
RDDR

Riley Wood and Tony Espinoza

Oct 21, 2020

CONTENTS

1	Quick Start	1
2	Architecture	5
2.1	Overview	5
2.2	Running RDDR	6
2.3	Proxies	6
2.4	Support for Different Transport Protocols	7
2.5	Filtering Non-Deterministic Noise	9
2.6	Support for Diffing Various Application Data	9
3	Configuration	15
4	Module Index	19
5	Introduction	21
6	Scope	23
7	Indices and tables	25
	Python Module Index	27
	Index	29

QUICK START

This page will get you up and running with RDDR in no time. We will deploy 3 instances of DVWA running at different security levels behind RDDR sharing a single database microservice.

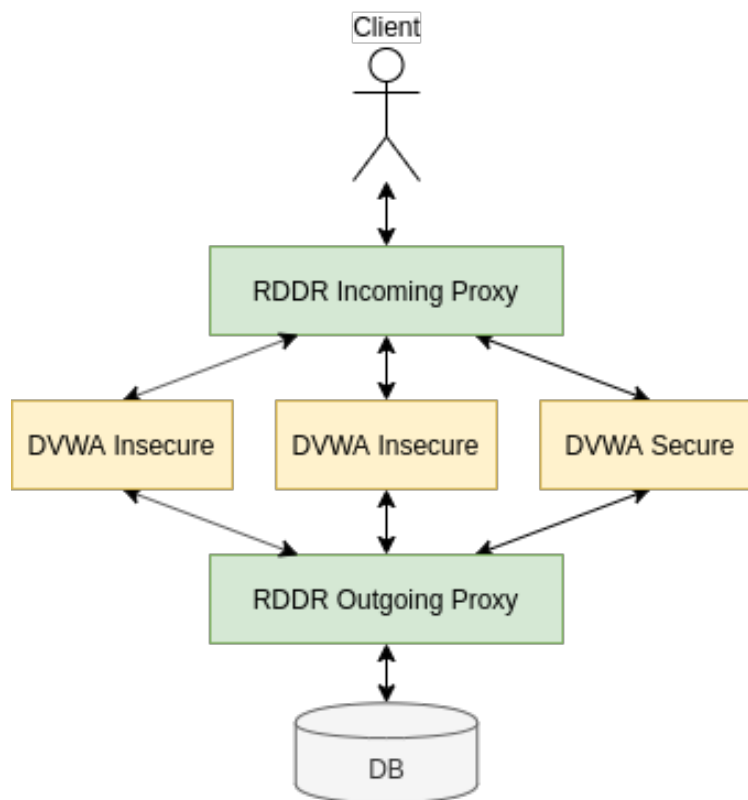


Fig. 1: DVWA deployment with RDDR block diagram

1. [Install Kubernetes on your system.](#) and enable the DNS service. Istio is not supported at this time; we've seen issues with Istio sidecars. Best to disable sidecars for now.

2. Clone the RDDR project repository:

```
git clone https://rjw245@bitbucket.org/rjw245/rddr.git
```

3. Move to the folder rddr/deployments/dvwa_frontend/k8s

```
cd rddr/deployments/dvwa_frontend/k8s
```

4. Apply all kubernetes yaml files in this directory. The command for microk8s is shown below:

```
microk8s.kubectl apply -f .
```

Wait until all pods are in the Running state. MySQL can take some time even after it's in the Running state to become fully ready.

5. Open localhost:31001 in your browser. You should see the login page:

The image shows the DVWA (Damn Vulnerable Web Application) login page. At the top center is the DVWA logo, which consists of the letters 'DVWA' in a bold, dark blue font, with a green and blue swoosh graphic to the right. Below the logo are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Both fields are empty and have a light gray border. Below the password field is a 'Login' button with a gray background and black text.

Fig. 2: DVWA login page

6. Login with user `admin`, password `password`.

7. Click “Create/Reset Database” and login once more.

8. Navigate to the “SQL Injection” tab. `/home/riley/rddr/docs/.venv/bin/pygmentize` 9. Enter a benign input in the User ID box, such as `1`. Press enter and you’ll see the request goes through.

