This directory contains the common infrastructure for the following tests (also referred below as projects).

- referrer-policy/
- mixed-content/
- upgrade-insecure-requests/

Subdirectories:

- `resources` : Serves JavaScript test helpers.
- `subresource` : Serves subresources, with support for redirects, stash, etc. The subresource paths are managed by `subresourceMap` and fetched in `requestVia*()` functions in `resources/common.js` .
- `scope` : Serves nested contexts, such as iframe documents or workers. Used from `invokeFrom*()` functions in `resources/common.js` .
- `tools` : Scripts that generate test HTML files. Not used while running tests.
- `/referrer-policy/generic/subresource-test` : Sanity checking tests for subresource invocation (This is still placed outside common/)

Test generator

The test generator (common/security-features/tools/generate.py) generates test HTML files from templates and a seed (`spec.src.json`) that defines all the test scenarios.

The project (i.e. a WPT subdirectory, for example `referrer-policy/`) that uses the generator should define per-project data and invoke the common generator logic in `common/security-features/tools` .

This is the overview of the project structure:

```
common/security-features/
└── tools/ - the common test generator logic
    ├── spec.src.json
    └── template/ - the test files templates
project-directory/ (e.g. referrer-policy/)
├── spec.src.json
├── generic/
│   ├── test-case.sub.js - Per-project test helper
│   ├── sanity-checker.js (Used by debug target only)
│   └── spec_json.js (Used by debug target only)
└── gen/ - generated tests
```

Generating the tests

Note: When the repository already contains generated tests, remove all generated tests first.

```
# Install json5 module if needed.
pip install --user json5

# Generate the test files under gen/ (HTMLs and .headers files).
path/to/common/security-features/tools/generate.py --spec path/to/project-directory/
```

```
# Add all generated tests to the repo.
git add path/to/project-directory/gen/ && git commit -m "Add generated tests"
```

This will parse the spec JSON5 files and determine which tests to generate (or skip) while using templates.

- The default spec JSON5: `common/security-features/tools/spec.src.json`.
  - Describes common configurations, such as subresource types, source context types, etc.
- The per-project spec JSON5: `project-directory/spec.src.json`.
  - Describes project-specific configurations, particularly those related to test generation patterns (`specification`), policy deliveries (e.g. `delivery_type`, `delivery_value`) and `expectation`.

For how these two spec JSON5 files are merged, see Sub projects section.

Note: `spec.src.json` is transitioning to JSON5 #21710.

During the generation, the spec is validated by `common/security-features/tools/spec_validator.py`. This is specially important when you're making changes to `spec.src.json`. Make sure it's a valid JSON (no comments or trailing commas). The validator reports specific errors (missing keys etc.), if any.

Removing all generated tests

Simply remove all files under `project-directory/gen/`.

```
rm -r path/to/project-directory/gen/
```

Options for generating tests

Note: this section is currently obsolete. Only the release template is working.

The generator script has two targets: `release` and `debug`.

- Using **release** for the target will produce tests using a template for optimizing size and performance. The release template is intended for the official web-platform-tests and possibly other test suites. No sanity checking is done in release mode. Use this option whenever you're checking into web-platform-tests.
- When generating for `debug`, the produced tests will contain more verbosity and sanity checks. Use this target to identify problems with the test suites when making changes locally. Make sure you don't check in tests generated with the debug target.

Note that **release** is the default target when invoking `generate.py`.

Sub projects

Projects can be nested, for example to reuse a single `spec.src.json` across similar but slightly different sets of generated tests. The directory structure would look like:

```
project-directory/ (e.g. referrer-policy/)
├── spec.src.json - Parent project's spec JSON
├── generic/
│   └── test-case.sub.js - Parent project's test helper
├── gen/ - parent project's generated tests
```

```
└── sub-project-directory/ (e.g. 4K)
    ├── spec.src.json - Child project's spec JSON
    ├── generic/
    │   └── test-case.sub.js - Child project's test helper
    └── gen/ - child project's generated tests
```

`generate.py --spec project-directory/sub-project-directory` generates test files under `project-directory/sub-project-directory/gen`, based on `project-directory/spec.src.json` and `project-directory/sub-project-directory/spec.src.json`.

