The LogLED: An LED-Based Information Security Dashboard

Paul Ackerman
The LogLED
An LED-Based Information Security Dashboard

GIAC (GCIA) Gold Certification

Author: Paul Ackerman PAckerman@VacciNetLLC.com
Advisor: Manuel Santander

Accepted: August 2015

Abstract
Cybersecurity staff in many small/medium organizations often find themselves shorthanded. We juggle tasks on a daily basis from many areas of security including device configuration and review, managing firewalls, IPS, Web content filters and email security solutions, architecture design, code review, penetration testing, intrusion detection, incident response and more. When in a crunch for resources it is especially important to ensure that we focus on issues that are important and that we measure our effectiveness in that regard. With many staff receiving hundreds of emails per day and alarms from several different systems, a simple, visual dashboard to ensure attention is given to security-related issues when needed would be immensely helpful. This paper presents a fun and interactive "eye-candy" tool (an LED light strip) that an organization can build to display several key information security metrics that can be seen and interpreted in less than a second providing an instantaneous measure of the security defense condition of the organization.
1. Introduction

1.1. If An Alarm Goes Off And No One Is Listening...

Each year, Mandiant produces a detailed view of breach-related information security trends called the M-trends report. Contained in this report is a statistic regarding the average number of days that passed between when breaches occurred and when they were detected. Of breaches detected in 2014, it took an average of 205 days to detect the compromise from when it originally occurred. In one case, the attacker had compromised the network over eight years before it was detected (Mandiant, 2014). These reports have raised awareness about the challenges we face detecting and investigating malicious activity that results in a breach. I say detecting and investigating because sometimes detecting malicious activity isn't enough. In the case of the Target breach and likely many others, several systems did, in fact, detect malicious activity but responders appear to have failed to investigate fully the alarms (Schwartz, 2014).

One possible reason this occurred that seems quite probable in many small and medium-sized organizations might be a lack of focus. We cannot afford to ignore controls related to preventing a compromise, but we also must be careful not to spend all of our time checking and locking doors and windows because we don't realize when an intruder is already in the building. On the opposite end of the spectrum, if a window alarm goes off every day because of weak batteries and we neglect to change the batteries, we would desensitize ourselves to the alarm and would likely dismiss it during an actual attack. We need to focus and act on what's important as Jeffrey Roman, News Writer for the Information Security Media Group says, "The bottom line? Security professionals need to pay as much attention to breach detection as they do to breach prevention." (Roman, 2014)

1.2. The "Intrusion Detection Four-Step-Prep"

To improve detection capabilities, consider what the author calls the "intrusion detection four-step-prep." The first step organizations should take is to determine what the bad guys are after. In other words, what is the most important data or systems the...
organization has? It may sound obvious, but many small businesses with whom the author has worked have not had an immediate answer when asked this question. Financially motivated attackers may target information that has inherent economic value such as customer and employee PII, PCI cardholder data, and financial account numbers and credentials. Domestic or foreign competitors may be after an organization's intellectual property or trade secrets while terrorists or foreign nations may seek to gain access to industrial control systems used in energy, transportation or communications organizations or attack the availability of systems.

Once the crown jewels are identified, the second step is to learn what an intrusion looks like from a detection standpoint. Once the reader understands what to look for, he or she must build and implement instrumentation capable of detecting these indicators of compromise, which is the third step. In addition to building the instrumentation, organizations must carefully and painstakingly tweak and tune these detection systems including AV, DLP, IDP/IPS and SIEM to generate alarms on, and only on important events. A thorough understanding of TCP/IP, sniffing traffic and common protocols like HTTP, SMTP, DNS and ARP and how they are attacked is essential to be able to properly, configure and maintain an intrusion detection system.

Finally, analysts and incident handlers must understand the first three steps and take the time to investigate thoroughly the alarms that the monitoring tools are generating and continue adjusting the instrumentation to reduce false positives. That last bit can be difficult to accomplish consistently.

1.3. White, Black, Red, Blue, Green, Yellow Hats and Others

Let's assume the reader has navigated their way through the first three steps of the intrusion detection four-step-prep and has systems in place that are generating appropriate alarms of suspicious activity towards the identified critical assets. As mentioned in the fourth step, if no one is paying attention to the alarms and the organization fails to detect a breach, all that effort has been in vain. One reason that companies do not succeed at this step is resource limitations. The 2015 Global Information Security Workforce Study from ISC2 revealed that nearly two out of three organizations felt they had too few information security positions (Figure 1).

Paul Ackerman, PAckerman@VacciNetLLC.com
When asked which positions were lacking the most in the same report, 46% of respondents indicated that more security analysts are needed (Figure 2).

Upon reviewing this information, the author considered if organizations felt they needed more security analysts because they didn't have any or if they had one or more and felt they needed additional analysts and created a survey to provide insight into this question. The survey was taken by 70 GIAC Advisory Board members from organizations with five or fewer security positions and indicated that the security analyst was, in fact, the most common position with 54% of the respondents having at least one (Figure 3). Unfortunately, that also means that 46% don't have one!

Paul Ackerman, PAckerman@VacciNetLLC.com
In fact, 13% of respondents didn't have any full-time security positions at all in their organization and another 23% only had one. The survey also revealed that the security department in many organizations consists of only one or two security analysts and managers. In fact, only 3 of the 70 respondents have an incident handler position. These statistics seems to indicate that security analysts are expected to perform the tasks of intrusion detection and incident handling. So it seems security analysts are both the most common and most over-burdened position in the security industry.

To illustrate this further, consider that Information Security Analysts don't sit around all day twiddling their thumbs just waiting for an intrusion to detect. A brief review of a dozen job postings on dice.com for security analysts contain more than seventeen different categories of job duties. Perhaps not surprisingly, security analysts are expected to work on policies and procedures as often as monitoring the network.
Consulting and project management duties show up just as often in position descriptions as incident response. It's no wonder that it takes 205 days on average for organizations to detect a breach (Mandiant, 2014).

Figure 5: Security Analyst Job Duties

Given all the evidence that security analysts are commonly expected to detect and respond to incidents as a fraction of their job duties, it leads to questioning what percentage of analysts have ended up ignoring or placing an alarm on the back burner due to some other job function taking priority. Something is needed to make critical incidents stand out in the day to day monotony of the job to ensure that analysts are addressing issues promptly.