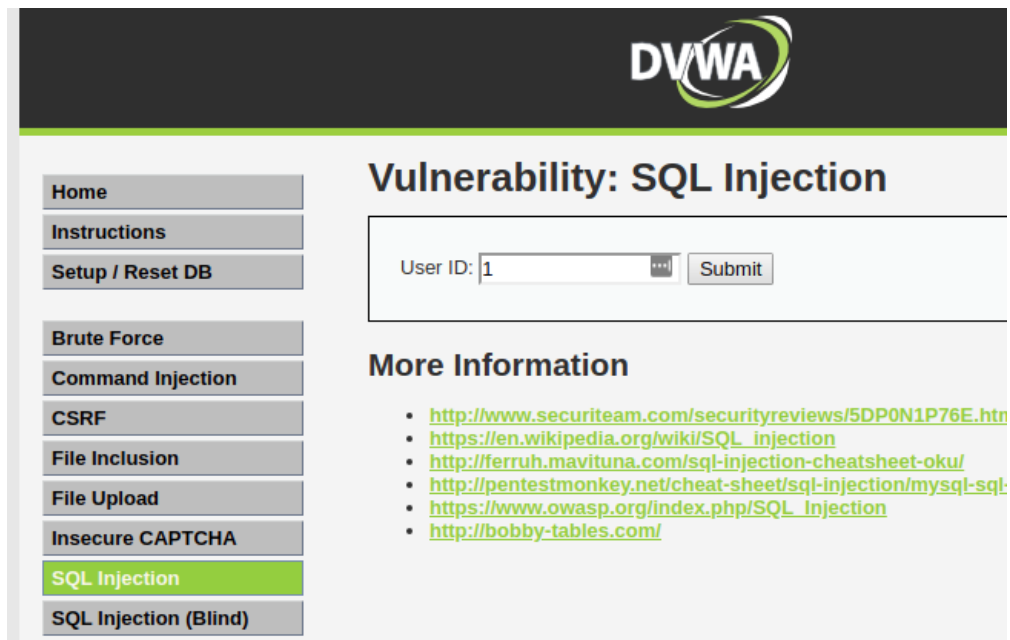
10. Enter a malicious input such as:

```
' UNION SELECT user,password from users WHERE '1'='1
```

and press enter.

11. You should see the request gets denied by RDDR:

This is because we’ve deployed multiple instances of DVWA configured for different security levels. The more secure instance properly sanitizes the SQL query which causes its query to the database to look different from that of the other instances. RDDR catches this divergence in behavior and blocks the response to the user so that the SQL sanitization bug cannot be exploited.



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The top header features the DVWA logo. On the left, a sidebar contains a list of navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (highlighted in green), and SQL Injection (Blind). The main content area is titled "Vulnerability: SQL Injection". Below the title, there is a form with a label "User ID:" followed by a text input field containing the value "1" and a "Submit" button. Below the form, there is a section titled "More Information" containing a list of five links related to SQL injection resources.

DVWA

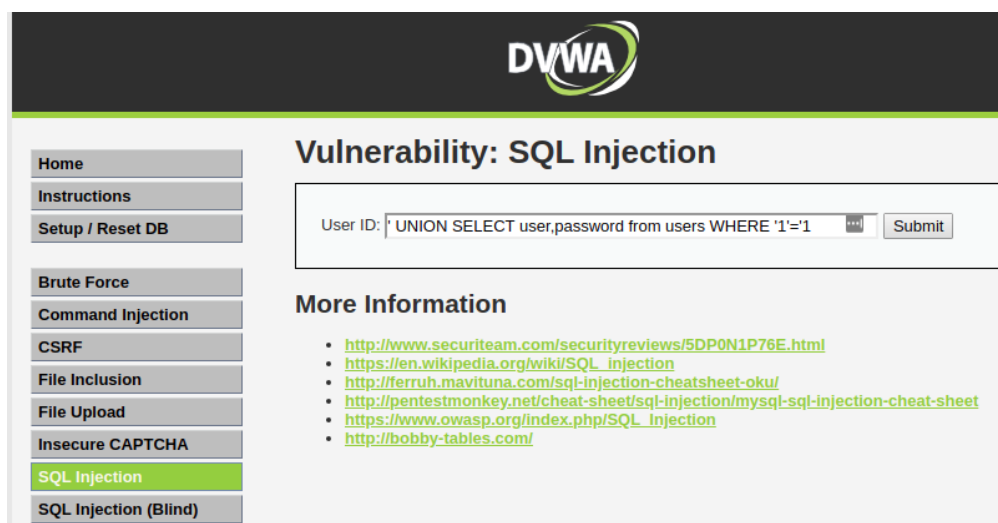
Vulnerability: SQL Injection

User ID:

More Information

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.htm>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- https://www.owasp.org/index.php/SQL_injection
- <http://bobby-tables.com/>

Fig. 3: DVWA SQL Injection with benign input



This screenshot shows the same DVWA interface as Figure 3, but with a malicious SQL injection payload entered into the "User ID" field. The payload is: `' UNION SELECT user,password from users WHERE '1'='1`. The rest of the page, including the sidebar and "More Information" section, remains identical to Figure 3.

DVWA

Vulnerability: SQL Injection

User ID:

More Information

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- https://www.owasp.org/index.php/SQL_injection
- <http://bobby-tables.com/>

Fig. 4: DVWA SQL Injection with malicious input

RDDRЯ

Access Denied

© Riley Wood and Tony Espinoza 2020

Fig. 5: RDDR Denying Access

ARCHITECTURE

Here I will describe how RDDR is structured and link to some of the classes that make it up.

