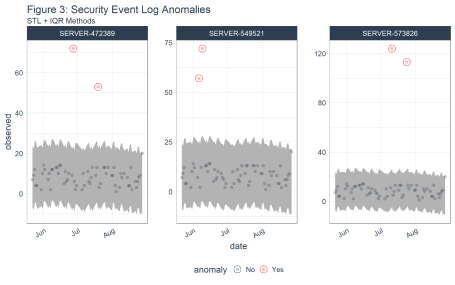
[Information Security (InfoSec)](https://en.wikipedia.org/wiki/Information_security) is critical to a business. For those new to **InfoSec, it is the state of being protected against the unauthorized use of information, especially electronic data. A single malicious threat can cause massive damage to a firm, large or small.** It’s this reason when I (Matt Dancho) saw [Russ McRee’s](https://twitter.com/holisticinfosec) article, [“Anomaly Detection & Threat Hunting with Anomalize”](https://holisticinfosec.blogspot.com/2018/06/toolsmith-133-anomaly-detection-threat.html), that I asked him to repost on the [Business Science blog](http://www.business-science.io/blog/index.html). In his article, Russ speaks to use of our new R package, anomalize, as a way to detect threats (aka **“threat hunting”**). Russ is Group Program Manager of the [Blue Team](https://danielmiessler.com/study/red-blue-purple-teams/) (the internal security team that defends against real attackers) for Microsoft’s Windows and Devices Group (WDG), now part of the Cloud and AI (C+AI) organization. He writes [toolsmith](https://holisticinfosec.blogspot.com/), a monthly column for information security practitioners, and has written for other publications including Information Security, (IN)SECURE, SysAdmin, and Linux Magazine. **The data Russ routinely deals with is massive in scale**: He processes [security event telemetry](https://en.wikipedia.org/wiki/Telemetry) of all types (operating systems, network, applications, service layer) for all of Windows, Xbox, the Universal Store (transactions/purchases), and a few others. Billions of events in short order.

**Learning Trajectory**

This is a great article from **master in information security, Russ McCree**. You’ll learn how Russ is using our new package for time series anomaly detection, anomalize, within his Blue Team (internal thread-defending team) work at Microsoft. He provides real-world examples of “threat hunting”, or the act of identifying malicious attacks on servers and how anomalize can help to *algorithmically* identify threats. Specifically, Russ shows you how to detect anomalies in security event logs as shown below.



**Threat Hunting With Anomalize**

By Russ McCree, Group Program Manager of Microsoft’s Windows and Devices Group

In reality, this is really a case of Deeper Functionality for Investigators in R within the original and paramount activity of [**D**igital **F**orensics/**I**ncident **R**esponse (DFIR)](https://en.wikipedia.org/wiki/Digital_forensics).

**Massive Data Requires Algorithmic Methods**

Those of us in the DFIR practice, and Blue Teaming at large, are overwhelmed by data and scale. **Success truly requires algorithmic methods**. If you’re not already invested here I have an **immediately applicable case study for you in tidy anomaly detection with anomalize**.

First, let me give credit where entirely due for the work that follows. Everything I discuss and provide is immediately derivative from *Business Science* ([@bizScienc](https://twitter.com/bizScienc)), specifically Matt Dancho ([@mdancho84](https://twitter.com/mdancho84)). When a client asked *Business Science* to build an open source anomaly detection algorithm that suited their needs, he created [anomalize](https://business-science.github.io/anomalize/):

“*a tidy anomaly detection algorithm that’s time-based (built on top of tibbletime) and scalable from one to many time series*,”

I’d say he responded beautifully.

**Anomalizing in InfoSec: Threat Hunting At Scale**

When his [blogpost introducing anomalize](http://www.business-science.io/code-tools/2018/04/08/introducing-anomalize.html) hit my (Russ’s) radar it lived as an open tab in my browser for more than a month until generating [this *toolsmith* article](https://holisticinfosec.blogspot.com/2018/06/toolsmith-133-anomaly-detection-threat.html). Please consider Matt’s post a mandatory read as step one of the process here. I’ll quote Matt specifically before shifting context:

“*Our client had a challenging problem: detecting anomalies in time series on daily or weekly data at scale. Anomalies indicate exceptional events, which could be increased web traffic in the marketing domain or a malfunctioning server in the IT domain. Regardless, it’s important to flag these unusual occurrences to ensure the business is running smoothly. One of the challenges was that the client deals with not one time series but thousands that need to be analyzed for these extreme events.*”

-Matt Dancho, creator of anomalize

**Key takeaway: Detecting anomalies in time series on daily or weekly data at scale. Anomalies indicate exceptional events.**

Now shift context with me to security-specific events and incidents, as they pertain to security monitoring, incident response, and threat hunting. In my [November 2017 post on redefining DFIR](https://holisticinfosec.blogspot.com/2017/11/toolsmith-129-dfir-redefined-deeper.html), I discussed **Time Series Regression (TSR) with the Holt-Winters method and a focus on seasonality and trends**. Unfortunately, I couldn’t share the code for how we applied TSR, but pointed out alternate methods, including **Seasonal and Trend Decomposition using Loess (STL)**, which:

* Handles any type of seasonality (which can change over time)
* Automatically smooths the trend-cycle (this can also be controlled by the user)
* Is robust to outliers (high leverage points that can shift the mean)

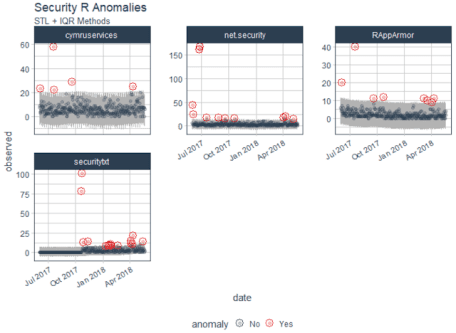
Here now, Matt has created a means to immediately apply the **STL method**, along with the **Twitter method** ([reference page](https://business-science.github.io/anomalize/reference/time_decompose.html)), as part of his time\_decompose() function, one of three functions specific to the anomalize package. The anomalize package includes the following main functions:

* time\_decompose(): Separates the time series into seasonal, trend, and remainder components. The methods used including **STL** and **Twitter**.
* anomalize(): Applies anomaly detection methods to the remainder component. The methods used including **IQR** and **GESD**.
* time\_recompose(): Calculates limits that separate the “normal” data from the anomalies.

I initially toyed with tweaking Matt’s demo to model downloads for security-specific R packages (yes, there are such things) from CRAN, including:

* [RAppArmor](https://cran.r-project.org/web/packages/RAppArmor/index.html)
* [net.security](https://cran.r-project.org/web/packages/net.security/index.html)
* [securitytxt](https://cran.r-project.org/web/packages/securitytxt/index.html)
* [cymruservices](https://cran.r-project.org/web/packages/cymruservices/index.html)

The latter two packages are courtesy of [Bob Rudis](https://twitter.com/hrbrmstr) of our beloved [Data-Driven Security: Analysis, Visualization and Dashboards](https://holisticinfosec.blogspot.com/2014/09/toolsmith-jay-and-bob-strike-back-data.html). Alas, this was a mere rip and replace, and really didn’t exhibit the use of anomalize in a deserving, varied, truly security-specific context. That said, I was able to generate immediate results doing so, as seen in **Figure 1: Security R Anomalies**.



As an initial experiment you can replace packages names with those of your choosing in [tidyverse\_cran\_downloads](https://business-science.github.io/anomalize/reference/tidyverse_cran_downloads.html), run it in R Studio.

**Code Tutorial: Anomalizing A Real Security Scenario**

I wanted to run anomalize against **a real security data scenario**, so I went back to the dataset from the original DFIR articles where I’d utilized counts of 4624 Event IDs per day, per user, on a given set of servers. As utilized originally, I’d represented results specific to only one device and user, but herein is the beauty of anomalize. We can achieve quick results across multiple times series (multiple systems/users). This premise is but one of many where time series analysis and seasonality can be applied to security data.