2. The Log Light

Paul Ackerman, PAckerman@VacciNetLLC.com
For people on the go, a visual representation of critical information that is very quick and easy to consume could help bring attention to issues and focus resources as needed. The log light is not a candle fashioned from a tree trunk (Figure 6). It is an LED strip controlled by a Teensy 2.0++ microcontroller that has been programmed to light up individual LEDs based on input that is provided from a serial connection to a PC. The serial interface gives the reader the ability to control the LED strip from a PC and convey any message the reader desires. In this project, the PC runs a PowerShell script that retrieves data from a SIEM API, parses new alarm data, determines which color each LED should be and then sends the data to the Teensy over the serial port. For this project, the strip displays critical alarms in red, high severity alarms in orange, medium alarms in blue and low or informational alarms in green.

Web-based dashboards are certainly useful, but they take some time to read and interpret. On the contrary, upon entering a room where the LogLED is displayed as it is in figure 11 conveys a status that is immediately apparent and significant. In figure 7 above, one can immediately discern that in this case, there are quite a few unacknowledged low and medium alarms as well as a couple high and critical alarms that need to be handled. The remainder of this paper will describe how to build the strip shown in Figure 7 for the reader's organization.
This paper will only cover integration with one SIEM solution, but the author designed the project in such a way that the reader can easily display any data that can be accessed from within PowerShell. The reader can simply alter the PowerShell script to retrieve data from any API, flat file or other source desired, and the rest of the code remains the same without any need to reprogram the Teensy. The result is an immediate situational awareness that can help draw an analysts' focus and ensure that important events and incidents are not being ignored.

### 2.1. The LED Strip

The LED strip is an LPD8806 and is found on adafruit.com for around $30 plus shipping. It is a 5-volt strip that comes in various lengths. However, this project specifically uses the 32 LED strip that is roughly one meter in length. If the reader chooses another length, the code supplied in the appendices will need to be modified to account for the number of LED's present. This particular strip has four leads: red, white, green and blue which are soldered to the Teensy. The red wire is +5V and should be connected to the +5V pin of the teensy to use the PC's USB port or a 5V 2A power supply to power the strip. A USB 3.0 port can supply a maximum of 900ma, and a USB 2.0 port can supply a maximum of 500ma. The teensy 2.0++ itself consumes up to 60.2ma leaving 440ma from a USB 2.0 port to power the strip. In testing, this 440ma produces such bright light that the color values used in the script were later reduced in order to dim the lights. In other words, an additional power supply should not be needed. However, if maximum brightness is desired for perhaps an outdoor use, a separate 5V 2A power supply could be used. The one-meter strip has a maximum current draw of 1.92amps at 5V.

---

Paul Ackerman, PAckerman@VacciNetLLC.com
The white lead on the strip is the ground and should connect to the ground pin of the teensy. If the reader is using a separate power supply, the white lead must be connected to both the Teensy's ground pin and the ground of the power supply. In either case, the Teensy's ground pin must be connected to the strip's white wire.

The green wire on the strip is the data and should connect to PIN D2 on the Teensy and the blue wire on the strip is the clock and should be connected to PIN D1 on the Teensy as shown in Figure 9. The reader is welcome to use other pins on the Teensy if desired, but the code on the Teensy will need to be updated to reflect which pins are data and clock. This is done by modifying the following two lines at the top of the Teensy code:

```c
int dataPin = 2;
int clockPin = 1;
```

Lastly, the strip has an input and output side. As LEDs are as the name suggests, diodes, current only flows in one direction so be sure to connect the "I" (Input) side of the strip to the teensy as shown in figure 10 rather than the "O" (Output) side.

![Connection Diagram]

**Figure 10: LPD8806 Input vs Output**

### 2.2. The Teensy 2.0++ Microcontroller

The Teensy 2.0++ is a very small USB development board similar to Arduino boards and will be used to control the LED strip. To program the Teensy, one needs only the Arduino IDE and the Teensyduino add-on. Teensy's are compatible with many Arduino libraries, and several libraries will also need to be downloaded. The Teensy was chosen for the project simply because the author had one lying around and because the Teensy 2.0 is 5V that matches the LED strip making things a little simplier. Projects in the
Teensy world are written in C and are called "sketches." The link to the sketch for this project is provided in the Appendix. To program the Teensy, perform the following steps:

- Download and install the Arduino IDE
- Download and install Teensyduino add-on
- Download the Windows Serial Installer
- Download the LogLED sketch from Github
- Download the Time.h and LPD8806.h and .cpp files from the sources listed in the code in comments. If the sources change later, the Github repository will be updated to reflect the new location.

Once you have installed the required software, you should be able to open the Arduino IDE, load the LogLED sketch and program the Teensy by clicking the check mark in the top left corner of the IDE. The first time you program the Teensy, you may need to press the button on the Teensy for it to enter programming mode. The Teensy will reboot after programming and immediately begin executing the code that was just loaded.

The sketch contains code that reads from the serial port and displays the contents accordingly. It is meant to consume data provided on the serial port by the PowerShell script. In addition to simply displaying the red, orange, green, blue values sent for each LED, several pre-defined effects have been loaded on the teensy from the LEDBeltKit sketch by PaintYourDragon (PaintYourDragon, 2014) in addition to one effect called emergency that was written by the author. These effects can be called from within PowerShell as desired. Effects can be used to draw attention to the strip when something significant happens. For example, if pulling alarm data, one could make the strip display an effect to indicate that the number of alarms is trending upward. Additional effects may be written by the reader and added to the sketch as desired.

