

Sanjiv Kawa
October 16th 2012

Red Hat: Community Web Crawler Version 2.0

Technical Documentation

Table of Contents

1. Downloading and Running	Page 2
2. Expected Output	Pages 3-4
3. Special Conditions	Page 4
4. File Storage	Page 5
5. Major Upgrades	Pages 6-15
5.1 HTMLParser	Page 6
5.2 ScoringAlgorithm	Page 7
5.3 Merging	Page 7
5.4 PullRSS	Page 8
5.5 Database	Pages 9-15
6. New Classes and Functions	Pages 16-20
6.1 HTMLParser	Page 16
6.2 ScoringAlgorithm	Pages 16-17
6.3 PullRSS	Pages 17-18
6.4 HTMLParserAndSecurityController	Page 19
6.5 InsertLastCrawlerRun	Page 19
6.6 MakeDownloadDirs	Page 19
6.7 GetLastCrawlerRunTime	Page 20
6.8 sqliteJdbcForHTMLandJSON	Page 20
7. Crawler Source Code Modifications	Pages 21-22
8. Additional Files	Page 23

1. Downloading and Running

To download and run the communityWebCrawler you must clone into the git repository.

```
git clone git://git.fedorahosted.org/communityWebcrawler.git
```

A directory will be created, titled “communityWebcrawler”

```
cd communityWebcrawler
```

There are **two** branches in this git repository

```
git branch
```

1. master is the default branch where the current working code resides.
2. version1 is the original communityWebCrawler without the additional functionality

To run the crawler:

```
ant run  
  
or  
  
ant CrawlerController
```

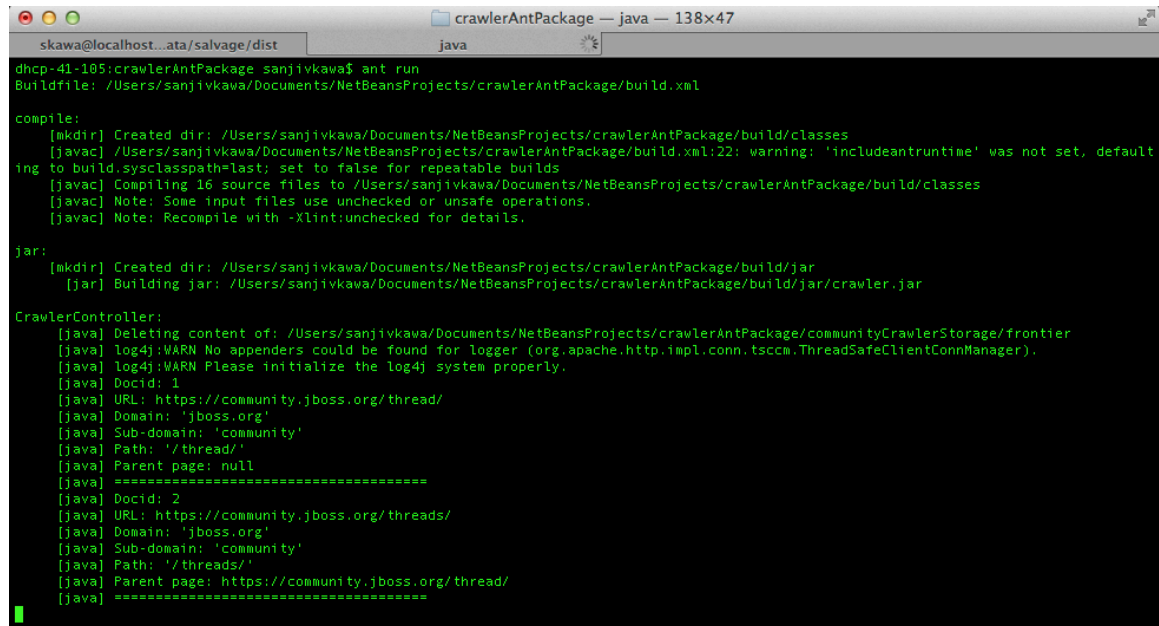
To run the PullRSS class:

```
ant PullRSS
```

Note: PullRSS requires the CrawlerController to have been run first. (Check Section 3 Special Conditions for details).

2. Expected Output

After the crawler has been run, the expected output to console should look relatively similar to this:

A terminal window titled 'crawlerAntPackage — java — 138x47' showing the output of the 'ant run' command. The output includes compilation steps, jar building, and the execution of the 'CrawlerController' class, which logs various details about crawling a URL.