For those that would like to follow along, load the following libraries.

library(tidyverse)

library(tibbletime)

library(anomalize)

I originally tried to write log data from log.csv straight to an anomalize.R script with logs = read\_csv("log.csv") into a [tibble](https://tibble.tidyverse.org/index.html) (ready your troubles with tibbles jokes), which was not being parsed accurately, particularly time attributes. To correct this, from Matt’s GitHub I grabbed tidyverse\_cran\_downloads.R, and modified it as follows:

# Path to security log data

logs\_path <- "https://raw.githubusercontent.com/holisticinfosec/toolsmith\_R/master/anomalize/log.csv"

# Group by server, Convert to tibbletime

security\_access\_logs <- read\_csv(logs\_path) %>%

group\_by(server) %>%

as\_tbl\_time(date)

security\_access\_logs

## # A time tibble: 198 x 3

## # Index: date

## # Groups: server [3]

## date count server

##

## 1 2017-05-22 7 SERVER-549521

## 2 2017-05-23 9 SERVER-549521

## 3 2017-05-24 12 SERVER-549521

## 4 2017-05-25 4 SERVER-549521

## 5 2017-05-26 4 SERVER-549521

## 6 2017-05-30 2 SERVER-549521

## 7 2017-05-31 10 SERVER-549521

## 8 2017-06-01 14 SERVER-549521

## 9 2017-06-02 12 SERVER-549521

## 10 2017-06-05 7 SERVER-549521

## # ... with 188 more rows

This helped greatly thanks to the tibbletime package, which **“is an extension that allows for the creation of time aware tibbles”**. Some immediate advantages of this include: the ability to perform time-based subsetting on tibbles, quickly summarising and aggregating results by time periods. Guess what, Matt’s colleague, Davis Vaughan, is the one who wrote tibbletime too. 

I then followed Matt’s sequence as he posted on *Business Science*, but with my logs defined as a function in Security\_Access\_Logs\_Function.R. Following, I’ll give you the code snippets, as revised from Matt’s examples, followed by their respective results specific to processing my Event ID 4624 daily count log.

First, let’s summarize daily logon counts across three servers over four months.  
The result is evident in **Figure 2: Server Logon Counts**.

# plot login counts across 3 servers

security\_access\_logs %>%

ggplot(aes(date, count)) +

geom\_point(color = "#2c3e50", alpha = 0.5) +

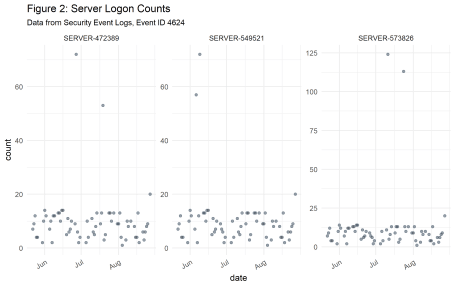
facet\_wrap(~ server, scale = "free\_y", ncol = 3) +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 30, hjust = 1)) +

labs(title = "Figure 2: Server Logon Counts",

subtitle = "Data from Security Event Logs, Event ID 4624")



Next, let’s determine which daily download logons are anomalous with Matt’s three main functions, time\_decompose(), anomalize(), and time\_recompose(), along with the visualization function, plot\_anomalies(), across the same three servers over four months. The result is revealed in **Figure 3: Security Event Log Anomalies**.

# Detect and plot security event log anomalies

security\_access\_logs %>%

# Data Manipulation / Anomaly Detection

time\_decompose(count, method = "stl") %>%

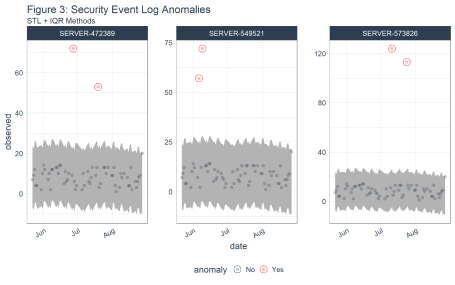
anomalize(remainder, method = "iqr") %>%

time\_recompose() %>%

# Anomaly Visualization

plot\_anomalies(time\_recomposed = TRUE, ncol = 3, alpha\_dots = 0.25) +

labs(title = "Figure 3: Security Event Log Anomalies", subtitle = "STL + IQR Methods")



Next, we can compare method combinations for the time\_decompose() and anomalize() methods:

* The (time\_decompose()) combined with the (anomalize()) method
* The (time\_decompose()) and (anomalize()) arguments

These are two different decomposition and anomaly detection approaches.