After programming the Teensy, upon booting, the Teensy executes a loop that reads data from the serial port and displays it. It looks for data that begins with the "@" and then takes 96 required arguments and two optional arguments. The "@" sign is there to distinguish random data that might be read from the port from real data intentionally being sent. Surely, 98 arguments seems like a lot but don't fret. The first 96 arguments are simply a red, green and blue decimal value (in that order) from 0-127 for each of the 32 LED's. The LED strip has separate red, green and blue LEDs that you illuminate at

Paul Ackerman, PAckerman@VacciNetLLC.com
different intensities. The values 0-127 represent the intensity of light for that color with 0 being none or off and 127 being the maximum. The 97th argument is a value for an LED that should blink. This value should range from 0 to 32 being 32 if none of the LEDs should blink. The 98th argument is used to display an effect. The values are 0-9 and perform the following effects:

0 - no effect
1 - Color Chase
2 - Color Fill
3 - Sparkles
4 - Christmas Sparkles
5 - KITT
6 - Candy Cane
7 - Icy
8 - Rainbow
9 - Emergency

If the string includes an effect, the effect is displayed first. Once the effect is finished, the Teensy will then display the colors supplied in the first 96 parameters for each LED. Parameters are separated by comma's, and the last parameter must also be followed by a terminating comma. Here is an example of a complete string that the Teensy can interpret:

```
@0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0,127
0,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,
127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,
127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127,0,127
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,32,1,
```

In the string above, the last parameter is a 1 which means the strip will perform the color chase effect that lasts a few seconds. Once the effect is complete, the first LED (0) on the strip would be lit using a value of 0 for red, 127 for green and 0 for blue resulting in a bright green color since these are the first three parameters supplied in the string. The next eleven LED's are also green. LEDs 12-22 have values: red=127, green=127, blue=0 which result in a bright yellow color. These are highlighted in yellow in the string above. The next two LEDs (23 and 24) have values red=127, green=35, blue=0 which yields

Paul Ackerman, PAckerman@VacciNetLLC.com
orange and LED 25 has values red=127, green=0, blue=0 resulting in bright red. The remaining LED's are provided values of 0 meaning the LED's are off. Valid red, green and blue values must be provided for each of the 32 LEDs on the strip. The 97th parameter has value 32 which means none of the LEDs will blink. When the Teensy is sent the string in the example above, the result is the display shown in Figure 7.

2.3. The LogRhythm API and PowerShell Script

At this point, the reader should be able to light up the Teensy with static data sent over the serial port with a very basic PowerShell script. However, displaying dynamic data from the SIEM solution was the goal for this project. A SIEM is a great source of information to display on the LED strip. Other sources of information could be central Anti-Virus servers or Vulnerability management systems. The script provided in this paper displays the current alarm data from LogRhythm, a SIEM solution. The script was written for the API available in version 6.2.5 of LogRhythm. Slight modifications may be necessary to support other versions, but it is presented as an example of how to work with the LED strip to produce meaningful results.

The script assumes the LogRhythm API is configured for Windows authentication and that PowerShell 4.0 is running in the context of a user that has permission to use the API. The script calls the GetFirstPageAlarmsByAlarmStatus method of the API to obtain an array of new alarms that have been generated by the SIEM. Alarms in LogRhythm have risk-based priorities that are a numerical score from 0 to 100 that indicate the severity of the alarm. The higher, the more severe. The goal is to use color to reflect the severity of new alarms displayed on the strip. After getting current alarm data, the script determines how many green, blue, orange and red LED's to illuminate by comparing the risk-based priority (RBP) to pre-determined values based on the following table:

- Green: RBP <60
- Blue: RBP 60 - 69
- Orange: RBP 70-79
- Red: RBP >=80

The reader is, of course, free to modify these values based on their RBP scores and risk thresholds. Once the number of LED's of each color has been determined, the script uses
that information to build a string in the proper format to send to the Teensy and finally sends the data to the serial port.

If the script cannot communicate with the API or the API returns an error, the script will send the following string to the Teensy:

```
"@40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,40,0,40,0,0,,"
```

This string will turn the entire strip purple since there are equal parts red and blue. Since purple is not normally used, it is immediately obvious when there is a communication problem between the PowerShell script and the LogRhythm API. The script is written in such a way that as soon as connectivity is re-established, the script will continue reading and displaying alarm data just as it had before the error without any user intervention required.

### 3. Implementation, Limitations, and Other Ideas

In summary, to build your own LogLED, take the following action.

1. Purchase the LPD8806 LED strip from adafruit.
2. Purchase a Teensy 2.0
3. Solder the strip and teensy together as described in section 2.1
4. Upload the sketch provided in the appendix to the teensy.
5. Setup LogRhythm or your SIEM's API
6. Connect the teensy to a PC and execute the provided PowerShell script.
7. Voila! You have your very own LogLED!

---

Paul Ackerman, PAckerman@VacciNetLLC.com
One limitation of the LogLED script is that it only displays 32 alarms at a time. There are certainly other ways to display more data but since the goal is to draw attention to unacknowledged alarms, it wasn't necessary to pursue ways to make the strip display larger numbers of alarms. If there are more than 32 alarms, the top LED will blink slowly to indicate that not all alarms are being displayed. Another possibility for the light still using LogRhythm data would be to display the current log volume as a percentage compared to the last hour in addition to alarm data. For example, if the average number of logs/second received is 1000 over the last hour, then a current value of 1000 logs/s could illuminate, perhaps 75% of the strip. This way, if the log volume suddenly increases, more of the strip could be illuminated, or if log volume suddenly drops, more of the strip would be dark. The number of LED’s lit up would create an immediate understanding of current log volume. The colors of the lights could still represent alarm severity, but it could no longer be a one-for-one representation of each alarm. Instead, percentages could be used. So, if log volume were normal, then 75% of the strip or 24 LEDs will be lit up. Consider the case where there are 14 alarms at that time with 2 being critical, 3 being high, 4 being medium and 6 being low. 2 of 14 is 14.28% or 3 LEDs. 3 of 14 is 21% or 7 LEDs. 4 of 14 is 28.57% or 7 LEDs, and 6 of 14 is 42.8% or 9 LEDs. If there is a remaining LED, it could be lit with the color with the highest remainder during the rounding process. Since there were 14 alarms, the 14th LED could be made to blink to indicate the current number of alarms. The result would be an easily understood visual indication of the amount of log activity and both the number and severity of current alarms. The LogRhythm API in version 6.2.5 does not offer the logs per second statistic. However, this figure is available via a performance monitor and thus could be written to file and should be able to be read by PowerShell instead of using the API.

Another idea might be to display an overall risk score from a vulnerability scanner like Saint, Nexpose or Nessus using the quantity of LED’s illuminated and the color to represent the number of critical, high, medium and low vulnerabilities currently in the environment. Or perhaps integration with an Anti-Virus product to display the number of threats detected in the environment in a period such as the last 24 hours. Threats that were removed successfully could be displayed in green while threats that

Paul Ackerman, PAckerman@VacciNetLLC.com
were not successfully removed or quarantined could be displayed in red. A sudden increase of threats would be easy to spot in this case.