2.1 Overview

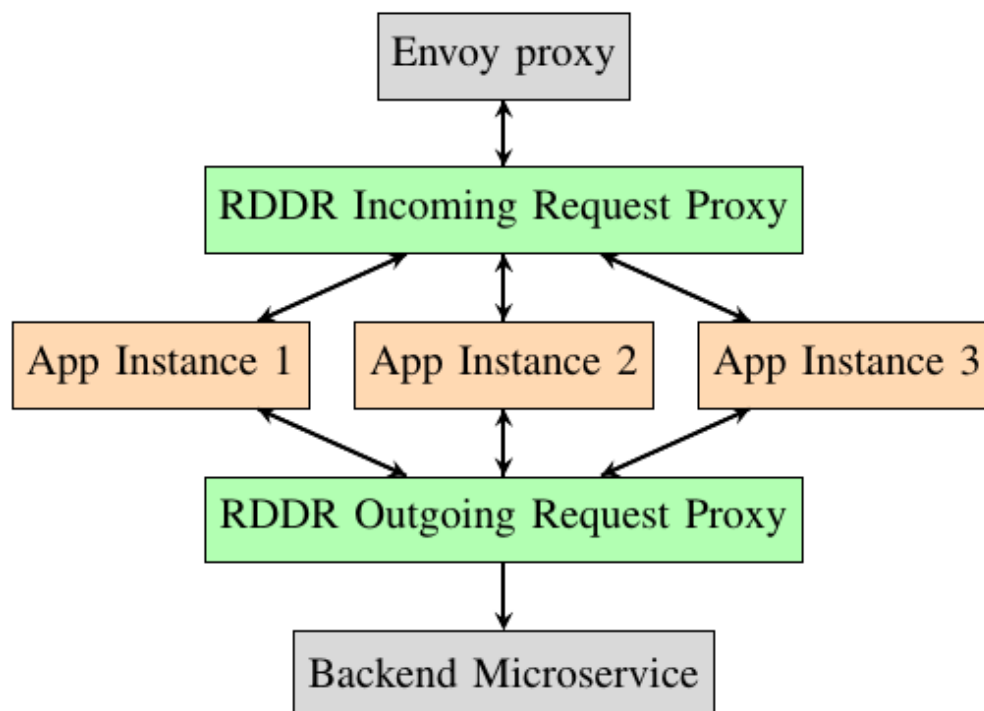


Fig. 1: RDDR Deployment Block Diagram

Above is a simplified view of an RDDR deployment. RDDR sits on either side of a set of instances of the same application. Incoming requests to the proxy are replicated to all application instances, and their responses are diffed and merged before being sent back. Any outgoing requests from the application instances pass through an outgoing proxy which diffes and merges the requests before passing the request on to the target microservice. The response back is then replicated to all app instances. If the application relies on multiple separate backend services, an outgoing request proxy should be created for each. The entire RDDR deployment is placed behind a production-grade Envoy proxy.

RDDR shines when the application instances differ from one another. Subtle variations in the application that do not change nominal behavior can help to catch bugs and prevent exploitation of them.

An ideal deployment will also consist of two copies of the application that are identical. These form the “filter pair” which helps RDDR to distinguish between random noise and real bugs. Because the filter pair are known to be identical, any differences in their behavior will be ignored across the entire set. See [Filtering Non-Deterministic Noise](#) for more information.

2.2 Running RDDR

RDDR is packaged as a Python module with a main method.

```
rddr.__main__.main()
```

Entrypoint for the RDDR proxy. Parses the config file and starts RDDR.

It should be executed like so once installed:

```
python -m rddr [-c path/to/config.yaml]
```

RDDR accepts the following arguments:

```
usage: python -m rddr [-h] [-c CONFIG]

optional arguments:
-h, --help            show this help message and exit
-c CONFIG, --config CONFIG
```

The main method parses the config file and starts the top level RDDR class, shown below:

```
class rddr.rddr.Rddr (config)
```

Bases: object

Top level class for Rddr. Encapsulates one incoming proxy and one or more outgoing proxy.

Parameters **config** (dict) – RDDR configuration dictionary

```
run ()
```

Endless loop. Calls run() on all proxies configured in separate threads.

2.3 Proxies

RDDR implements separate proxies for incoming versus outgoing requests. Both of these proxy classes share the RddrProxy parent class.

```
class rddr.proxies.proxy.RddrProxy (mp_manager, config)
```

Bases: abc.ABC

This is an abstract class, a parent to the incoming and outgoing proxies used by RDDR.

Proxies are built on the asyncio library in Python 3.8. This framework was found to be faster and cleaner than the prior state machine-based implementations.

Parameters **config** (dict) – Dictionary of user config provided to RDDR at the command line

2.3.1 Incoming Proxy

The `RddrIncomingProxy` class implements one proxy between a client and N variants of a server.

```
class rddr.proxies.incoming_proxy.RddrIncomingProxy (mp_manager, config)
    Bases: rddr.proxies.proxy.RddrProxy
```

Implements an incoming proxy for RDDR. Replicates incoming requests to N applications and diffs their responses before forwarding their response.

Parameters `config` (`dict`) – Dictionary of user config provided to RDDR at the command line

```
init_server ()
```

Start an asyncio server for this proxy. Passes the `_new_client` member method as the new client callback.

2.3.2 Outgoing Proxy

The `RddrOutgoingProxy` class implements one proxy between N variants of an application and one server they want to query.

```
class rddr.proxies.outgoing_proxy.RddrOutgoingProxy (mp_manager, config, dest)
    Bases: rddr.proxies.proxy.RddrProxy
```

Implements an outgoing proxy for RDDR. Merges outgoing requests from N applications to some other microservice and replicates the response back.