- The child project's `spec.src.json` is merged into parent project's `spec.src.json`.
  - Two spec JSON objects are merged recursively.
  - If a same key exists in both objects, the child's value overwrites the parent's value.
    - If both (child's and parent's) values are arrays, then the child's value is concatenated to the parent's value.
  - For debugging, `generate.py` dumps the merged spec JSON object as `generic/debug-output.spec.src.json`.
- The child project's generated tests include both of the parent and child project's `test-case.sub.js`:

```
<script src="project-directory/test-case.sub.js"></script>
<script src="project-directory/sub-project-directory/test-case.sub.js">
</script>
<script>
  TestCase(...);
</script>
```

Updating the tests

The main test logic lives in `project-directory/generic/test-case.sub.js` with helper functions defined in `/common/security-features/resources/common.js` so you should probably start there.

For updating the test suites you will most likely do **a subset** of the following:

- Add a new subresource type:
  - Add a new sub-resource python script to `/common/security-features/subresource/`.
  - Add a sanity check test for a sub-resource to `referrer-policy/generic/subresource-test/`.
  - Add a new entry to `subresourceMap` in `/common/security-features/resources/common.js`.
  - Add a new entry to `valid_subresource_names` in `/common/security-features/tools/spec_validator.py`.
  - Add a new entry to `subresource_schema` in `spec.src.json`.
  - Update `source_context_schema` to specify in which source context the subresource can be used.
- Add a new subresource redirection type
  - TODO: to be documented. Example: https://github.com/web-platform-tests/wpt/pull/18939
- Add a new subresource origin type

    - TODO: to be documented. Example: https://github.com/web-platform-tests/wpt/pull/18940
- Add a new source context (e.g. "module sharedworker global scope")
    - TODO: to be documented. Example: https://github.com/web-platform-tests/wpt/pull/18904
- Add a new source context list (e.g. "subresource request from a dedicated worker in a `<iframe srcdoc>`")
    - TODO: to be documented.
- Implement new or update existing assertions in `project-directory/generic/test-case.sub.js`.
- Exclude or add some tests by updating `spec.src.json` test expansions.
- Implement a new delivery method.
    - TODO: to be documented. Currently the support for delivery methods are implemented in many places across `common/security-features/`.
- Regenerate the tests and MANIFEST.json

How the generator works

This section describes how `spec.src.json` is turned into scenario data in test HTML files which are then processed by JavaScript test helpers and server-side scripts, and describes the objects/types used in the process.

The spec JSON

`spec.src.json` is the input for the generator that defines what to generate. For examples of spec JSON files, see referrer-policy/spec.src.json or mixed-content/spec.src.json.

Main sections

- **`specification`**

  Top level requirements with description fields and a `test_expansion` rule. This is closely mimicking the Referrer Policy specification structure.

- **`excluded_tests`**

  List of `test_expansion` patterns expanding into selections which get skipped when generating the tests (aka. blocklisting/suppressing)

- **`test_expansion_schema`**

  Provides valid values for each field. Each test expansion can only contain fields and values defined by this schema (or `"*"` values that indicate all the valid values defined this schema).

- **`subresource_schema`**

  Provides metadata of subresources, e.g. supported delivery types for each subresource.

- **`source_context_schema`**

  Provides metadata of each single source context, e.g. supported delivery types and subresources that can be sent from the context.

- **`source_context_list_schema`**

  Provides possible nested combinations of source contexts. See SourceContexts Resolution section below for details.

Test Expansion Pattern Object

Test expansion patterns (`test_expansion` s in `specification` section) define the combinations of test configurations (*selections*) to be generated. Each field in a test expansion can be in one of the following formats:

- Single match: `"value"`
- Match any of: `["value1", "value2", ...]`
- Match all: `"*"`

The following fields have special meaning:

- **`name`** : just ignored. (Previously this was used as a part of filenames but now this is merely a label for human and is never used by generator. This field might be removed in the future (https://github.com/web-platform-tests/wpt/issues/21708))
- **`expansion`** : if there is more than one pattern expanding into a same selection, the pattern appearing later in the spec JSON will overwrite a previously generated selection. To make clear this is intentional, set the value of the `expansion` field to `default` for an expansion appearing earlier and `override` for the one appearing later.