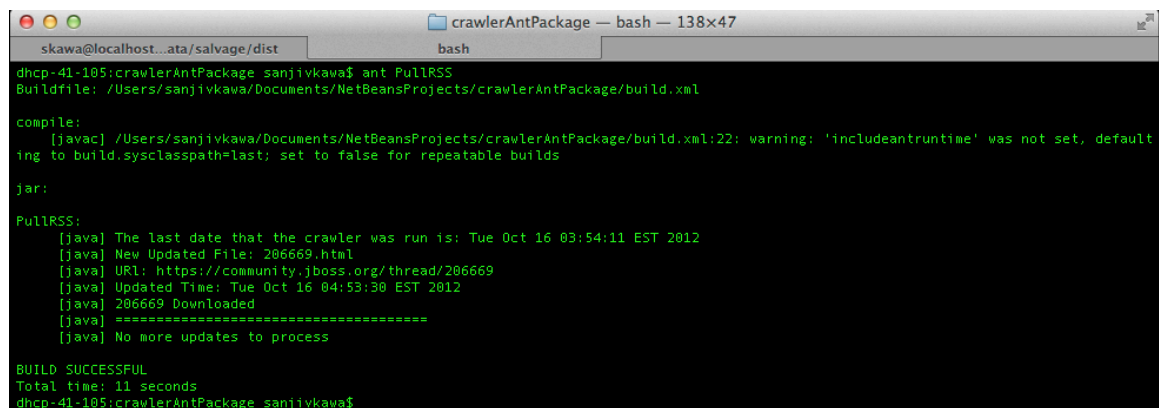
```
skawa@localhost...ata/salvage/dist java
dhcp-41-105:crawlerAntPackage sanjivkava$ ant run
Buildfile: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build.xml

compile:
[mkdir] Created dir: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build/classes
[javac] /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build.xml:22: warning: 'includeantruntime' was not set, default
ing to build.sysclasspath=last; set to false for repeatable builds
[javac] Compiling 16 source files to /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build/classes
[javac] Note: Some input files use unchecked or unsafe operations.
[javac] Note: Recompile with -Xlint:unchecked for details.

jar:
[mkdir] Created dir: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build/jar
[jar] Building jar: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build/jar/crawler.jar

CrawlerController:
[java] Deleting content of: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/communityCrawlerStorage/frontier
[java] log4j:WARN No appenders could be found for logger (org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager).
[java] log4j:WARN Please initialize the log4j system properly.
[java] DocId: 1
[java] URL: https://community.jboss.org/thread/
[java] Domain: 'jboss.org'
[java] Sub-domain: 'community'
[java] Path: '/thread/'
[java] Parent page: null
[java] =====
[java] DocId: 2
[java] URL: https://community.jboss.org/threads/
[java] Domain: 'jboss.org'
[java] Sub-domain: 'community'
[java] Path: '/threads/'
[java] Parent page: https://community.jboss.org/thread/
[java] =====
```

After the PullRSS class has been run, the expected output to console should look relatively similar to this:

A terminal window titled 'crawlerAntPackage — bash — 138x47' showing the output of the 'ant PullRSS' command. The output includes compilation steps, jar building, and the execution of the 'PullRSS' class, which logs details about the last run date, new updated files, and download status.

```
skawa@localhost...ata/salvage/dist bash
dhcp-41-105:crawlerAntPackage sanjivkava$ ant PullRSS
Buildfile: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build.xml

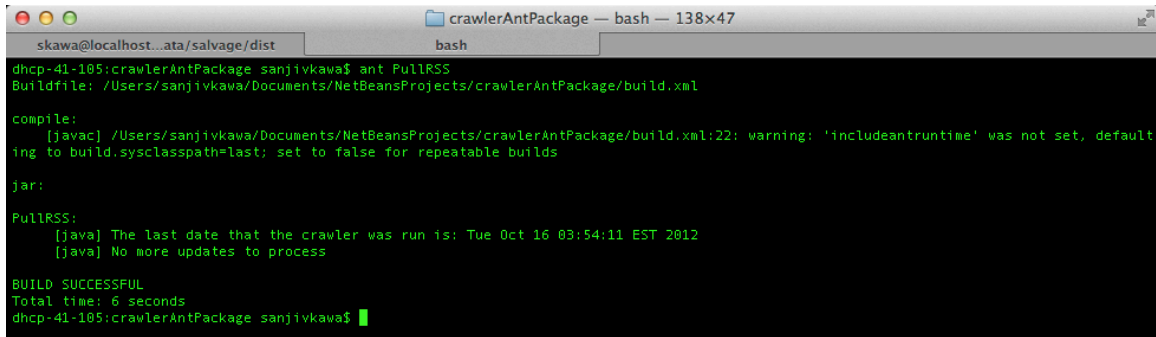
compile:
[javac] /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build.xml:22: warning: 'includeantruntime' was not set, default
ing to build.sysclasspath=last; set to false for repeatable builds

jar:

PullRSS:
[java] The last date that the crawler was run is: Tue Oct 16 03:54:11 EST 2012
[java] New Updated File: 206669.html
[java] URL: https://community.jboss.org/thread/206669
[java] Updated Time: Tue Oct 16 04:53:30 EST 2012
[java] 206669 Downloaded
[java] =====
[java] No more updates to process

BUILD SUCCESSFUL
Total time: 11 seconds
dhcp-41-105:crawlerAntPackage sanjivkava$
```

If the run is up to date and the PullRSS class has been called, this is the expected output to console should look relatively similar to this:

A terminal window titled 'crawlerAntPackage — bash — 138x47' showing the execution of the 'ant PullRSS' command. The prompt is 'dhcp-41-105:crawlerAntPackage sanjivkava\$'. The output shows the build file path, compilation warnings, and the PullRSS task results.

```
dhcp-41-105:crawlerAntPackage sanjivkava$ ant PullRSS
Buildfile: /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build.xml

compile:
[javac] /Users/sanjivkava/Documents/NetBeansProjects/crawlerAntPackage/build.xml:22: warning: 'includeantruntime' was not set, default
ing to build.sysclasspath=last; set to false for repeatable builds

jar:

PullRSS:
[java] The last date that the crawler was run is: Tue Oct 16 03:54:11 EST 2012
[java] No more updates to process

BUILD SUCCESSFUL
Total time: 6 seconds
dhcp-41-105:crawlerAntPackage sanjivkava$
```

3. Special Conditions

PullRSS will only work after a crawl or partial crawl has been initialized. This is because the class relies on the Last Run Date Table to be created and populated; which is only done after the crawler has started a run. PullRSS is used as an update feature as opposed to a full run feature.

4. File Storage

The program downloads and processes a number of files. These files are created after the crawler has been initialized:

HTMLandJSON.db is the primary database for the program, it is stored under:

```
communityWebcrawler/HTMLandJSON.db
```

The program downloads every HTML Page that the crawler processes. These are located in:

```
communityWebcrawler/communityCrawlerStorage/HTMLPages/
```

Every HTML file is processed in order to strip the raw code and extract the relevant plain text data. This data is stored in JSON files, located in:

```
communityWebcrawler/communityCrawlerStorage/JSON/
```

The PullRSS program utilizes an RSS file in order to download all of the updated or new HTML files. This is stored under:

```
communityWebcrawler/thread.xml
```

5. Major Upgrades

Version 2.0 works on the same crawler platform as Version 1.0. There have been many additional features added in order to tailor the crawler to find security relevant issues from the JBoss Community Forums. The upgrades are as follows:

- 5.1 HTMLParser
- 5.2 ScoringAlgorithm
- 5.3 Merging
- 5.4 PullRSS
- 5.5 Database

5.1 HTMLParser

The HTMLParser utilizes a HTML flat file, which is provided by the crawler after every page download.

In order to gain an understanding of the files format the class will open the HTML file and parse it. From there the file is loaded into an object that exists in memory.

The information that is extracted from the raw HTML file is the pages title, the users original post, and all of the following reply posts.

The way that I chose to do this was through recursion.

The program will hit the body division of the HTML page. The objective is to pull out the first post, which will always be the original post in a properly structured HTML page. This is done by hitting the first p tag, or paragraph tag within the body, It will then store that post into array index 1 and then delete the p tag from memory, not from the file.

This results in all of the p tags shifting up, so the next run through the loop will cause the 2nd p tag which will be the start of the replies to be stored in index 2, that is then deleted from memory and the loop is continued.

After all p tags have been exhausted in the body and stored in the array list the HTML processor will then remove all raw HTML data and output a plaintext JSON file containing the title of the page, the original post and the reply posts.

Just to note, the JSON's file name will match the HTML's file name for consistency.

This is a fairly universal program that just requires a HTML page from a forum. I can't claim it to be completely universal as I have only tested it on 3 websites and received 3 successful results.

5.2 ScoringAlgorithm

The ScoringAlgorithm utilizes the JSON file that is outputted from the HTMLParser. It will parse the JSON file and load it into memory.

After this, it will open up a properties file that contains keywords and phrases that are security relevant. This properties file is user definable.

The keywords are then matched against the plain text in the JSON file using a string matching algorithm. If a keyword is found it is inserted into a hash map along with its frequency count.

An example of this is if the word “security” was found 5 times then “security” will be a key in the hash map with the value 5.

The populated frequency hash map is then passed into a scoring algorithm, which will determine a percentile score of how relevant the current JSON file is.

This percentage score is then compared against a user defined threshold.

Lets assume that the threshold is set to 30%

If the score is greater than or equal to 30% then we have a valid security issue, if not, then we don't.

If the security issue is valid then the JSON's file name, keywords and frequency count over 0 and total score are written out to a SQLite DB.

5.3 Merging

I had to select the appropriate place for my classes to be called. I determined that this was in the class titled WriteURLToFile. This class would be the driver for my classes.

WriteURLToFile quite simply is where the crawler has defined how to save the HTML files that the HTML Processor needs.

Directly after a HTML file is downloaded it is put through the HTMLParser and then a JSON is outputted and put directly into the ScoringAlgorithm that outputs scores to the SQLite DB.

I noticed that there was no real way of keeping track of what file number is related to a live URL version. So I committed some changes in the code and a map that shows the HTML page number, live URL and JSON file number is outputted to an SQLite DB.

5.4 PullRSS

Crawler Version 1.0 had no way of tracking which files it had previously downloaded and whether or not they need to be updated. This is the sole purpose of PullRSS is to add this functionality in.

PullRSS will only work after a crawl or partial crawl has been initialized. This is because the class relies on the Last Run Date Table to be created and populated; which is only done after the crawler has started a run.

PullRSS will grab the current thread.xml file from the JBoss Community RSS feed. It will then compare the dates of the file against the last date that the crawler started its run.

If the crawlers date is before the first date (most recent post) in the thread.xml file, the class will start pulling and updating the previously downloaded crawler file. It will simply overwrite any previous file. If the previous file does not exist it will create it.