Parameters

- **config** (`dict`) – Dictionary of user config provided to RDDR at the command line
- **dest** (`str`) – Destination address where incoming requests will be forwarded. String format: “<HOST>:<PORT>”

```
init_server ()
```

Start an asyncio server for this proxy. Passes the `_new_client` member method as the new client callback.

2.4 Support for Different Transport Protocols

We intend RDDR to support a variety of transport protocols. The latest version of RDDR supports:

- Unencrypted TCP
- SSL

Each proxy (incoming and outgoing) can be configured for a different transport protocol. This is useful in the cloud, where applications can be stitched from many smaller microservices that all speak different protocols. See [Configuration](#) for more on configuring the protocol.

```
class rddr.protocols.protocol.RddrProtocol
    Bases: abc.ABC
```

Abstract class to be used as a base for concrete protocols.

```
abstract create_server (host, port)
```

Coroutine. Creates a server socket.

Parameters

- **host** (`str`) – Hostname/IP to bind server to
- **port** (`int`) – Port to bind server to

Return type `None`

get_stream_addr (*stream*)

Given an asyncio stream (either `StreamReader` or `StreamWriter`) returns a string representing the host and port of the party on the other end of the streams. Format: “<host>:<port>” May return `None` if address cannot be retrieved.

Parameters **stream** (`Union[StreamReader, StreamWriter]`) – `StreamReader` or `StreamWriter` used to communicate with remote party

Return type `Optional[str]`

abstract async open_connection (*host, port*)

Abstract coroutine. Opens a connection to `host:port`. Returns (`StreamReader`, `StreamWriter`)

Parameters

- **host** (`str`) – Host to connect to
- **port** (`int`) – Port to connect to

Return type `Tuple[StreamReader, StreamWriter]`

2.4.1 TCP

class `rddr.protocols.protocol_tcp.RddrProtocolTcp`

Bases: `rddr.protocols.protocol.RddrProtocol`

Support for unencrypted TCP.

create_server (*host, port*)

Coroutine. Creates a server socket.

Parameters

- **host** (`str`) – Hostname/IP to bind server to
- **port** (`int`) – Port to bind server to

Return type `None`

async open_connection (*host, port*)

Creates a new unencrypted socket for forwarding data. Returns (`StreamReader`, `StreamWriter`).

Parameters

- **host** (`str`) – Host to connect to
- **port** (`int`) – Port to connect to

Return type `Tuple[StreamReader, StreamWriter]`

2.4.2 SSL

```
class rddr.protocols.protocol_ssl.RddrProtocolSsl (cert='certs/clientcert.pem',
                                                    key='certs/clientkey.pem')
```

Bases: `rddr.protocols.protocol.RddrProtocol`

Support for SSL on top of TCP

Parameters

- **cert** (`str`) – Path to the certificate file
- **key** (`str`) – Path to the key file

```
create_server (host, port)
```

Coroutine. Creates a server socket with SSL context.

Parameters

- **host** (`str`) – Hostname/IP to bind server to
- **port** (`int`) – Port to bind server to

Return type `None`

```
async open_connection (host, port)
```

Creates a new socket and wraps it in an encrypted session for forwarding data. Returns (`StreamReader`, `StreamWriter`).

Parameters

- **host** (`str`) – Host to connect to
- **port** (`int`) – Port to connect to

Return type `Tuple[StreamReader, StreamWriter]`

2.5 Filtering Non-Deterministic Noise

You can deploy two identical app instances which will help to filter any non-deterministic noise in the system. See the `filter` parameter in [Configuration](#).

Consider a query that fetches a random number from the application. Every instance will generate a different random number. Without filtering, the incoming proxy would flag this divergence as a potential bug. However, if the proxy sees that the two identical apps also differ in their responses, we can safely ignore this region of the response and in doing so we've filtered out the non-deterministic noise.

2.6 Support for Diffing Various Application Data

Filtering non-deterministic noise as described above requires the proxies to do a minimal amount of parsing of the data. For example, if the data is in JSON format, we may want ignore certain non-deterministic keys of the data structure. For a text file, we may instead want to ignore certain lines. We need a way to tokenize the data being transferred so that we can ignore particular tokens. Since the tokenizing algorithm is likely to vary across applications, we have defined a simple interface for others to extend.

2.6.1 Interface Specification

Simply implement a class that inherits from `AbstractRddrDiff` and implement the functions `diff_traffic`, `modify_traffic`, and optionally `render_denial` and `validate_params`:

```
class rddr.AbstractRddrDiff (mp_manager, shared_state, do_filter=False, logger=None,  
                             params=None)
```

Bases: `abc.ABC`

Defines the interface for all RDDR diff plugins. Users may extend this class to add support for a particular protocol to RDDR. Diff plugins may optionally specify configuration parameters that a user may provide through the config YAML file. The `diff-params` key of the YAML file is reserved on each proxy for use by the diff plugin applied to that proxy. The schema expected by the plugin should be well-specified. Diff plugins should implement `validate_params` to validate the schema of the user-provided `diff-params`.