For example a test expansion pattern (taken from referrer-policy/spec.src.json, sorted/formatted for explanation):

```
{
  "name": "insecure-protocol",
  "expansion": "default",

  "delivery_type": "*",
  "delivery_value": "no-referrer-when-downgrade",
  "source_context_list": "*",

  "expectation": "stripped-referrer",
  "origin": ["same-http", "cross-http"],
  "redirection": "*",
  "source_scheme": "http",
  "subresource": "*"
}
```

means: "All combinations with all possible `delivery_type` , `delivery_value` = `no-referrer-when-downgrade` , all possible `source_context_list` , `expectation` = `stripped-referrer` , `origin` = `same-http` or `cross-http` , all possible `redirection` , `source_scheme` = `http` , and all possible `subresource` .

Selection Object

A selection is an object that defines a single test, with keys/values from `test_expansion_schema` .

A single test expansion pattern gets expanded into a list of selections as follows:

- Expand each field's pattern (single, any of, or all) to list of allowed values (defined by the `test_expansion_schema`)
- Permute - Recursively enumerate all selections across all fields

The following field has special meaning:

- **`delivery_key`**: This doesn't exist in test expansion patterns, and instead is taken from `delivery_key` field of the spec JSON and added into selections. (TODO(https://github.com/web-platform-tests/wpt/issues/21708): probably this should be added to test expansion patterns to remove this special handling)

For example, the test expansion in the example above generates selections like the following selection (which eventually generates this test file):

```
{
  "delivery_type": "http-rp",
  "delivery_key": "referrerPolicy",
  "delivery_value": "no-referrer-when-downgrade",
  "source_context_list": "worker-classic",

  "expectation": "stripped-referrer",
  "origin": "same-http",
  "redirection": "no-redirect",
  "source_scheme": "http",
  "subresource": "fetch"
}
```

Excluding Test Expansion Patterns

The `excluded_tests` section have objects with the same format as Test Expansion Patterns that define selections to be excluded.

Taking the spec JSON, the generator follows this algorithm:

- Expand all `excluded_tests` to create a denylist of selections
- For each `specification` entries: Expand the `test_expansion` pattern into selections and check each against the denylist, if not marked as suppresed, generate the test resources for the selection

SourceContext Resolution

The `source_context_list_schema` section of `spec.src.json` defines templates of policy deliveries and source contexts. The `source_context_list` value in a selection specifies the key of the template to be used in `source_context_list_schema`, and the following fields in the selection are filled into the template (these three values define the **target policy delivery** to be tested):

- `delivery_type`
- `delivery_key`
- `delivery_value`

Source Context List Schema

Each entry of **`source_context_list_schema`**, defines a single template of how/what policies to be delivered in what source contexts (See also PolicyDelivery and SourceContext).

- The key: the name of the template which matches with the `source_context_list` value in a selection.
- `sourceContextList`: an array of `SourceContext` objects that represents a (possibly nested) context.
  - `sourceContextType` of the first entry of `sourceContextList` should be always `"top"`, which represents the top-level generated test HTML. This entry is omitted in the scenario JSON object passed to JavaScript runtime, but the policy deliveries specified here are handled by the generator, e.g. written as `<meta>` elements in the generated test HTML.
- `subresourcePolicyDeliveries`: an array of `PolicyDelivery` objects that represents policies specified at subresource requests (e.g. `referrerPolicy` attribute of `<img>` elements).

PolicyDelivery placeholders

Instead to ordinal `PolicyDelivery` objects, the following placeholder strings can be used in `sourceContextList` or `subresourcePolicyDeliveries`.

- `"policy"`:
  - Replaced with the target policy delivery.
- `"policyIfNonNull"`:
  - Replaced with the target policy delivery, only if `delivery_value` is not `null`. If `delivery_value` is `null`, then the test is not generated.
- `"anotherPolicy"`:
  - Replaced with a `PolicyDelivery` object that has a different value from the target policy delivery.
  - Can be used to specify e.g. a policy that should be overridden by the target policy delivery.

`source_context_schema` and `subresource_schema`

These represent supported delivery types and subresources for each source context or subresource type. These are used

- To filter out test files for unsupported combinations of delivery types, source contexts and subresources during SourceContext resolution.
- To determine what delivery types can be used for `anotherPolicy` placeholder.

