Data Driving Security
FireHOL

Package Created By:
Pol S.
Guillem G.
Jeffrey M.

About the package

This package is free for use free of charge. Can be used as many times as you like to and query as much as you want since all the data is downloaded once. This packages is highly recommended under investigations, analysis of attacks and so on.

Functions

- GetDataFrame

The function **GetDataFrame** is in charge of creating the first Dataframe with it's possible values.

Value	Description
IP	This value is the IP of the host/machine we will query on
Categoria	There are different categories available using the command ***list.category***
Pais	There are different countries as well, available with the command ***list.country***

```
#'
#' GetDataFrame
#'
#'
#'
#' @return dataset
#' @export
#'
#' @examples
getDataFrame <- function() {
  IPS <- data.frame(ip = character(),
                  categoria = character(),
                  pais = character(),
                  stringsAsFactors = F)
  print(save.path)
  return(IPS)
}
```

- downloadFireHolData

The function **downloadFireHolData** download the source data that's used in this project to your local machine. First we set the function to save it in a temporary directory and using the build in download.file function we get the master.zip file from the github repository. After it we set the destination file and the name of how it will be saved. Using again one of the build-in tools of R we unzip the data and save it uncompressed.

```
#'
#' Downloads data from scratch.
#'
#' @param save.path
#'
#' @export
#'
#' @examples
downloadFireHolData <- function(save.path = tempdir()) {
  download.file(url = "https://github.com/firehol/blocklist-ipsets/archive/master.zip",
               destfile = paste(save.path, "\\", "master.zip", sep = ""))
  utils::unzip(zipfile = paste(save.path, "\\", "master.zip", sep = ""),
              exdir = paste(save.path, "\\", "data", sep = ""))
  return(paste(save.path, "\\", "data", sep = ""))
}
```

- tidyDataIPs

The function **tidyDataIPs** explains itself. This will clean the IPs downloaded from the source and create a DataFrame. It uses all the .ipset files found in the raw.path variable saved as src.files. Then there's a loop assigned that for each file_name in the src.files we created moves it to a temporary DataFrame (More info about how to create a DataFrame reading *read.table*). Later we test whether tmp is empty or not with *nrow*, if it's not we assign a Category value to it and then we use *rbin* function to combine the value **ips** and **tmp** creating the final DataFrame. After this function the table is clean and combine ready to be used.

```
#'
#' @param working.directory
#'
#' @return ips
#' @export
#'
#' @examples
tidyDataIPs <- function(raw.path) {
 src.files <- list.files(path = raw.path, pattern = ".ipset")
 ips <- data.frame(ip = character(),
                   categ = character(),
                   country = character(),
                   stringsAsFactors = F)

 for (file_name in src.files){
   tmp <- read.table(file = paste(raw.path, file_name, sep = ""), skipNul = T,
                     col.names = c("ip"), na.strings = "NULL", stringsAsFactors = F)
   if (nrow(tmp) > 0) {
    file_info <- read.table(file = paste(raw.path, file_name, sep = ""),
                            comment.char="/", sep = "\t", quote = "", stringsAsFactors = F, nrows=50 )
    categ <- dplyr::filter(file_info, stringr::str_detect(V1,"Category"))
    if(nrow(categ) > 0) {
      categ <- stringr::str_trim(stringr::str_split(categ$V1, ":")[[1]][2])
      tmp$categ <- rep(x = categ, nrow(tmp))
      ips <- rbind(ips, tmp)
    }
   }
 }
 ips$country <- rep(x = " ", nrow(ips))
 return(ips)
}
```

- tidyDataCountries

The function **tidyDataCountries does the same as tidyDataIPs** since it cleans it by creating a new DataFrame that matches IP ranges and Countries.
This time we choose the folder geolite2_country and choose those with the .netset extension.
We set countries to NULL in order to clean the list. Later we use a loop to read all the file and apply to them the RegEx that will find the line where the Country is set and saved as country_tmp. Now let's save in tmp2 a column with ranges called range.
So for each ip in the column range search if the ip itself is within a range (check *iptools::range_boundaries* for more information). Later on it check that if the country_tmp is ok (greather than 0) then removes all the unnecessary characters and gets the value. After all the parsing is done the new DataFrame is created.

```
#'
#' @param working.directory
#'
#' @return countries
#' @export
#'
#' @examples
tidyDataCountries <- function(raw.path) {
  raw.path <- paste(path.raw.data, "geolite2_country", sep="")
  raw.path <- paste(raw.path, "\\", sep="")
  src.files <- list.files(path = raw.path, pattern = ".netset")
  countries <- NULL

  for (file_name in src.files){
    file_info <- read.table(file = paste(raw.path, file_name, sep = ""),
                        comment.char="/", sep = "\t", stringsAsFactors = F,
                        nrows = 50, row.names = NULL)
    country_tmp <- dplyr::filter(file_info, stringr::str_detect(V1,"--"))
    tmp2 <- read.table(file = paste(raw.path, file_name, sep = ""),
                   skipNul = T, na.strings = "NULL", col.names = c("range"),
                   stringsAsFactors = F, row.names= NULL)

    for (ip in tmp2) {
      tmp_boundary <- iptools::range_boundaries(ip)
      tmp2$min <- tmp_boundary[1]$minimum_ip
      tmp2$max <- tmp_boundary[2]$maximum_ip
    }
    if(nrow(country_tmp) > 0) {
      country <- stringr::str_trim(stringr::str_split(country_tmp$V1, "#")[[1]][2])
      country <- stringr::str_trim(stringr::str_split(country, ",")[[1]][1])
    }
    tmp2$country <- rep(x = country, nrow(tmp2))
    countries <- rbind(countries,tmp2)
  }
  return(countries)
}
```