Parameters

- **do_filter** (bool) – If True, will use the first two traffic streams as a filter pair to filter out non-deterministic noise.
- **logger** (Optional[Logger]) – The logger instance to use for printing messages.
- **params** (Optional[dict]) – Miscellaneous user-provided config for the plugin, from the user's YAML config file. Subclasses should define clearly what they expect to be passed as parameters.

`diff_traffic` (*traffic*)

Diff's the traffic from N instances. Also indicates how many bytes of each traffic stream has been processed and whether or not more bytes of the stream are needed to process it. This default implementation will never detect divergence, always processes the entire stream and never requests more bytes. Subclasses may raise the `RddrInsufficientData` exception if `diff_traffic` was called on partial data (i.e. more data is required from the instances to make a decision). The proxy tunnel will handle this exception by reading from the instances once more before calling `diff_traffic` again.

Parameters `traffic` (List[bytes]) – List of bytestrings from N instances.

Return type List[Tuple[int, bool]]

Returns A list of 2-tuples, one tuple for each traffic stream provided through the “traffic” argument. Each tuple is of the form (int, bool). The first element of the tuple is the number of bytes of that stream that have been differenced and can safely be sent along to the client. If this value is zero, no bytes have yet been parsed. If this value is less than zero, then the streams differ from one another, and the traffic SHOULD NOT be forward to the client. The second element of the tuple is a flag indicating whether or not more bytes are required from the traffic source in order to parse this stream. This is useful if the plugin tokenizes the streams and has to this point received a partial token and requires more bytes to fully difference everything.

`modify_traffic` (*traffic*, *n_instances*)

This function replicates one incoming stream into N for each of the N application variants. In the process, it may make modifications to the replica for each instance as necessary. This can be necessary if there are unique tokens that need to be substituted for each instance, as in the case of CSRF tokens in HTML forms. This default implementation makes no modifications to the traffic.

Parameters

- **traffic** (bytes) – Request to modify per recipient in addrlist.
- **n_instances** (int) – Number of app instances in this deployment

Return type List[bytes]

Returns List of the traffic to send to each of the app variants.

render_denial()

The diff interface can implement a custom error message appropriate for the application layer protocol being handled. An error message, for example. Default implementation returns empty byte string.

Return type bytes

Returns Bytestring to be sent back to the client if divergent behavior is seen.

validate_params()

Validates the `diff-params` key in the user config file. By default, does nothing.

We have packaged four classes which implement the interface for JSON, HTTP, raw bytes, and Postgres respectively. These are shown below:

2.6.2 JSON

```
class rddr_diff_builtins.RddrJsonDiff(mp_manager, shared_state, do_filter=False, logger=None, params=None)
```

Bases: `rddr.diff_interface.AbstractRddrDiff`

Diff tool for JSON documents that ships with RDDR. JSON is expected to be embedded in an HTTP response. Differences key by key. Does not modify incoming traffic.

Parameters

- **do_filter** (bool) –
- **logger** (Optional[Logger]) –
- **params** (Optional[dict]) –

diff_traffic(traffic)

Parses JSON documents embedded in HTTP responses. May request more bytes of a given stream if a partial JSON document has been received and cannot yet be parsed. Differences key by key.

See interface definition `rddr.AbstractRddrDiff.diff_traffic()` for more.

Parameters traffic (List[bytes]) – List of traffic from app instances.

Return type List[Tuple[int, bool]]

render_denial()

Returns an HTTP response string containing a 500 error and an “access denied” message, with the RDDR logo. See `static/denied.html` for the content.

Return type bytes

2.6.3 HTTP

```
class rddr_diff_builtins.RddrHttpDiff(mp_manager, shared_state, do_filter=False, logger=None, params=None)
```

Bases: `rddr.diff_interface.AbstractRddrDiff`

Diff tool for HTTP that ships with RDDR. Capable of handling CSRF tokens. N instances may generate form tokens or other per-instance tokens. Plugin will save these tokens and send one along to the client. Upon seeing the client’s token later, will substitute the token appropriate for each server.

diff_traffic(traffic)

Diff’s HTML delimited by line breaks.

Upon encountering noise within a line (i.e. the filter pair differ), will extract the largest contiguous set of characters within the line that differ and save the value reported by each server. These tokens can be reinserted in a user's subsequent requests on sight. The reinsertion is implemented by `modify_traffic`. This is necessary when an application being N-versioned uses anti-CSRF tokens in its user input forms. The proxy must send the appropriate token back to each instance of the application for it to service the user's request.

See interface definition `rddr.AbstractRddrDiff.diff_traffic()` for more.

Parameters `traffic` (List[bytes]) – List of traffic from app instances.

Return type List[Tuple[int, bool]]

modify_traffic (`traffic`, `n_instances`)

Return a list of bytestrings, one to send to each application instance.

This method will re-insert any saved tokens it finds in the user's traffic with the token originally sent by each instance. See `diff_traffic` for further explanation of the utility of this feature.

Parameters

- **traffic** (bytes) – Request to modify per recipient in addrlist.
- **n_instances** (int) – Number of app instances in this deployment

Return type List[bytes]

render_denial ()

Returns an HTTP response string containing a 500 error and an “access denied” message, with the RDDR logo. See `static/denied.html` for the content.