Example

For example, the following entry in `source_context_list_schema`:

```
"worker-classic": {
  "sourceContextList": [
    {
      "sourceContextType": "top",
      "policyDeliveries": [
        "anotherPolicy"
      ]
    },
    {
      "sourceContextType": "worker-classic",
      "policyDeliveries": [
        "policy"
```

```
      ]
    }
  ],
  "subresourcePolicyDeliveries": []
}
```

Defines a template to be instantiated with `delivery_key`, `delivery_type` and `delivery_value` values defined outside `source_context_list_schema`, which reads:

- A classic `WorkerGlobalScope` is created under the top-level Document, and has a policy defined by `delivery_key`, `delivery_type` and `delivery_value`.
- The top-level Document has a policy different from the policy given to the classic worker (to confirm that the policy of the classic worker, not of the top-level Document, is used).
- The subresource request is sent from the classic `WorkerGlobalScope`, with no additional policies specified at the subresource request.

And when filled with the following values from a selection:

- `delivery_type`: `"http-rp"`
- `delivery_key`: `"referrerPolicy"`
- `delivery_value`: `"no-referrer-when-downgrade"`

This becomes:

```
"worker-classic": {
  "sourceContextList": [
    {
      "sourceContextType": "top",
      "policyDeliveries": [
        {
          "deliveryType": "meta",
          "key": "referrerPolicy",
          "value": "no-referrer"
        }
      ]
    },
    {
      "sourceContextType": "worker-classic",
      "policyDeliveries": [
        {
          "deliveryType": "http-rp",
          "key": "referrerPolicy",
          "value": "no-referrer-when-downgrade"
        }
      ]
    }
  ],
  "subresourcePolicyDeliveries": []
}
```

which means

- The top-level Document has `<meta name="referrer" content="no-referrer">`.
- The classic worker is created with `Referrer-Policy: no-referrer-when-downgrade` HTTP response headers.

Scenario Object

The **scenario** object is the JSON object written to the generated HTML files, and passed to JavaScript test runtime (as an argument of `TestCase`). A scenario object is an selection object, minus the keys used in SourceContext Resolution:

- `source_context_list`
- `delivery_type`
- `delivery_key`
- `delivery_value`

plus the keys instantiated by SourceContext Resolution:

- `source_context_list`, except for the first `"top"` entry.
- `subresource_policy_deliveries`

For example:

```
{
  "source_context_list": [
    {
      "sourceContextType": "worker-classic",
      "policyDeliveries": [
        {
          "deliveryType": "http-rp",
          "key": "referrerPolicy",
          "value": "no-referrer-when-downgrade"
        }
      ]
    }
  ],
  "subresource_policy_deliveries": [],

  "expectation": "stripped-referrer",
  "origin": "same-http",
  "redirection": "no-redirect",
  "source_scheme": "http",
  "subresource": "fetch"
}
```

TopLevelPolicyDelivery Object

The ***TopLevelPolicyDelivery** object is the first `"top"` entry of `SourceContextList` instantiated by SourceContext Resolution, which represents the policy delivery of the top-level HTML Document.

The generator generates `<meta>` elements and `.headers` files of the top-level HTML files from the TopLevelPolicyDelivery object.

This is handled separately by the generator from other parts of selection objects and scenario objects, because the `<meta>` and `.headers` are hard-coded directly to the files in the WPT repository, while policies of subcontexts are generated via server-side `common/security-features/scope` scripts.

TODO(https://github.com/web-platform-tests/wpt/issues/21710): Currently the name `TopLevelPolicyDelivery` doesn't appear in the code.

How the test runtime works

All the information needed at runtime is contained in an scenario object. See the code/comments of the following files.

- `project-directory/generic/test-case.js` defines `TestCase`, the entry point that receives a scenario object. `resources/common.sub.js` does the most of common JavaScript work.
  - Subresource URLs (which point to `subresource/` scripts) are calculated from `origin` and `redirection` values.
  - Initiating fetch requests based on `subresource` and `subresource_policy_deliveries`.
- `scope/` server-side scripts serve non-toplevel contexts, while the top-level Document is generated by the generator. TODO(https://github.com/web-platform-tests/wpt/issues/21709): Merge the logics of `scope/` and the generator.
- `subresource/` server-side scripts serve subresource responses.