"));
     636 +         assertEquals("_", Executrix.cleanFileEnding("'"));
     637 +         assertEquals("_", Executrix.cleanFileEnding("\""));
     638 +         assertEquals("_", Executrix.cleanFileEnding("\\"));
```

```
639 +         assertEquals("_", Executrix.cleanFileEnding("/"));
640 +         assertEquals("_", Executrix.cleanFileEnding("\n"));
641 +         assertEquals("_", Executrix.cleanFileEnding("\t"));
642 +     }
643 +
644 +     @Test
645 +     void testCleanFileEndingDoesNotStripHyphenOrUnderscore() {
646 +         // Hyphen and underscore are safe in file extensions and must not be
      replaced
647 +         assertEquals(".tar.gz", Executrix.cleanFileEnding(".tar.gz"));
648 +         assertEquals(".out-v2", Executrix.cleanFileEnding(".out-v2"));
649 +         assertEquals(".out_v2", Executrix.cleanFileEnding(".out_v2"));
650 +     }
651 +
652 +     // ------------------------------------------------------------------------
653 +     // setInFileEnding / setOutFileEnding – sanitization enforced by setters
654 +     // ------------------------------------------------------------------------
655 +
656 +     @Test
657 +     void testSetFileEndingsNormalValuesUnchanged() {
658 +         // Legitimate extensions must not be altered by the setters
659 +         e.setInFileEnding(".in");
660 +         assertEquals(".in", e.getInFileEnding());
661 +
662 +         e.setOutFileEnding(".out");
663 +         assertEquals(".out", e.getOutFileEnding());
664 +     }
665 +
666 +     @Test
667 +     void testSetFileEndingsSanitizeInjectionPayloads() {
668 +         e.setInFileEnding("`id`");
669 +         assertEquals("_id_", e.getInFileEnding());
670 +
671 +         e.setOutFileEnding(";rm -rf /");
672 +         assertEquals("_rm_-rf__", e.getOutFileEnding());
673 +     }
674 +
675 +     // ------------------------------------------------------------------------
676 +     // configure() path – sanitization applied when reading from Configurator
677 +     // ------------------------------------------------------------------------
```

```
678 +
679 +     @Test
680 +     void testConfigureFileEndingsSanitizedFromConfig() {
681 +         // Injection payloads in config must be neutralized at construction
      time
682 +         final Configurator cfg = new ServiceConfigGuide();
683 +         cfg.addEntry("IN_FILE_ENDING", "`id > /tmp/pwned.txt`");
684 +         cfg.addEntry("OUT_FILE_ENDING", "$(whoami)");
685 +
686 +         final Executrix ex = new Executrix(cfg);
687 +         assertEquals("_id____tmp_pwned.txt_", ex.getInFileEnding());
688 +         assertEquals("__whoami_", ex.getOutFileEnding());
689 +     }
690 +
691 +     @Test
692 +     void testConfigureNormalFileEndingsUnchanged() {
693 +         // Normal extensions must pass through configure() without modification
694 +         final Configurator cfg = new ServiceConfigGuide();
695 +         cfg.addEntry("IN_FILE_ENDING", ".xml");
696 +         cfg.addEntry("OUT_FILE_ENDING", ".json");
697 +
698 +         final Executrix ex = new Executrix(cfg);
699 +         assertEquals(".xml", ex.getInFileEnding());
700 +         assertEquals(".json", ex.getOutFileEnding());
701 +     }
702 +
606 703         private static void readAndNuke(final String name) throws IOException {
607 704             final File f = new File(name);
608 705             assertTrue(f.exists(), "File " + name + " must exist");
```

Comments 0

Please sign in to comment.