It will then feed the new file into the HTMLParser and ScoringAlgorithm class, in order to determine whether or not the post is a relevant security issue.

PullRSS keeps track of these files in the Updated_Files_Table within HTMLandJSON.db. The files ID, HTML file number, URL and Update time is recorded.

5.5 Database

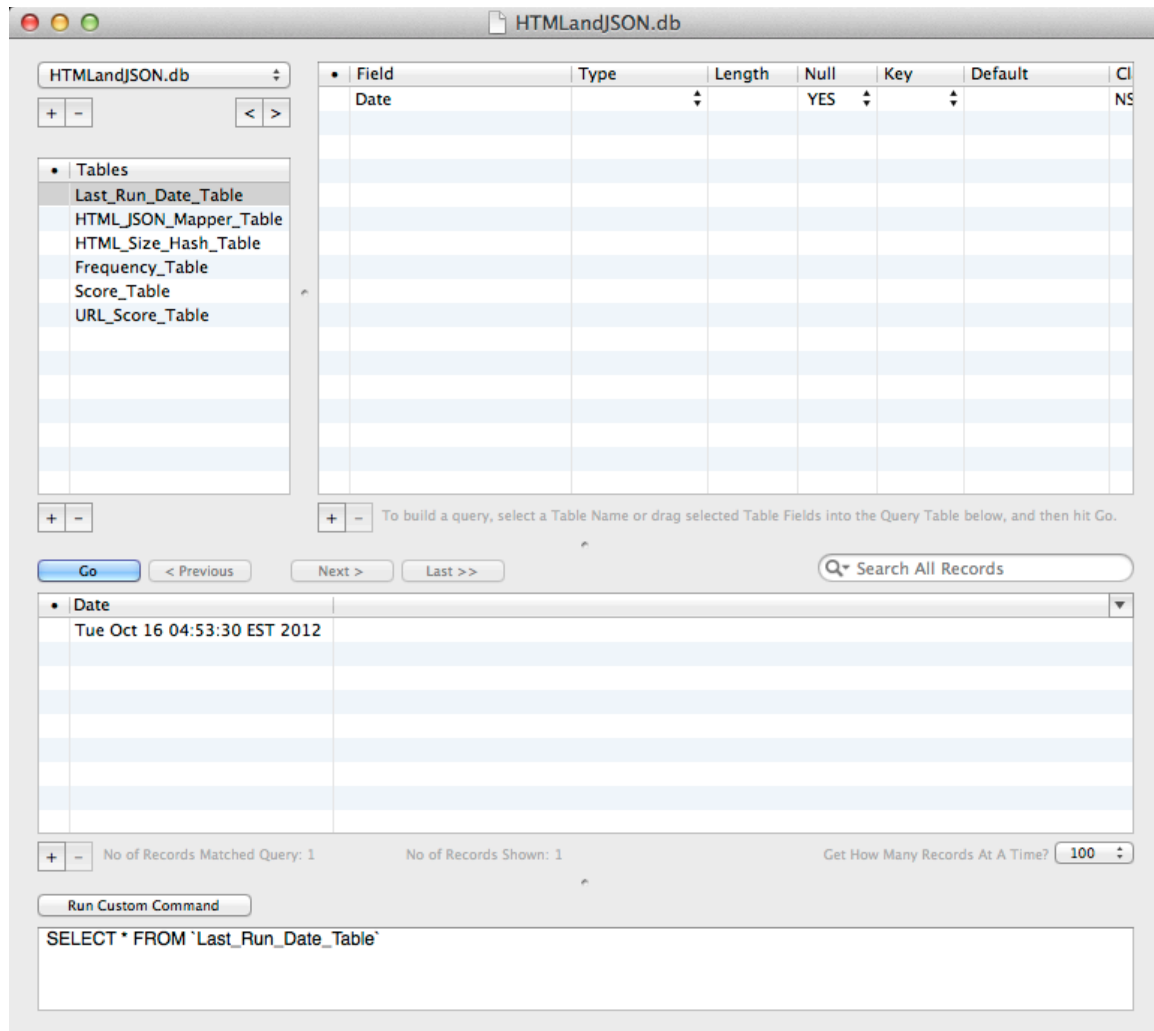
The crawler now has a detailed database called HTMLandJSON.db. The purpose of this was to have an organized structure that contains information relating to each file.

There are 7 tables that will be discussed below:

Last Run Date Table

This contains the last date that the crawler has started its run. The table is used by PullRSS in order to determine what files need to be pulled based on the previous start date.

For example, if the last run was October 10th 2012. PullRSS would start downloading all files that have changed since October 10th 2012.



HTML JSON Mapper Table

This table stores the ID, URL, HTML File number and JSON File number of each page. It acts as a relation map so that the URL is logically mapped to its downloaded HTML file and JSON file.

The screenshot shows a database management application window titled "HTMLandJSON.db". On the left, a sidebar lists several tables: "Last_Run_Date_Table", "HTML_JSON_Mapper_Table" (which is selected and highlighted in purple), "HTML_Size_Hash_Table", "Frequency_Table", "Score_Table", and "URL_Score_Table".