- look_countries

The function **look_countries** is not shown to the public since it’s an internal feature to look for the correct country. What it does is is check the countries

```
#'
#' @param rango
#' @param df2
#'
#' @return
#' @export
#'
#' @examples
look_countries <- function(df2, df_row){
  temp <- iptools::ip_in_range(df_row[1], df2[1])
  if (temp){

    return(df2[4])
  }else {
    return(FALSE)
  }
}
```

- look_countries2

The function **look_countries2** is not shown to the public neither and check the same as **look_countries** does.

```
#'
#' @param rango
#' @param df2
#'
#' @return
#' @export
#'
#' @examples
look_countries2 <- function(df2, df_row){
  temp <- iptools::ip_in_range(rango, df2[1])
  if(test){
    print(rango)
    print(df2)
  }
}
```

- ips_merge

The function **ips_merge** is not shown to the public. Merge the rows in a bigger DataFrame.

```
#'
#' @param df
#' @param df2
#'
#'
#' @export
#'
ips_merge <- function(df_row,df2){
  if(iptools::is_ipv4(df_row[1])){
    for (i in 1:nrow(df2)) {
      df2_row <- df2[i,]
      if(iptools::ip_in_range(df_row[[1]],df2_row[1])){
        print(df2_row[4])
        df_row[3] <- df2_row[4]
        break
      }
    }
  } else{
    df_row[3] <- Unknown
  }
}
```

- mergeDFs2

The function **mergeDFs2** is not shown to the public. Merges everything in order to apply the location from the countries DataFrames to the IPs. This version uses the *apply* function since we couldn’t find another way to do so.

```
#'
#' @param df
#' @param df2
#'
#' @return
#' @export
#'
#' @examples
mergeDFs2 <- function (df,df2){
  apply(df,1,ips_merge,df2)
}
```

- mergeDFs

The function **mergeDFs** merge two different DataFrames in order to apply the location to the IPs. ***Currently not working it's still under development.***

```
#'
#' @param ips
#' @param countries
#'
#' @return final_ips
#' @export
#'
#' @examples
mergeDFs <- function(df,df2){
  for (filadf in df){
    ip <- filadf[[1]]
    if (iptools::is_ipv4(ip)){
      for(filadf2 in df2){
        rango <- filadf2[1]
        if (filadf2[2] != "Invalid"){ #it is an ip +a mask
          if (iptools::ip_in_range(ip, rango)){
            df[i,3] = df2[j,4]
          }
        }
        else { # compare IPs
          if (ip == rango) df[i,3] = df2[j,4]
        }
      }
    }
  }
  return(df)
}
```

- list.ip.count

The function **list.ip.count** returns the count of how many times an attack has matched an IP entered by the user. Then a new DataFrame called IPsFound will be displayed.

```
#'
#' A l'introduir una IP retorna la quantitat d'atacs rebuts
#'
#' @param ip
#'
#' @return dataset
#' @export
#'
#' @examples
#' IPsRecount <- list.ip.count('5.188.86.174')
list.ip.count <- function(ip){
  IPsFound <- IPS[IPS$ip == ip,]

  return(sum(duplicated(IPsFound$ip)) + 1)
}
```

- list.iprelated.dataset

The function **list.iprelated.dataset** returns a DataSet of attacks classified by campaign and country.

```
#'
#' A l'introduir una IP mostra el dataset dels atacs que ha rebut
#' classificada pel tipus de campanya i pais
#'
#' @param ip
#'
#' @return dataset
#' @export
#'
#' @examples
#' IPRelatedDataset <- list.iprelated.dataset('5.188.86.174')
list.iprelated.dataset <- function(ip){
  IPsFound <- IPS[IPS$ip == ip,]

  count <- plyr::count(IPsFound)
  colnames(count)[4] <- "appearances"
  return(count)
}
```

- list.category

The function **list.category** show all the available categories.

```
#'
#'@param IPS
#'
#' @return list
#' @export
#'
#' @examples
#' ListCategories <- list.category()
list.category <- function(IPS){
  return(dplyr::distinct(IPS[2]))
}
```

- list.category.count

The function **list.category.count** returns a dataset of the quantity of attacks matching a category

```
#'
#' @param IPS
#'
#' @return dataset
#' @export
#'
#'
#' @examples
#' CategoriesCount <- list.category.count()
list.category.count <- function(IPS){
  return(as.data.frame(table(IPS[2])))
}
```

- list.country

The function **list.country** returns a list of countries that does have IPs matching an attack.

```
#'
#' @param IPS
#'
#' @return list
#' @export
#'
#' @examples
#' ListCountries <- list.country()
list.country <- function(){
  return (dplyr::distinct(IPS[3]))
}
```

- list.country.count

The function **list.country.count** returns a list of countries and the number of attacks registered for each country.

```
#'
#' @param IPS
#'
#' @return dataset
#' @export
#'
#' @examples
#' list.category.count(IPS)
list.country.count <- function(IPS){
  return (table(IPS[3]))
}
```