Return type bytes

validate_params ()

Validates the `diff-params` config field for this particular class.

2.6.4 Bytewise

class `rddr_diff_builtins.RddrByteDiff` (`mp_manager`, `shared_state`, `do_filter=False`, `logger=None`, `params=None`)

Bases: `rddr.diff_interface.AbstractRddrDiff`

Parameters

- **do_filter** (bool) –
- **logger** (Optional[Logger]) –
- **params** (Optional[dict]) –

diff_traffic (`traffic`)

Validates that messages match byte for byte.

See interface definition `rddr.AbstractRddrDiff.diff_traffic()` for more.

Parameters `traffic` (List[bytes]) – List of traffic from app instances. Key = instance address “host:port” Value = Bytes response

Return type List[Tuple[int, bool]]

2.6.5 PostgreSQL

```
class rddr_diff_builtins.RddrPostgresDiff(mp_manager, shared_state, do_filter=False,  
                                           logger=None, params=None)
```

Bases: `rddr.diff_interface.AbstractRddrDiff`

This class enables support for diffing Postgres traffic across N application instances. This diff plugin supports `diff-params`. `diff-params` should be a dictionary with one key: `tokens`. `tokens` is a list of lists of bytestrings, one bytestring per application instance. This allows you to preconfigure tokens you expect to be different among the Postgres instances. An example is the string reported for the server version – different variants will provide different strings. By specifying that here, you can avoid flagging that as divergent behavior.

Parameters

- **do_filter** (bool) –
- **logger** (Optional[Logger]) –
- **params** (Optional[dict]) –

diff_traffic (*traffic*)

Validates that Postgres messages match. Ignores certain packet types. See member `_backend_pkt_types_to_ignore` for the full list of ignored packet types. Prior to diffing, will substitute tokens preconfigured in the config file under the `diff-params` key for the associated proxy.

See interface definition `rddr.AbstractRddrDiff.diff_traffic()` for more.

Parameters **traffic** (List[bytes]) – List of traffic from app instances.

Return type List[Tuple[int, bool]]

render_denial ()

The diff interface can implement a custom error message appropriate for the application layer protocol being handled. An error message, for example. Default implementation returns empty byte string.

Return type bytes

Returns Bytestring to be sent back to the client if divergent behavior is seen.

validate_params ()

Validates the `diff-params` config field for this particular class.

These classes each implement the functions `diff_traffic` and `modify_traffic`.

Users may write their own classes that implement the above functions to tailor RDDR's filtering engine for their own application. Simply include the name of your class via the `diff-class` field of your config file.

CONFIGURATION

Here is an example config file:

```
addrlist:
  - 172.17.0.2:443
  - 172.17.0.3:443
  - 172.17.0.4:443
filter: True
incoming-proxy:
  host: 0.0.0.0
  port: 4443
  protocol: ssl
  diff-class: "rddr_diff_builtins.RddrHttpDiff"
  enforcing: True
outgoing-proxies:
  1.2.3.4:3036:
    host: 0.0.0.0
    ports:
      - 3001
      - 3002
      - 3003
    diff-class: "rddr_diff_builtins.RddrJsonDiff"
    protocol: ssl
    enforcing: True
  some-hostname:5432:
    host: 192.168.99.1
    ports:
      - 4001
      - 4002
      - 4003
    diff-class: "rddr_diff_builtins.RddrPostgresDiff"
    diff-params: {
      ...
    }
    protocol: tcp
    enforcing: True
verbosity: "DEBUG"
```

And here is what each of these options configures:

addrlist Addresses (host:port) of the N application variants we want to proxy. If you are doing filtering (i.e. filter is True), make sure the addresses of the two instances comprising your filter pair appear **first in this list**.

filter Specify whether or not to filter non-deterministic noise. This requires you to deploy two identical application instances. This is known as the filter pair. **You must list the filter pair instances as the**

first two entries in addrlist.

incoming-proxy Configuration of the single proxy for incoming requests to the N variants

host The hostname or IP **which the proxy will bind to**. 0.0.0.0 indicates it will bind to all available hostnames/IPs.

port The port **which the proxy will bind to**.

protocol The transport protocol to use for this proxy. One of `tcp`, `ssl`.

diff-class Python class to import for filtering non-deterministic noise. This class must implement the functions `diff_traffic` and `modify_traffic`. RDDR ships with four such classes: `rddr_diff_builtins.RddrHttpDiff`, `rddr_diff_builtins.RddrJsonDiff`, `rddr_diff_builtins.RddrByteDiff`, and `rddr_diff_builtins.RddrPostgresDiff`,

enforcing When the N variants' responses to a request differ (after filtering non-deterministic noise), `enforcing` determines whether or not the server will forward the merged response to the client. `enforcing = True` will block the response.

outgoing-proxies Configure one outgoing proxy per backend service that the N variants make requests to.