The main area displays the structure of the selected "HTML_JSON_Mapper_Table". It is a table with the following columns: Field, Type, Length, Null, Key, Default, and Charset. The data rows are:

Field	Type	Length	Null	Key	Default	Charset
ID			YES			NS
HTML_File			YES			NS
HTML_URL			YES			NS
JSON_File			YES			NS

Below the structure table, there is a "Go" button and a search bar labeled "Search All Records".

The main data table shows the following records:

ID	HTML_URL	HTML_File	JSON_File
1	https://community.jboss.org/thread/	1.html	1.json
2	https://community.jboss.org/threads/	2.html	2.json
3	https://community.jboss.org/thread/206709?tstart=0	3.html	3.json
5	https://community.jboss.org/thread/206577?tstart=0	5.html	5.json
8	https://community.jboss.org/thread/206697?tstart=0	8.html	8.json
9	https://community.jboss.org/thread/197780	9.html	9.json
10	https://community.jboss.org/thread/206666?tstart=0	10.html	10.json
11	https://community.jboss.org/thread/199201?tstart=0	11.html	11.json
13	https://community.jboss.org/thread/177302	13.html	13.json

At the bottom, there is a "Run Custom Command" button and a text area containing the SQL query: "SELECT * FROM 'HTML_JSON_Mapper_Table'".

HTML Size Hash Table

This table stores the size of each HTML file as well as the md5 hash of each file.

The screenshot shows a database management interface for a file named 'HTMLandJSON.db'. On the left, a sidebar lists several tables: 'Last_Run_Date_Table', 'HTML_JSON_Mapper_Table', 'HTML_Size_Hash_Table' (which is selected), 'Frequency_Table', 'Score_Table', and 'URL_Score_Table'. The main area displays the structure of the 'HTML_Size_Hash_Table' with the following fields:

Field	Type	Length	Null	Key	Default	Cl
ID			YES			NS
HTML_Byte_Size			YES			NS
HTML_MD5			YES			NS

Below the table structure, there is a 'Go' button and a search bar labeled 'Search All Records'. The results of the query are displayed in a table with the following data:

ID	HTML_Byte_Size	HTML_MD5
1	122644	[B@33c13d0b
2	122644	[B@7786f3b3
3	73396	[B@1c33c658
5	92813	[B@1de4eb5b
8	72575	[B@40c5577c
9	139439	[B@4e1a70b8
10	134808	[B@6c43da1b
11	114216	[B@4aee260b
13	152132	[B@451415c8

At the bottom, there is a 'Run Custom Command' button and a text area containing the SQL query: `SELECT * FROM 'HTML_Size_Hash_Table'`. Below the text area, it shows 'No of Records Matched Query: 118' and 'No of Records Shown: 118'. On the right, there is a dropdown menu for 'Get How Many Records At A Time?' set to 'All'.

Frequency Table

The frequency table holds the ID, keyword and frequency of a HTML File. The ID can be looked up in the HTML_JSON_Mapper_Table. The purpose of this table is to map the exact keyword and frequency found for each security relevant post.

HTMLandJSON.db

HTMlandJSON.db

Tables

- Last_Run_Date_Table
- HTML_JSON_Mapper_Table
- HTML_Size_Hash_Table
- Frequency_Table
- Score_Table
- URL_Score_Table

Field	Type	Length	Null	Key	Default	Cl
ID			YES			NS
Keyword			YES			NS
Frequency			YES			NS

To build a query, select a Table Name or drag selected Table Fields into the Query Table below, and then hit Go.

Go

Search All Records

ID	Keyword	Frequency
17	security	13
17	risk	1
17	confidential	2
21	security	5
131	security	71
131	risk	1
131	confidential	7
135	security	42
135	confidential	1

No of Records Matched Query: 16 No of Records Shown: 16 Get How Many Records At A Time? All

Run Custom Command

```
SELECT * FROM 'Frequency_Table'
```

Score Table

This table provides the exact percentile format score that each security relevant page received. It provides the ID of the page, Score and JSON file number.

The screenshot shows a database management interface for a file named 'HTMLandJSON.db'. On the left, a sidebar lists several tables: 'Last_Run_Date_Table', 'HTML_JSON_Mapper_Table', 'HTML_Size_Hash_Table', 'Frequency_Table', 'Score_Table' (which is selected), and 'URL_Score_Table'. The main area displays the structure of the 'Score_Table' with the following columns: Field, Type, Length, Null, Key, Default, and Cl. The table structure is as follows:

Field	Type	Length	Null	Key	Default	Cl
ID			YES			NS
Score			YES			NS
JSON_File			YES			NS

Below the table structure, there is a 'Go' button and a search bar labeled 'Search All Records'. The main data view shows a list of records from the 'Score_Table' with columns: ID, Score, and JSON_File. The records are as follows:

ID	Score	JSON_File
17	46.875	17.json
21	31.25	21.json
131	100.0	131.json
135	100.0	135.json
136	46.875	136.json
137	61.11111111...	137.json
157	31.25	157.json
159	37.5	159.json

At the bottom, there is a 'Run Custom Command' button and a text area containing the SQL query: `SELECT * FROM `Score_Table``. Below the text area, it shows 'No of Records Matched Query: 8' and 'No of Records Shown: 8'. On the right, there is a dropdown menu for 'Get How Many Records At A Time?' set to 'All'.