Finally, one could potentially rotate through several different metrics, displaying each of the three discussed above by using different colors for each. For instance, handled Anti-Virus threats might use the color white and teal or purple for unhandled threats. This way, when viewing the LOGLed, the colors used could tell the user which metric is currently being displayed.

4. Conclusion

Shiny lights are great eye candy for your SOC and for a total cost of less than $60 in parts the oohs and aahs create a great return on investment. What this paper is all about, however, is not the eye candy, but the utility of bringing awareness to issues that must be handled promptly. What the author has only eluded to up till now, but is hopefully apparent, is that to use the LED strip, the reader must first decide what the most meaningful data is that the strip could display. Then, instrumentation must be built to produce that information. This process is, in essence, the true goal of this paper; to use a cool gadget to entice the reader into taking a serious and enlightened look at information security metrics as they relate to intrusion detection. This in turn should increase the awareness and responsiveness to the alarms being generated which should help reduce the average number of days that pass between compromise and detection. To get started, SANS SEC503 Intrusion Detection in-depth will help the reader with identifying indicators of compromise for high-risk data and assets and with building and properly configuring IDS systems and other instrumentation to detect these events. These are the second and third steps of the intrusion detection four step prep. Even a shiny LED on the wall can't force analysts to perform the fourth step in the intrusion detection four-step-prep however - to thoroughly investigate alarms and to continuously tune and tweak detection systems to ensure that we are affording ourselves the opportunity to investigate thoroughly generated alarms.
5. References


6. Appendix A
The Teensy Sketch and the PowerShell script are both provided here for completeness, however they both exist on GitHub so that they may be updated if any errors are found or with additional functionality as the project progresses. Please check the GitHub site for newer versions. The GitHub link is:

https://github.com/VacciNet/LogLED

6.1 The LogLED.ino Sketch:

```c
#include "LPD8806.h"
#include "SPI.h"
#include <Time.h>
#include <string.h>

// Example to control LPD8806-based RGB LED Modules in a strip!
// NOTE: WILL NOT WORK ON TRINKET OR GEMMA due to floating-point math
**************************************************************************/

int dataPin = 2;
int clockPin = 1;
// Set the first variable to the NUMBER of pixels. 32 = 32 pixels in a row
// The LED strips are 32 LEDs per meter but you can extend/cut the strip
LPD8806 strip = LPD8806(32, dataPin, clockPin);

int ledPin = 11;
int ExternalLED = 20; //connect your test LED to pin 20 on Teensy
int ModeZ;
int PSCmd;
char c;
time_t pctime;
boolean TimeisSet;
long SecsStart, SecsEnd, SecsElapsed;

// LED function prototypes, do not remove these!
void colorChase(uint32_t c, uint8_t wait);
void colorWipe(uint32_t c, uint8_t wait);
void dither(uint32_t c, uint8_t wait);
void scanner(uint8_t r, uint8_t g, uint8_t b, uint8_t wait);
void wave(uint32_t c, int cycles, uint8_t wait);
void rainbowCycle(uint8_t wait);
uint32_t Wheel(uint16_t WheelPos);

void setup()
{
  // Start up the LED strip
  strip.begin();

  // Update the strip, to start they are all 'off'
  strip.show();

digitalWrite(ledPin, HIGH); // set the onboard LED on
Serial.begin(9600);
pinMode(ExternalLED, OUTPUT);
delay(2000);
}
```

Paul Ackerman, PAckerman@VacciNetLLC.com
bool Flash = 0;
uint8_t flashr = 0;
uint8_t flashg = 0;
uint8_t flashb = 0;
uint8_t flashled = 0;

void loop()
{
    uint8_t* ModeZ;

    int codePosition = 2;
    if (ModeZ[1] == 6)
    {
        if (ModeZ[99] > 0)
        {
            if (ModeZ[99] == 9)
            {
                Serial.println(F("*** Teensy functions programmed on this board ***"));
                Serial.println(F("1 - Chase"));
                Serial.println(F("2 - Fill"));
                Serial.println(F("3 - Sparkles"));
                Serial.println(F("4 - Christmas Sparkles"));
                Serial.println(F("5 - Cops"));
                Serial.println(F("6 - CandyCane"));
                Serial.println(F("7 - Icy"));
                Serial.println(F("8 - Rainbow"));
                Serial.println(F("9 - This Menu"));
                Serial.println(F("@ - Send RGB codes for each led plus index to blink"));
            }
            if (ModeZ[99] == 1)
            {
                Serial.println(F("Color Chase"));
                clear();
                colorChase(strip.Color(127,127,127), 20); // white
                colorChase(strip.Color(127,0,0), 20); // red
                colorChase(strip.Color(127,127,0), 20); // yellow
                colorChase(strip.Color(0,127,0), 20); // green
                colorChase(strip.Color(0,127,127), 20); // cyan
                colorChase(strip.Color(0,0,127), 20); // blue
                colorChase(strip.Color(127,0,127), 20); // magenta
            }
            if (ModeZ[99] == 2)
            {
                Serial.println(F("Color Fill"));
                clear();
                colorWipe(strip.Color(127,127,127), 20); // white
                colorWipe(strip.Color(127,0,0), 20); // red
                colorWipe(strip.Color(127,127,0), 20); // yellow
                colorWipe(strip.Color(0,127,0), 20); // green
                colorWipe(strip.Color(0,127,127), 20); // cyan
                colorWipe(strip.Color(0,0,127), 20); // blue
                colorWipe(strip.Color(127,0,127), 20); // magenta
            }
            if (ModeZ[99] == 3)
            {
                Serial.println(F("Color Sparkles-cyan-slow/black-fast/magenta-slow/black-fast/yellow-slow/black-fast"));
                // Color sparkles
                clear();
                dither(strip.Color(0,127,127), 50); // cyan, slow
                dither(strip.Color(0,0,0), 15); // black, fast
dither(strip.Color(127,0,127), 50);       // magenta, slow
dither(strip.Color(0,0,0), 15);           // black, fast
dither(strip.Color(127,127,0), 50);       // yellow, slow
dither(strip.Color(0,0,0), 15);           // black, fast
}

