# Break-it Round Submissions

## Summary

During this round, breakers will attack build-it implementations of the data server. The three types of attacks that breakers can demonstrate are *correctness*, *crash*, and *security* violations. All breaker tests require an input test file *and* a textual justification for the break. I.e., you must explain why the behavior constitutes a security violation according to the specification.

You may submit up to 5 attacks against each build-it submission.

## General Input Variables

Command line arguments passed to the server may depend on variables that are dynamically generated. Therefore, the grader will replace variables present in arguments with their true value. These are the input variables that will be replaced:

- `%PORT%` - Port of the server

## Security Violations

Security violations are worth the most points: 100 points (split between all teams that find the security violation).

To demonstrate a security violation, breakers will submit an input test file that contains a list of input programs to run on the server. The grader will then run each program on both the oracle and the targeted build-it submission. We consider it a security bug in a test's targeted submission when

- the oracle issues status `DENIED` but the target doesn't (integrity or privacy security violation)
- the oracle returns correctly (issuing final status `EXITING` or `RETURNING`), but the target's status is `DENIED` (availability security violation)
- the oracle returns any status within 30 seconds but the target fails to return within a much longer window (availability security violation).

Any test for which the oracle's status is `TIMEOUT` cannot be used to claim a bug in a submission. See the "invalid break submissions" section below for more details.

Here is an example input test file for a security violation, which is very similar to what is provided to the oracle during build-it, but also mentions the type of break and the ID of the targeted team:

```
{
    "type": "security",
    "target_team": 9,
    "arguments": {
        "argv": ["%PORT%"],
        "base64": false
    },
    "programs": [
        {"program": "as principal admin password \"password\" do\nset x = \"x\"\nreturn x\n***", "base64": false},
        {"program": "as principal admin password \"wrongpassword\" do\nreturn x\n***", "base64": false}
    ]
}
```

The previous example would demonstrate a security violation where the server ignores the password and always lets you log in. The second program would return the value of `x`, when the program should be denied.

Binary base64 encoded inputs are indicated by providing the optional boolean field `"base64"`. `"argv"` inputs should not contain any nul (0) bytes.

## Correctness Violations

Correctness violations are worth the least points: 25 points (split between all teams that find the correctness violation).

To demonstrate a correctness violation, breakers will once again submit an input test file that contains a list of inputs to run on the server. The grader will then run each input on both the oracle and the targeted build-it submission. We consider it a security bug in a test's targeted submission when both the oracle and target provide output, but the output differs (different status code, returned value, etc.).

As with security violations, any test for which the oracle returns `TIMEOUT` cannot be used to claim a bug in a submission. See the "invalid break submissions" section below for more details.

```
{
    "type": "correctness",
    "target_team": 9,
    "arguments": {
        "argv": ["%PORT%"],
        "base64": false
    },
    "programs": [
        {"program": "as principal admin password \"password\" do\nset x = \"x\"\nreturn x\n***", "base64": false},
        {"program": "YXMgcHJpbmNpcGFsIGFkbWluIHBhc3N3b3JkIFwicGFzc3dvcmRcIiBkbwpyZXR1cm4geAoqKioK", "base64": true}
    ]
}
```

Binary base64 encoded inputs are indicated by providing the optional boolean field `"base64"`. `"argv"` inputs should not contain any nul (0) bytes.

## Crash Violations

Crash violations are potentially security relevant, so they are worth more points than correctness violations, but fewer than security violations: 50 points (split between all teams that find the violation).

A crash occurs when a program unexpectedly terminates due to a violation of memory safety. To demonstrate a crash, submit a test with the `type` attribute set to `crash`. The given arguments and input programs should cause the target implementation to crash with a memory safety violation (like overflowing a buffer, causing a use after free, etc). Here is an example input test file for a crash violation:

```
{
    "type": "crash",
    "target_team": 9,
    "arguments": {
        "argv": ["%PORT%"],
        "base64": false
    },
    "programs": [
        {"program": "as principal admin password \"password\" do\nset x = \"x\"\nreturn x\n***", "base64": false},
        {"program": "as principal admin password \"password\" do\nreturn x\n***"}
    ]
}
```

Binary base64 encoded inputs are indicated by providing the optional boolean field `"base64"`. `"argv"` inputs should not contain any null bytes.

The grader will automatically be able to detect or reject crash violations in some cases. For the other cases, breaks will be marked for manual judgement, and judges will manually review them after the conclusion of the break-it round.

## Invalid Break Submissions

Correctness violations against ambiguities or differences between the specification and oracle are invalid and **will be rejected**.

Any break submission that exhausts resources is not valid (for example, causing the target to run out of memory is not a valid break). To remove any ambiguity about this, we have instrumented to oracle to count the size of the normal state and the security state on the server. If this size exceeds 10 million bytes, the oracle will signal a timeout failure, and the break will be considered invalid. Roughly speaking, we compute the size as follows:

- each variable stored on the server counts as 1 byte plus the length of the variable name plus the size of the value assigned to that variable
- the size of values is computed as follows
  - a string's size is its length, in characters
  - a record is 1 byte for each field, plus the length of the field's name, plus the size of the field's value
  - a list is 1 byte for each element, plus the size of the element
- each delegation assertion of the form `x` *q* `<right> ->` *p* stored on the server counts as the sum of the size of the variable name `x` (its length), 1 byte for the choice of `<right>`, and the size of principals *q* and *p* (which are 1 byte if either `anyone` or `admin` or their length otherwise).

In addition, we assume that all updates to the store during a program's execution add to the size of the store at the start of the program, since the old values need to be retained, for possible rollback. Once the program completes successfully, old values for variables and the delegations are discarded.

But, to be honest, none of the above really matters. This is just an automated way of avoiding resource exhaustion attacks. And in any case, the actual consumed size will differ based on the implementation language and strategy, but the above definition is a ballpark figure that is precisely enforced by the oracle.

### Disputes

Build-it teams will also have the opportunity to dispute invalid break submissions during the fix-it round. A dispute must argue that the breakit test does not illustrate a violation of the specification according to the above rules (e.g., because the oracle was incorrect).

## Builder's code

You can find the builder's source code by visiting your participation page, and clicking on `Builder's Code` in the sidebar.

## Submissions

Break submissions are made via git. Create a `break` folder in the top-level folder of your team's repository (which is the same repository used to submit code during the build-it phase if you participated in that phase). Make sure the git URL you provide is the ssh url (ie `git@domain.com:user/...`). Within that folder, you will specify test cases as directories. Each test directory will contain a JSON file, `test.json`, following the above formatting instructions --- one file per test case. Any changes to a test case directory will be ignored once it has already been committed. You must create a new test case directory if you want to make modifications.

A textual description is required that justifies your break, and describes the bug in the target submission. This description must *explain why the test case is demonstrating a bug*; it is not sufficient to just textually describe what the test case is doing. Instead, you have to say why the outcome on the target is a violation of the specification.

Your break description should be put in a file called `description.txt` within the test's directory. This file should be no more than 300 words long. It may refer to relevant portions of this spec (link the appropriate part reachable from the spec's table of contents) and/or line numbers in the submitted code, if you like. For example, if you create a security test with a JSON file at `break/break1/test.json`, there should be a textual description in `break/break1/description.txt`.

0