URL Score Table

This table provides a mapping between the live security related URL of a page and its score.

HTMLLandJSON.db

+ -

< >

• Tables

Last_Run_Date_Table

HTML_JSON_Mapper_Table

HTML_Size_Hash_Table

Frequency_Table

Score_Table

URL_Score_Table

+ -

To build a query, select a Table Name or drag selected Table Fields into the Query Table below, and then hit Go.

Go

Q Search All Records

ID	HTML_URL	Score
17	https://community.jboss.org/thread/172052	46.875
21	https://community.jboss.org/thread/199990	31.25
131	https://community.jboss.org/thread/172052?decorator=print&displayFullThread=true	100.0
135	https://community.jboss.org/thread/172052?start=45&tstart=0	100.0
136	https://community.jboss.org/thread/172052?start=0&tstart=0	46.875
137	https://community.jboss.org/thread/172052?start=30&tstart=0	61.11111111111111
157	https://community.jboss.org/thread/199990?start=0&tstart=0	31.25
159	https://community.jboss.org/thread/199990?decorator=print&displayFullThread=true	37.5

+ -

No of Records Matched Query: 8

No of Records Shown: 8

Get How Many Records At A Time?

All

Run Custom Command

SELECT * FROM `URL_Score_Table`

Updated Files Table

This table provides a listing of the updated files pulled from the JBoss Community RSS feeds via the PullRSS class.

The screenshot shows a database management application window titled "HTMLandJSON.db". On the left, a list of tables includes "Updated_Files_Table", which is currently selected. Above this list is a small table showing field metadata:

Field	Type	Length	Null	Key	Default	Cl
ID			YES			NS
HTML_File			YES			NS
HTML_URL			YES			NS
Date			YES			NS

Below the table list, a "Go" button is visible. To the right of the "Go" button, a text box contains the instruction: "To build a query, select a Table Name or drag selected Table Fields into the Query Table below, and then hit Go." Below this, a search bar labeled "Search All Records" is present.

The main area of the application displays the data from the "Updated_Files_Table" in a table with the following columns: ID, HTML_File, HTML_URL, and Date. The table contains 5 records:

ID	HTML_File	HTML_URL	Date
206711	206711.html	https://community.jboss.org/thread/206711	Tue Oct 16 05:12:41 EST 2012
206669	206669.html	https://community.jboss.org/thread/206669	Tue Oct 16 05:24:22 EST 2012
206688	206688.html	https://community.jboss.org/thread/206688	Tue Oct 16 05:26:55 EST 2012
206670	206670.html	https://community.jboss.org/thread/206670	Tue Oct 16 05:30:44 EST 2012
206712	206712.html	https://community.jboss.org/thread/206712	Tue Oct 16 05:32:48 EST 2012

Below the data table, a status bar shows "No of Records Matched Query: 5" and "No of Records Shown: 5". To the right, a dropdown menu for "Get How Many Records At A Time?" is set to "100".

At the bottom, there is a section for "Run Custom Command" with a text area containing the SQL query: `SELECT * FROM 'Updated_Files_Table'`.

6. New Classes and Functions

Due to all of the new functionality there have been the creation of many new classes and functions in Version 2.0. These are listed and briefly defined below.

6.1 HTMLParser

(Detailed description of this class is located in Section 5.1)

HTMLParser Functions:

Document input(File): This function will attempt to parse the HTML file that is received as a parameter input. The function returns a parsed version of the current HTML file

ArrayList processHTML(Document): The purpose of this function is to locate the areas within the body section of the HTML file that contain text based paragraphs. The function receives the parsed HTML file from the parseHTML function. Once the paragraphs are found within the parsed HTML file, they are converted from HTML format to plaintext. An ArrayList will hold the title of the HTML page, as well as the individual paragraphs in separate indexes. The function returns HTMLPlainText, which is an ArrayList that contains the plaintext format of the HTML page's title as well as the individual paragraphs in separate indexes

File writeToJSON(ArrayList, File): The purpose of this function is to receive the ArrayList from processHTML and output the contents of it to a JSON file. The JSON file that is outputted will match the file name of the HTML file that has been processed. Before the ArrayList can be outputted to a JSON format, the filename must be converted from .html to .json. The function returns the JSON file

6.2 ScoringAlgorithm

(Detailed description of this class is located in Section 5.2)

ScoringAlgorithm Functions:

JSONObject parseJSON(File): This function will attempt to parse the JSON file that is received as a parameter input. This function returns jsonObject, which is a parsed version of the JSON input file

HashMap iterateSecurityWords(JSONObject): This function will open the securityWords properties file and pass the current security keyword into the processJSON function. The processJSON function will then return the frequency count of the current security word. Both the security word and the occurrence will be stored into a hash map. This function returns keywordFrequency, which is the occurrence of all security related words in the current JSON file

Integer processJSON(JSONObject, String): The purpose of this function is to compare the current security keyword against the current JSON file. A frequency count will be generated from this. This function utilizes the StringUtils function from the apache common lang3 package. This function returns freqCount, which is the occurrence of the given security keyword.

boolean scoreJSON(HashMap): This function will iterate through the provided HashMap and pick out the keywords with correlating frequencies. It will then provide a score of the JSON file and return a T/F Boolean called validSecurity. This is what suggests whether or not the JSON file is a valid security issue.

void writeSecurityIssue(Boolean, File, HashMap): The purpose of this function is to write out the valid security issue to a .db file. The file will be appended with each valid JSON file. The contents of the file will contain the JSON file name, the keyword to frequency map and the matching percentile. This function will only process if the validSecurity boolean is true.