if (ModeZ[99] == 4)
{
    Serial.println(F("Christmas Sparkles-green-slow/red-slow/green-slow/red-fast");
clear();
    // Color sparkles
dither(strip.Color(0,127,0), 50);           // green, slow
dither(strip.Color(127,0,0), 50);           // red, slow
dither(strip.Color(0,127,0), 15);           // green, fast
dither(strip.Color(127,0,0), 15);           // red, fast
}

if (ModeZ[99] == 5)
{
    Serial.println(F("Back and Forth Lights"));
clear();
    // Back-and-forth lights
    for (int i = 30; i > 0; i =i - 15)
    {
        scanner(127,0,0, i);        // red, slow
        scanner(0,0,127, i);        // blue, fast
    }
}

if (ModeZ[99] == 6)
{
    Serial.println(F("Candy Cane");
clear();
    // Wavy ripple effects
    wave(strip.Color(127,0,0), 4, 20);        // candy cane
}

if (ModeZ[99] == 7)
{
    Serial.println(F("Icy"));
clear();
    // Wavy ripple effects
    wave(strip.Color(0,0,100), 1, 40);        // icy
}

if (ModeZ[99] == 8)
{
    Serial.println(F("Rainbow"));
clear();
    // make a rainbow.
rainbowCycle(0);  // make it go through the cycle fairly fast
}

if (ModeZ[99] == 9)
{
    Serial.println(F("Emergency"));
clear();
    // Danger!
cops(50);  // 50ms delay
}

ModeZ[99] = 0;
}

ModeZ[1] = 0;
//Flash the provided LED
flashled = ModeZ[98];
for (int i = 0; i < strip.numPixels() + 1; i++)
{
    strip.setPixelColor(i, 0);
strip.show();             // refresh strip display
}
for (int i = 0; i < strip.numPixels() + 1; i++)
{
    strip.setPixelColor(i, ModeZ[codePosition], ModeZ[codePosition + 1],
    ModeZ[codePosition + 2]);
    strip.show();             // refresh strip display
    delay(5);                // hold image for a moment
    codePosition = codePosition + 3;
}

if (Flash == 0)
{
    uint8_t flashstartpos = ((flashled -1) * 3) + 2;
    flashr = ModeZ[flashstartpos];
    flashg = ModeZ[flashstartpos + 1];
    flashb = ModeZ[flashstartpos + 2];
    strip.show();    // refresh strip display
    delay(50);
    Flash = 1;
} else
{
    strip.setPixelColor(flashled, flashr, flashg, flashb);
    strip.show();
    delay(50);
    Flash = 0;
}

//----------------------------------------------------------------------------
uint8_t* GetMode()
{
    //clear read buffer
    while (Serial.available()){ Serial.read();}

    //attempt to read serial port for 2 seconds
    SecsStart = now();
    uint8_t codes[390];
    uint8_t flash;
    do
    {
        if (Serial.available())
        {
            c = Serial.read();
            if (c == '8')
            {
                int d = 1;
                int code = 0;
                codes[1] = 6;
                for (int i = 1; i < 391; i++)
                {
                    c = Serial.read();
                    int ic = c - '0';
                    if (!(c == ','))
                        code = (10 * code) + ic;
                    else
                    {
                        d++;
                        codes[d] = code;
                        code = 0;
                    }
                }
            }
            return codes;
        }
    }
    return NULL;
}
void clear() {
    // Clear strip data before start of next effect
    for (int i=0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0);
    }
}

void cops(int dly) {
    int quarter = strip.numPixels() / 4;
    // Triple flash red and blue in this pattern
    // RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
    // ........BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
    for (int s = 0; s < 4; s++) {
        for (int r = 0; r < 3; r++){
            for (int i = 0; i < (quarter); i++){
                strip.setPixelColor(i, 127,0,0);
            }
            for (int i = (quarter * 2); i < (quarter * 3); i++){
                strip.setPixelColor(i, 127,0,0);
            }
            strip.show();
            delay(dly);
            for (int i = 0; i < strip.numPixels(); i++){
                strip.setPixelColor(i, 0, 0,0);
            }
            strip.show();
            delay(dly);
        }
    }
    for (int r = 0; r < 3; r++){
        for (int i = quarter; i < (quarter * 2); i++){
            strip.setPixelColor(i, 0, 0,127);
        }
        for (int i = (quarter * 3); i < strip.numPixels(); i++){
            strip.setPixelColor(i, 0, 0,127);
        }
        strip.show();
        delay(dly);
        for (int i = 0; i < strip.numPixels(); i++){
            strip.setPixelColor(i, 0, 0,0);
        }
        strip.show();
        delay(dly);
    }
    // RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR strobe 3 times
    // BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB strobe 3 times
    // Repeat 2 more times - half speed
    for (int s = 0; s < 3; s++){
        for (int r = 0; r < 3; r++){
The LogLED
An LED-Based Information Security Dashboard