**Twitter + GESD**:

Following Matt’s method using Twitter’s AnomalyDetection package, combining time\_decompose(method = "twitter") with anomalize(method = "gesd"), while adjusting the trend = "4 months" to adjust median spans, we’ll focus only on SERVER-549521. In **Figure 4: SERVER-549521 Anomalies, Twitter + GESD**, you’ll note that there are anomalous logon counts on SERVER-549521 in June.

# Get only SERVER549521 access

SERVER549521 <- security\_access\_logs %>%

filter(server == "SERVER-549521") %>%

ungroup()

# Anomalize using Twitter + GESD

SERVER549521 %>%

# Twitter + GESD

time\_decompose(count, method = "twitter", trend = "4 months") %>%

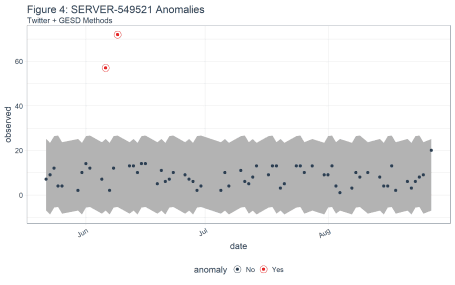
anomalize(remainder, method = "gesd") %>%

time\_recompose() %>%

# Anomaly Visualziation

plot\_anomalies(time\_recomposed = TRUE) +

labs(title = "Figure 4: SERVER-549521 Anomalies", subtitle = "Twitter + GESD Methods")



**STL + IQR**:

Again, we note anomalies in June, as seen in **Figure 5: STL + IQR Methods**. Obviously, the results are quite similar, as one would hope.

# STL + IQR

SERVER549521 %>%

# STL + IQR Anomaly Detection

time\_decompose(count, method = "stl", trend = "4 months") %>%

anomalize(remainder, method = "iqr") %>%

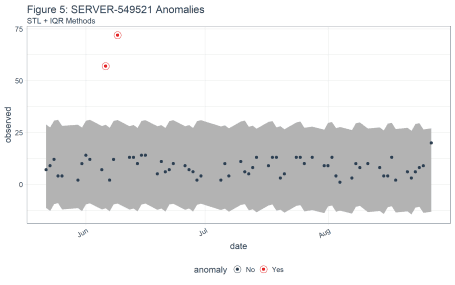
time\_recompose() %>%

# Anomaly Visualization

plot\_anomalies(time\_recomposed = TRUE) +

labs(title = "Figure 5: SERVER-549521 Anomalies",

subtitle = "STL + IQR Methods")



Finally, let use Matt’s plot\_anomaly\_decomposition() for visualizing the inner workings of how algorithm detects anomalies. The result is a four part visualization, including observed, season, trend, and remainder as seen in **Figure 6**.

# Created from Anomalize project, Matt Dancho

# https://github.com/business-science/anomalize

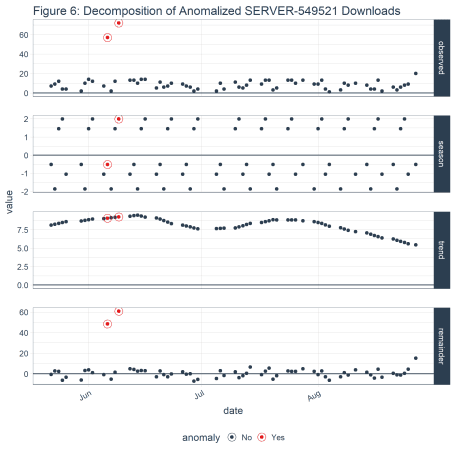
SERVER549521 %>%

time\_decompose(count) %>%

anomalize(remainder) %>%

plot\_anomaly\_decomposition() +

labs(title = "Figure 6: Decomposition of Anomalized SERVER-549521 Downloads")



**Future Work In InfoSec: Anomalize At A Larger Scale**

I’m really looking forward to putting these methods to use at a **much larger scale**, across a far broader event log dataset. I firmly assert that Blue Teams are already way behind in combating automated adversary tactics and problems of sheer scale, so…much…data. It’s only with tactics such as Matt’s anomalize package, and others of its ilk, that defenders can hope to succeed. *Business Science* is building a series of videos in addition, so keep an eye out there and on their GitHub for more great work that we can apply a blue team/defender’s context to.