6.3 PullRSS

(Detailed description of this class is located in Section 5.4)

PullRSS Functions:

URL xmlFile(): This function will attempt to open the XML file that contains the recent updated files from the JBOSS community forums. This function returns xmlFile which is a File that contains the recent updated files in the JBOSS community forums.

void processXML(File): The xmlFile that is passed in as a parameter is parsed, from there the first tag within the XML file that is hit is the entry tag. Nested within this tag is the updated tag which holds the date and time that the thread was updated. The HREF tag is also nested within the entry tag, this holds the link of the correlating entry. This function will pull all updated tags and HREF tags from each entry in the XML file and store it into a HashMap.

`void RSSPuller(HashMap)`: Initially this function will attempt to retrieve the `lastRunDate` that the crawler was run from the `HTMLandJSON.db` file. From there it will compare the `lastRunDate` against all dates in the RSS file. If the date of the thread within the RSS file is after the `lastRunDate` then the correlating HTML files are downloaded. The previous copies of the HTML files are overwritten by the updated ones. A database table is created to keep track of which files are updated.

`String getTagValue(String, Element)`: The purpose of this function is to traverse the entry tag looking for a provided String tag name as well as the provided element. This function returns `nValue.getNodeValue()`; the current value of the node.

`String formatHTMLString(String)`: This function takes in the raw URL string and formats it. This function returns `html`, which is a sanitized URL String.

`Date formatDate(String)`: This function takes in a raw date string, formats it and then parses it to a Date data type. This function returns `time`, which is a formatted Date.

`String getFileNameFromURL(URL)`: This function will return the `fileName` of the URL that is passed in.

`void writeHTMLFile(URL)`: After taking in a URL, the entire page is downloaded and stored as a .HTML file.

`Date lastCrawlerRun(HashMap)`: This function will take in a HashMap via parameter. From there it will iterate through the HashMap in order to locate the `lastRunDate` of the crawler. This function returns `lastRunDate`, which will set the last run date of the crawler.

`Date getLastCrawlerRun()`: This function accesses the `Last_Run_Date_Table` within the `HTMLandJSON.db` flat file. It returns `lastDate` from the database, which is the last Date that the crawler was run.

6.4 HTMLParserAndSecurityController

This class contains one function that acts as a driver for the HTMLParser and ScoringAlgorithm classes.

`void run(File inputFile, String pageURL)`: This is for the HTMLParser and JSON scoring. This class will bring in the most recently downloaded HTML file. It will then pass all HTML files into the input function for parsing. Once the HTML files are parsed, they are passed into the HTMLProcessor function so that relevant plaintext can be extracted and stored into an ArrayList. Finally the HTML Parser program outputs a JSON file that holds the contents of the ArrayList from the HTMLProcessor function. The ScoringAlgorithm is first introduced when `writeToJSON` is called from the HTMLParser class. The `writeToJSON` function returns the recently parsed JSON file. The JSONFile is then parsed and matched against security keywords in order to identify a frequency. This is done within the `iterateSecurityWords` function. The JSON is then scored and determined whether or not it is a valid security issue. A database is then created called HTMLandJSON. There are 2 tables within the database. The first is HTML_JSON_DB - this will hold the HTML file number and JSON file number for a specific URL. The second is JSON_Frequency_DB - this will hold the JSON file number, keyword and frequency count > 0 as well as total score.

6.5 InsertLastCrawlerRun

This class contains one function that is responsible for creating a entry in the database for the start of the crawler.

`void insertLastRunDate()`: This function will retrieve the lastCrawlerRun date and write it into the database.

6.6 MakeDownloadDirs

This class is called as soon as the crawler is run. It has one function that is responsible for creating the storage directory for the downloaded HTML and JSON files.

`void createDirs()`: Simply enough, this function creates the storage directories for the HTML and JSON files. It is called on program start up.

6.7 GetLastCrawlerRunTime

This class contains one function that is responsible for handling the RSS XML file from the JBoss community forums and inserting the extracted date into the database.

Date lastCrawlerDate(): This function will grab the XML file via the JBoss community forums RSS feed. From there it will extract the first entry and store it before the crawler is run to signify that the date and time of the last post was the start time of the crawler being run. This function returns lastCrawlerDate, which is a Date that specifies the last date that the crawler was run.