Paul Ackerman, PAckerman@VacciNetLLC.com

```cpp
for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 127, 0, 0);
}
strip.show();
delay(dly * 2);
for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0, 0, 0);
}
strip.show();
delay(dly * 2);

for (int r = 0; r < 3; r++) {
    for (int i = 0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0, 0, 127);
    }
    strip.show();
    delay(dly * 2);
    for (int i = 0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0, 0, 0);
    }
    strip.show();
    delay(dly * 2);
}
// Flash red and blue back and forth in this pattern
// RRRRRRRRRRRRRRRR............
// ................BBBBBBBBBBBBBB
// Then switch
// BBBBBBBBBBBBBBBBB.............
// ................RRRRRRRRRRRRRRR
for (int r = 0; r < 4; r++) {
    for (int i = 0; i < (quarter * 2); i++) {
        strip.setPixelColor(i, 127, 0, 0);
    }
    for (int i = (quarter * 2); i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0, 0, 0);
    }
    strip.show();
    delay(dly * 2);
    for (int i = 0; i < (quarter * 2); i++) {
        strip.setPixelColor(i, 0, 0, 0);
    }
    for (int i = (quarter * 2); i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0, 0, 127);
    }
    strip.show();
    delay(dly * 2);
}
for (int r = 0; r < 6; r++) {
    for (int i = 0; i < (quarter * 2); i++) {
        strip.setPixelColor(i, 0, 0, 127);
    }
    for (int i = (quarter * 2); i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0, 0, 0);
    }
    strip.show();
    delay(dly * 2);
    for (int i = 0; i < (quarter * 2); i++) {
        strip.setPixelColor(i, 0, 0, 0);
    }
    for (int i = (quarter * 2); i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 127, 0, 0);
    }
    strip.show();
    delay(dly * 2);
}
```
// Double flash red and blue in this pattern
// RR..RR..RR..RR..RR..RR..RR..RR.. strobe 3 times
// ..BB..BB..BB..BB..BB..BB..BB..BB strobe 3 times
// Repeat 3 times
for (int s = 0; s < 4; s++) {
  for (int r = 0; r < 3; r++){
    for (int i = 0; i < strip.numPixels(); i = i + 4){
      strip.setPixelColor(i, 127,0,0);
      strip.setPixelColor(i + 1, 127,0,0);
    }
    strip.show();
    delay(dly * 1.5);
    for (int i = 0; i < strip.numPixels(); i = i + 4){
      strip.setPixelColor(i, 0, 0,0);
      strip.setPixelColor(i + 1, 0, 0,0);
    }
    strip.show();
    delay(dly);
  }
  for (int r = 0; r < 3; r++){
    for (int i = 2; i < strip.numPixels(); i = i + 4){
      strip.setPixelColor(i, 0, 0,127);
      strip.setPixelColor(i + 1, 0, 0,127);
    }
    strip.show();
    delay(dly * 1.5);
    for (int i = 2; i < strip.numPixels(); i = i + 4){
      strip.setPixelColor(i, 0, 0,0);
      strip.setPixelColor(i + 1, 0, 0,0);
    }
    strip.show();
    delay(dly);
  }
}

// Use whole strip switching back and forth red to blue
// first solid then strobing more and more
// RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
// BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
// RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
// BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
// Quarter speed
for (int r = 0; r < 2; r++) {
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 127, 0, 0);
  }
  strip.show();
  delay(dly * 4);
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0, 0, 127);
  }
  strip.show();
  delay(dly * 4);
}

// RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR strobe 6 times
// BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB strobe 6 times
// Repeat 1 once - full speed
for (int s = 0; s < 2; s++){
  for (int r = 0; r < 6; r++){
    for (int i = 0; i < strip.numPixels(); i++){
      strip.setPixelColor(i, 127,0,0);
    }
    strip.show();
    delay(dly);
  }
  for (int i = 0; i < strip.numPixels(); i++){
    strip.setPixelColor(i, 0, 0, 0);
  }
}
```cpp
void setup() {
  Serial.begin(9600);
  // set pin mode
  pinMode(strip.numPixels(), OUTPUT);
  // update LED pixels
  for (int r = 0; r < 6; r++){
    for (int i = 0; i < strip.numPixels(); i++){
      strip.setPixelColor(i, 0, 0, 127);
    }
    strip.show();
    delay(dly);
    for (int i = 0; i < strip.numPixels(); i++){
      strip.setPixelColor(i, 0, 0, 0);
    }
    strip.show();
    delay(dly);
  }
}

void loop() {
  // RRRRRRR.................BBBBBBBBB strobe 3 times
  // ........RRRRRRRRBBBBBBBB....... strobe 3 times
  // repeat 4 times full speed
  for (int s = 0; s < 4; s++){
    for (int r = 0; r < 3; r++){
      for (int i = 0; i < quarter; i++){
        strip.setPixelColor(i, 127,0,0);
      }
      for (int i = (quarter * 3); i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0,0,127);
      }
      strip.show();
      delay(dly * 1.5);
      for (int i = 0; i < strip.numPixels(); i++){
        strip.setPixelColor(i, 0, 0, 0);
      }
      strip.show();
      delay(dly);
    }
    for (int r = 0; r < 3; r++){
      for (int i = quarter; i < (quarter * 2); i++) {
        strip.setPixelColor(i, 127,0,0);
      }
      for (int i = (quarter * 2); i < (quarter * 3); i++) {
        strip.setPixelColor(i, 0,0,127);
      }
      strip.show();
      delay(dly * 1.5);
      for (int i = 0; i < strip.numPixels(); i++){
        strip.setPixelColor(i, 0, 0, 0);
      }
      strip.show();
      delay(dly);
    }
  }
}
```

Paul Ackerman, PAckerman@VacciNetLLC.com
The LogLED
An LED-Based Information Security Dashboard

25

Paul Ackerman, PAckerman@VacciNetLLC.com

int quarter = strip.numPixels() / 4;

for (int i = 0; i < quarter; i++) {
  strip.setPixelColor(i, 0, 0, 127);
  strip.show();
  delay(dly*1.5);
}

for (int i = quarter; i < (quarter * 3); i++) {
  strip.setPixelColor(i, 127, 127, 127);
  strip.show();
  delay(dly *1.5);
}

for (int i = 0; i < quarter; i++) {
  strip.setPixelColor(i, 127, 0, 0);
  strip.show();
  delay(dly*1.5);
}

for (int i = (quarter * 3); i < strip.numPixels(); i++) {
  strip.setPixelColor(i, 0, 0, 127);
  strip.show();
  delay(dly*1.5);
}

for (int i = quarter; i < (quarter * 3); i++) {
  strip.setPixelColor(i, 127, 127, 127);
  strip.show();
  delay(dly*1.5);
}

for (int i = 0; i < quarter; i++) {
  strip.setPixelColor(i, 127, 0, 0);
  strip.show();
  delay(dly*1.5);
}

for (int i = quarter; i < (quarter * 3); i++) {
  strip.setPixelColor(i, 127, 0, 0);
  strip.show();
  delay(dly*1.5);
}

for (int i = 0; i < strip.numPixels(); i++) {
  strip.setPixelColor(i, 0, 0, 0);
  strip.show();
  delay(dly);
}