<dest 1> This proxy will forward traffic to this destination. Specify the destination as in `addrlist: host:port`

host The hostname or IP **which the proxy will bind to**. 0.0.0.0 indicates it will bind to all available hostnames/IPs.

ports The list of ports **which the proxy will bind to**. Each port should accept connections from one of the N instances. The first port in the list corresponds to the first instance in `addrlist`, the second port with the second instance, and so on. Update the N variants to send their requests to the bound host and the port associated with them. Proxy will bind to `<host>:<port_1>`, ..., `<host>:<port_N>`.

protocol The transport protocol to use for this proxy. One of `tcp`, `ssl`.

diff-class Python class to import for filtering non-deterministic noise.

diff-params Some RDDR diff plugins accept a set of configuration parameters. The plugin will specify its own schema for this field.

enforcing If the N variants differ in their outgoing requests (after filtering non-deterministic noise), `enforcing` determines whether or not the server will forward the merged response to the client. `enforcing = True` will block the request.

...

verbosity Verbosity level. One of `INFO`, `DEBUG`, `WARNING`, `ERROR`.

This RDDR deployment can be visualized as:

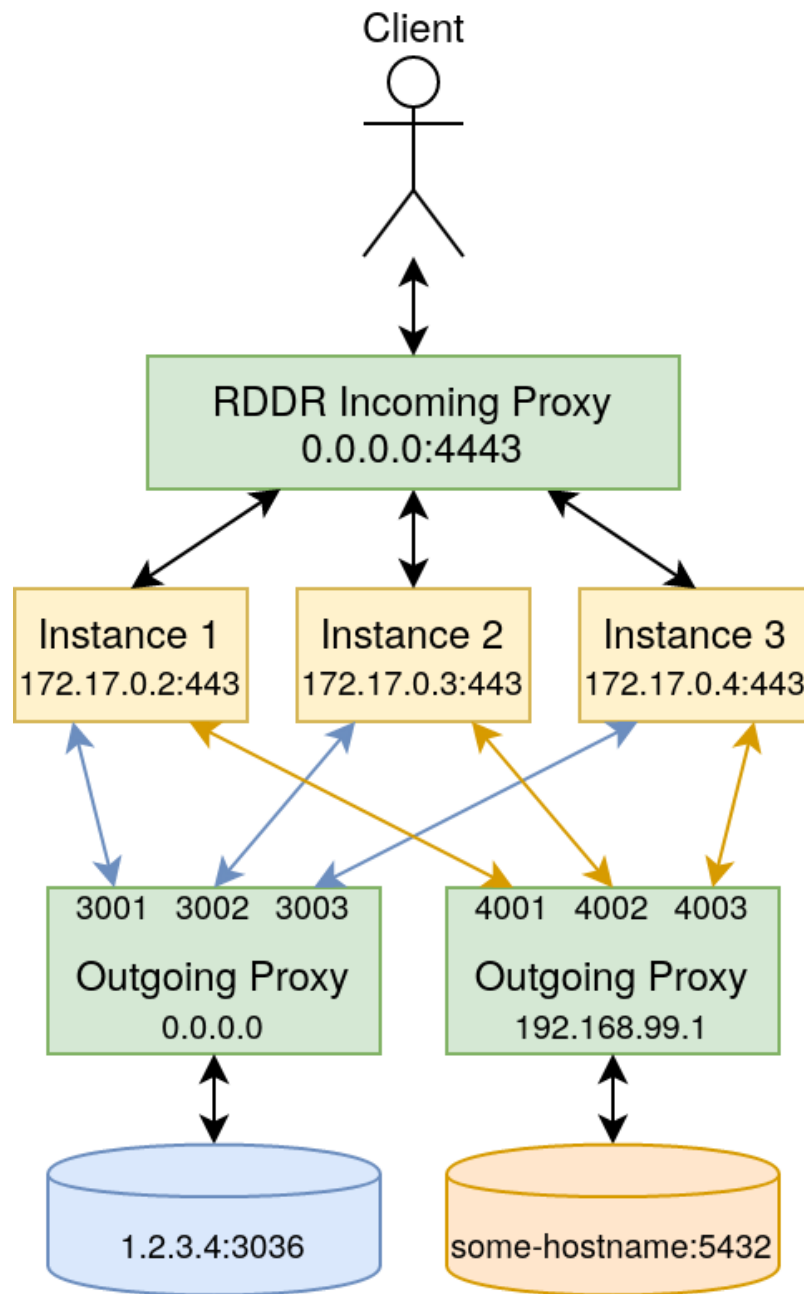


Fig. 1: Block diagram of the above configuration

MODULE INDEX

INTRODUCTION

RDDR is an N-Versioning proxy that can identify bugs in your cloud microservice and prevent them from being exploited.

RDDR sits in front of N variants of any microservice and proxies their incoming and outgoing traffic. It replicates incoming traffic to each instance and then diffs their responses; it also diffs their outgoing requests and replicates the response to each of them. If any difference in behavior is seen among the variants, RDDR will block the response.