6.8 sqliteJdbcForHTMLandJSON

This class is responsible for the creation and modifications for all database entries.

sqliteJdbcForHTMLandJSON Functions:

void database(Integer, String, File, byte[]):

This function will create and insert entries into the HTML_Size_Hash_Table and HTML_JSON_Mapper_Table within the HTMLandJSON.db. It will be called through out the programs structure in order to create entries into the db tables.

void database(Integer, String): This function will create the HTML_JSON_Mapper_Table within the HTMLandJSON.db. It will be called through out the programs structure in order to create entries into the db tables.

void crawlerRunDates(Date): This function will create and insert entries into the Last_Run_Date_Table within the HTMLandJSON.db. It will be used to store the last run date of the crawler.

void crawlerRunDates(Date): This function will create the Last_Run_Date_Table within the HTMLandJSON.db.

7. Crawler Source Code Modifications

This section will outline the changes made in Version 2.0 with regard to the crawlers existing source code from Version 1.0.

Source Code:

CrawlerController.java:

- I added in a call to the MakeDownloadDirs class within the main class, so that the download directories are created on program start.
- I added in a call to the InsertLastCrawlerRun class within the main class, so as soon as the crawler is initiated the start date is inserted from the RSS feed into the database.

DownloadPage.java

- I made some major additions to the writeURLtoFile(Page) function. This is where the crawler downloads the HTML pages. Initially the HTML pages were being saved without the .html extension. I corrected this. It seemed logical to call the HTMLParserAndSecurityController class here so that after a page is downloaded it is immediately passed into that class and processed. The function is also responsible for creating a hash value for each file that is downloaded and then writing said hash value into the database.

Config.java

- I made an addition to the visit(Page) function. If the incoming URL string matches the pattern of the site.properties URL then the page will be downloaded into a HTML file. If it does not, then the page will not be downloaded. Currently this restricts pages to be downloaded from either community.jboss.org/thread/ or community.jboss.org/threads/

Property Files:

crawlerLogic.properties:

- I changed the storage location of files to communityCrawlerStorage/

securityWords.properties:

- I created a new properties file that holds the security keywords for the pattern matching algorithm.

config.properties:

- I changed the storage location of files to communityCrawlerStorage/
- I created an entry for the HTML files to be downloaded here:
communityCrawlerStorage/HTMLPages/
- I created an entry for JSON file to be downloaded here:
communityCrawlerStorage/JSON/

sites.properties:

- I changed the URL to <https://community.jboss.org/threads/> instead of <https://community.jboss.org/thread/>
- I added in restrict1 and restrict2, these are called so that the crawler is restricted to download files matching these URL formats.

Build File:

build.xml:

I wrote an ant build file that is responsible for cleaning, compiling and running both the CrawlerController and PullRSS classes.

8. Additional Files

Some additional files that have been created in Version 2.0 that are non-existent in Version 1.0 are:

batchDelete.pl This Perl file is used to permanently delete:

- communityCrawlerStorage + communityCrawlerStorage/*
- *.db
- thread.xml
- build + build/*

Ideally batchDelete.pl was used to delete all files in order to re-test the configurations of the crawler. However it can be used alternatively to delete all files if necessary.

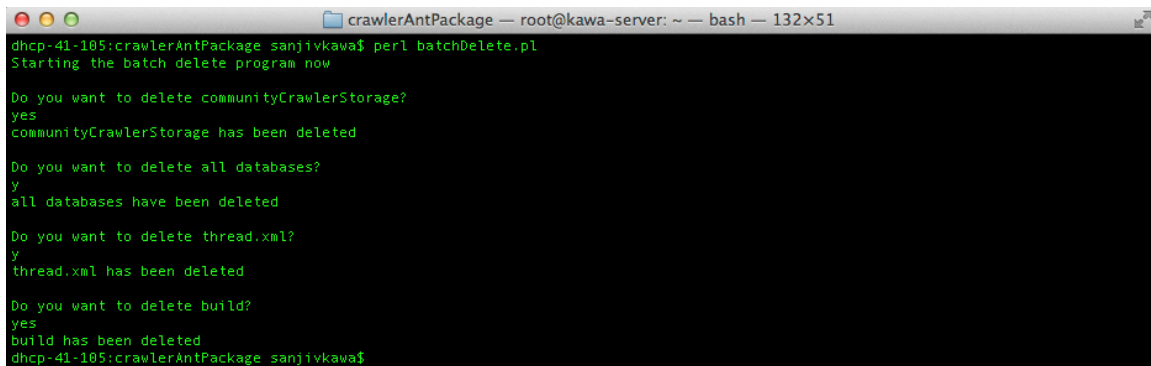
To run batchDelete.pl:

```
perl batchDelete.pl
```

To delete file or directory type: y or yes

Else, file(s) will not be deleted.

Expected Perl console output:



```
dhcp-41-105:crawlerAntPackage sanjivkawa$ perl batchDelete.pl
Starting the batch delete program now

Do you want to delete communityCrawlerStorage?
yes
communityCrawlerStorage has been deleted

Do you want to delete all databases?
y
all databases have been deleted

Do you want to delete thread.xml?
y
thread.xml has been deleted

Do you want to delete build?
yes
build has been deleted
dhcp-41-105:crawlerAntPackage sanjivkawa$
```

README.txt: this file has quick instructions of how to run the program.