for (int r = 0; r < 3; r++){
  for (int i = quarter; i < (quarter * 3); i++){
    strip.setPixelColor(i, 127, 127, 127);
    strip.show();
    delay(dly *1.5);
  }
  for (int i = quarter; i < (quarter * 3); i++){
    strip.setPixelColor(i, 127, 127, 127);
    strip.show();
    delay(dly*1.5);
  }
  for (int i = 0; i < quarter; i++) {
    strip.setPixelColor(i, 127, 0, 0);
    strip.show();
    delay(dly*1.5);
  }
  for (int i = (quarter * 3); i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0, 0, 127);
    strip.show();
    delay(dly*1.5);
  }
}

for (int r = 0; r < 3; r++){
  for (int i = 0; i < quarter; i++) {
    strip.setPixelColor(i, 0, 0, 127);
    strip.show();
    delay(dly*1.5);
  }
  for (int i = 0; i < quarter; i++) {
    strip.setPixelColor(i, 0, 0, 127);
    strip.show();
    delay(dly*1.5);
  }
  for (int i = (quarter * 3); i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 127, 0, 0);
    strip.show();
    delay(dly*1.5);
  }
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, 0, 0, 0);
    strip.show();
    delay(dly);
  }
}
for (int r = 0; r < 3; r++){
  for (int i = quarter; i < (quarter * 3); i++){
    strip.setPixelColor(i, 127,127,127);
  }
  strip.show();
  delay(dly *1.5);
  for (int i = 0; i < strip.numPixels(); i++){
    strip.setPixelColor(i, 0, 0, 0);
  }
  strip.show();
  delay(dly);
}

for (int r = 0; r < 3; r++){
  for (int i = 0; i < quarter; i++){
    strip.setPixelColor(i, 127,127,127);
  }
  for (int i = (quarter * 3); i < strip.numPixels(); i++){
    strip.setPixelColor(i, 127,127,127);
  }
  strip.show();
  delay(dly *1.5);
  for (int i = 0; i < strip.numPixels(); i++){
    strip.setPixelColor(i, 0, 0, 0);
  }
  strip.show();
  delay(dly);
}

for (int r = 0; r < 3; r++){
  for (int i = quarter; i < (quarter * 3); i++){
    strip.setPixelColor(i, 127,0,0);
  }
  strip.show();
  delay(dly *1.5);
  for (int i = 0; i < strip.numPixels(); i++){
    strip.setPixelColor(i, 0, 0, 0);
  }
  strip.show();
  delay(dly);
}

// Cycle through the color wheel, equally spaced around the belt
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;
  for (j=0; j < 384 * 5; j++) {     // 5 cycles of all 384 colors in the wheel
    for (i=0; i < strip.numPixels(); i++) {
      // tricky math! we use each pixel as a fraction of the full 384-color
      // wheel (thats the i / strip.numPixels() part)
      // Then add in j which makes the colors go around per pixel
      // the % 384 is to make the wheel cycle around
      strip.setPixelColor(i, Wheel(((i * 384 / strip.numPixels()) + j) % 384));
    }
    strip.show();   // write all the pixels out
    delay(wait);
  }
} 
// fill the dots one after the other with said color
The LogLED
An LED-Based Information Security Dashboard

Paul Ackerman, PAckerman@VacciNetLLC.com

// good for testing purposes
void colorWipe(uint32_t c, uint8_t wait) {
    int i;
    for (i=0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, c);
        strip.show();
    }
}

// Chase a dot down the strip
void colorChase(uint32_t c, uint8_t wait) {
    int i;
    for (i=0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, 0); // turn all pixels off
    }
    for (i=0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, c); // set one pixel
        strip.show(); // refresh strip display
        delay(wait); // hold image for a moment
        strip.setPixelColor(i, 0); // erase pixel (but don't refresh yet)
    }
    strip.show(); // for last erased pixel
}

// An "ordered dither" fills every pixel in a sequence that looks
// sparkly and almost random, but actually follows a specific order.
void dither(uint32_t c, uint8_t wait) {
    // Determine highest bit needed to represent pixel index
    int hiBit = 0;
    int n = strip.numPixels() - 1;
    for(int bit=1; bit < 0x8000; bit <<= 1) {
        if(n & bit) hiBit = bit;
    }
    int bit, reverse;
    for(int i=0; i<(hiBit << 1); i++) {
        // Reverse the bits in i to create ordered dither:
        reverse = 0;
        for(bit=1; bit <= hiBit; bit <<= 1) {
            if(i & bit) reverse |= 1;
        }
        strip.setPixelColor(reverse, c);
        strip.show();
        delay(wait);
    }
    delay(250); // Hold image for 1/4 sec
}

// "Larson scanner" = Cylon/KITT bouncing light effect
void scanner(uint8_t r, uint8_t g, uint8_t b, uint8_t wait) {
    int i, j, pos, dir;
    pos = 0;
    dir = 1;
    for(i=0; i<=((strip.numPixels()-1) * 8); i++) {
        // Draw 5 pixels centered on pos. setPixelColor() will clip
        // any pixels off the ends of the strip, no worries there.
        // we'll make the colors dimmer at the edges for a nice pulse
        // look
        strip.setPixelColor(pos - 2, strip.Color(r/4, g/4, b/4));
        strip.setPixelColor(pos - 1, strip.Color(r/2, g/2, b/2));
        strip.setPixelColor(pos, strip.Color(r, g, b));
        strip.setPixelColor(pos + 1, strip.Color(r/2, g/2, b/2));
        }
strip.setPixelColor(pos + 2, strip.Color(r/4, g/4, b/4));

strip.show();
delay(wait);
// If we wanted to be sneaky we could erase just the tail end
// pixel, but it's much easier just to erase the whole thing
// and draw a new one next time.
for(j=-2; j<= 2; j++)
  strip.setPixelColor(pos+j, strip.Color(0,0,0));
// Bounce off ends of strip
pos += dir;
if(pos < 0) {
  pos = 1;
  dir = -dir;
} else if(pos >= strip.numPixels()) {
  pos = strip.numPixels() - 2;
  dir = -dir;
}
}