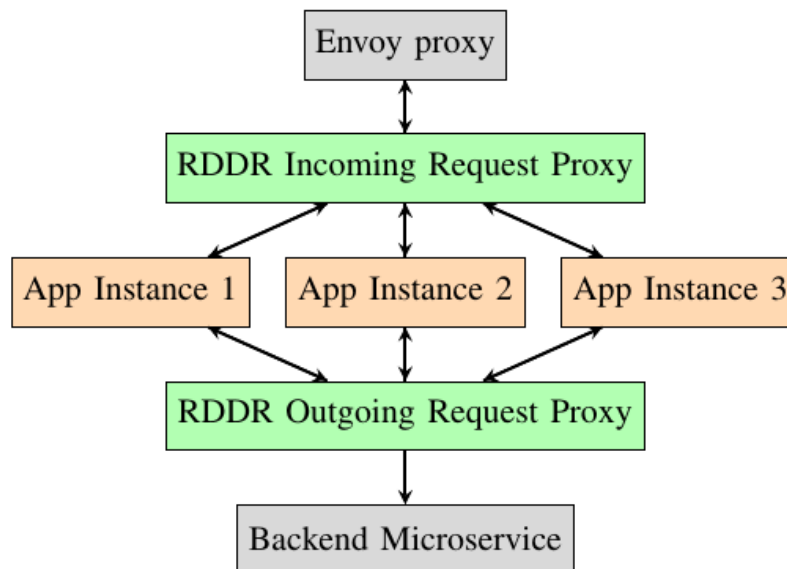


Fig. 1: RDDR Deployment Block Diagram

RDDR stands for:

- Replicate** Incoming requests are replicated to each variant of the microservice.
- De-noise** RDDR filters random noise from their responses.
- Diff** RDDR diffs the remainder of their responses to detect difference in behavior.
- Respond** If no differences are detected, the merged response is returned to the requester.

Read [Quick Start](#) to get started using RDDR.

RDDR was developed at the [Spark Research Lab](#) at the University of Texas at Austin.

SCOPE

This documentation will cover the organization of the RDDR Python module. If time permits, I will also document the Kubernetes deployment I have used for testing. Kubernetes is not required. You can deploy RDDR outside of a container and point it at your replicated instances however they are deployed.

INDICES AND TABLES

- `genindex`
- *Module Index*
- `search`

PYTHON MODULE INDEX

r

rddr.__main__, 6

A

`AbstractRddrDiff` (class in `rddr`), 10

C

`create_server()` (`rddr.protocols.protocol.RddrProtocol` method), 7
`create_server()` (`rddr.protocols.protocol_ssl.RddrProtocolSsl` method), 9
`create_server()` (`rddr.protocols.protocol_tcp.RddrProtocolTcp` method), 8

D

`diff_traffic()` (`rddr.AbstractRddrDiff` method), 10
`diff_traffic()` (`rddr_diff_builtins.RddrByteDiff` method), 12
`diff_traffic()` (`rddr_diff_builtins.RddrHttpDiff` method), 11
`diff_traffic()` (`rddr_diff_builtins.RddrJsonDiff` method), 11
`diff_traffic()` (`rddr_diff_builtins.RddrPostgresDiff` method), 13

G

`get_stream_addr()`
(`rddr.protocols.protocol.RddrProtocol` method), 8

I

`init_server()` (`rddr.proxies.incoming_proxy.RddrIncomingProxy` method), 7
`init_server()` (`rddr.proxies.outgoing_proxy.RddrOutgoingProxy` method), 7

M

`main()` (in module `rddr.__main__`), 6
`modify_traffic()` (`rddr.AbstractRddrDiff` method), 10
`modify_traffic()` (`rddr_diff_builtins.RddrHttpDiff` method), 12

O

`open_connection()`
(`rddr.protocols.protocol.RddrProtocol` method), 8
`open_connection()`
(`rddr.protocols.protocol_ssl.RddrProtocolSsl` method), 9
`open_connection()`
(`rddr.protocols.protocol_tcp.RddrProtocolTcp` method), 8

R

`Rddr` (class in `rddr.rddr`), 6
`rddr.__main__` (module), 6
`RddrByteDiff` (class in `rddr_diff_builtins`), 12
`RddrHttpDiff` (class in `rddr_diff_builtins`), 11
`RddrIncomingProxy` (class in `rddr.proxies.incoming_proxy`), 7
`RddrJsonDiff` (class in `rddr_diff_builtins`), 11
`RddrOutgoingProxy` (class in `rddr.proxies.outgoing_proxy`), 7
`RddrPostgresDiff` (class in `rddr_diff_builtins`), 13
`RddrProtocol` (class in `rddr.protocols.protocol`), 7
`RddrProtocolSsl` (class in `rddr.protocols.protocol_ssl`), 9
`RddrProtocolTcp` (class in `rddr.protocols.protocol_tcp`), 8
`RddrProxy` (class in `rddr.proxies.proxy`), 6
`render_denial()` (`rddr.AbstractRddrDiff` method), 11
`render_denial()` (`rddr_diff_builtins.RddrHttpDiff` method), 12
`render_denial()` (`rddr_diff_builtins.RddrJsonDiff` method), 11
`render_denial()` (`rddr_diff_builtins.RddrPostgresDiff` method), 13
`run()` (`rddr.rddr.Rddr` method), 6

V

`validate_params()` (`rddr.AbstractRddrDiff` method), 11

```
validate_params()  
    (rddr_diff_builtins.RddrHttpDiff    method),  
12  
validate_params()  
    (rddr_diff_builtins.RddrPostgresDiff method),  
13
```