// Sine wave effect
#define PI 3.14159265
void wave(uint32_t c, int cycles, uint8_t wait) {
  float y;
  byte  r, g, b, r2, g2, b2;
  // Need to decompose color into its r, g, b elements
  g = (c >> 16) & 0x7f;
  r = (c >>  8) & 0x7f;
  b =  c        & 0x7f;
  for(int x=0; x<(strip.numPixels()*5); x++)
  {
    for(int i=0; i<strip.numPixels(); i++) {
      y = sin(PI * (float)cycles * (float)(x + i) / (float)strip.numPixels());
      if(y >= 0.0) {
        // Peaks of sine wave are white
        y  = 1.0 - y; // Translate Y to 0.0 (top) to 1.0 (center)
        r2 = 127 - (byte)((float)(127 - r) * y);
        g2 = 127 - (byte)((float)(127 - g) * y);
        b2 = 127 - (byte)((float)(127 - b) * y);
      } else {
        // Troughs of sine wave are black
        y += 1.0; // Translate Y to 0.0 (bottom) to 1.0 (center)
        r2 =  (byte)((float)r * y);
        g2 =  (byte)((float)g * y);
        b2 =  (byte)((float)b * y);
      }
      strip.setPixelColor(i, r2, g2, b2);
    }
    strip.show();
    delay(wait);
  }
}

/* Helper functions */

//Input a value 0 to 384 to get a color value.
//The colours are a transition r - g - b - back to r
uint32_t Wheel(uint16_t WheelPos)
{
  byte r, g, b;
  switch(WheelPos / 128)
  {
    case 0:
      r = 127 - WheelPos % 128; // red down
      g = WheelPos % 128;       // green up
      b = 0;                    // blue off
      break;
  }
break;
case 1:
g = 127 - WheelPos % 128; // green down
b = WheelPos % 128;       // blue up
r = 0;                    // red off
break;
case 2:
b = 127 - WheelPos % 128; // blue down
r = WheelPos % 128;       // red up
g = 0;                    // green off
break;
} 
return(strip.Color(r,g,b));
}

5.1. 6.2 The LogRhythm_auto.ps1 PowerShell Script

# LogRhythm_auto.ps1
# Paul Ackerman
# V 1.0 August 2015
#
# This script is one component of several referenced by the paper:
# "The LogLED - An LED-Based Information Security Dashboard" by Paul Ackerman available
# at
#
# The purpose of this script is to provide a capability for the InfoSec
# team to monitor critical events and the defence condition of the network.
# The script connects to the LogRhythm API to obtain information about current
# alarms and sends that information to a Teensy micro-controller that drives
# an LPD8806 LED strip.
#
# The script is heavily commented. Please refer to the comments for
# specific functionality.
#
# If you make improvements to the script, please share them with:
# PAckerman@VacciNetLLC.com
#
$com = $null
If ($Host.Name -match 'ise')
{
    Write-Host "WARNING: This script must run in the console"n'Right click script and
select 'Run with PowerShell'"
    break;
}

# change VID/PID string to match your device (if it's not a standard Teensy)
# also following code assumes only one Teensy is connected
try
{
    get-WmiObject Win32_SerialPort -EA 1
    where {$_._FNPDeviceID-match"16C0&PID_0487" -or $_._FNPDeviceID-match"16C0&PID_0483"}
    foreach($com -eq $null)
    {
        $com = $_._DeviceID; $com
    }
catch{$_}
If ($com -eq $null)
{
    "Can't find Teensy, check it is connected and that you have not changed VID/PID"
    break
}
try
{
    $port=new-Object System.IO.Ports.SerialPort $com,9600,None,8,one
```powershell
$port.ReadTimeout = 90
$port.WriteTimeout = 90
}
catch{

# Main loop
while($true) {

# $APIError is used to keep track of when an error occurs reaching the LogRhythm API
# When $APIError is false, no error has occurred.
$APIError = $false

# This script assumes the LogRhythm web service has been configured with # Windows Authentication and this script is being executed by a user with permission # to access the API.
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}

# Create the URI to be called
$URI = "https://logmanagerhostname:4443/LogRhythm.API/Services/AlarmServiceWindowsAuth.svc?wsdl"

try {
    # Call the URI using windows auth
    $AlarmService = New-WebServiceProxy -uri $URI -UseDefaultCredential
} catch {
    # There was an error connecting to the API.
    $APIError = $true
}

# Set Parameters
$startdate = Get-Date
$enddate = $startdate.AddDays(-90)
$enddate = Get-Date
$enddate = $endDate.AddDays(1)

try {
    # Pull the alarms from the last 90 days that have a status of "New". $Results will be an array of alarms including # Various information such alarm creation date, status and severity
    $Results = $AlarmService.GetFirstPageAlarmsByAlarmStatus($startdate, $true, $enddate, $true, "New", $true, $true, $true, 5000, $true)
} catch {
    $APIError = $true
}

# The number of alarms will be tracked in order to display an effect when the number of alarms increases
$numAlarms = $Results.Alarms.Count

# This dumps just the severity rating from the alarms into an array called $RBP. This is a numeric score from 0 to 100 # and will be used to determine the color value for the alarm
$RBP = $Results.Alarms.RBPAvg

# These variable represent the quantity of LEDs that will be illuminated of each color. They must all start at 0.
$green = 0
$blue = 0
$orange = 0
$red = 0
```

Paul Ackerman, PAckerman@VacciNetLLC.com

© 2015 The SANS Institute

Author retains full rights.
An LED-Based Information Security Dashboard

Paul Ackerman, PAckerman@VacciNetLLC.com

# Here we loop through all the alarms and increment the $green, $blue, $orange and $red values depending on the severity of the alarm.
# Feel free to adjust the risk scores below to suit your liking.
foreach ($code in $RBP)
{
    if ($code -lt 60)
    {
        # Alarms with scores below 60 will be green.
        $green = $green + 1
    }
    elseif($code -le 69)
    {
        # Alarms with scores between 60 and 69 will be blue. This started as yellow but yellow and orange can be difficult for
        # some people to differentiate so blue was chosen to replace yellow.
        $blue = $blue + 1
    }
    elseif($code -le 79)
    {
        # Alarms with scores between 70 and 79 will be orange.
        $orange = $orange + 1
    }
    else
    {
        # Alarms with scores 80 or above will be red.
        $red = $red + 1
    }
}
# Write the number of each color and the number of alarms to the console for informational purposes
write-host "Green:" $green " Blue:" $blue " Orange:" $orange " Red:" $red
$numAlarms = $numAlarms - 1 # The Teensy looks for serial data beginning with an "@". All other data is considered garbage
$LEDcode = "@"
# The effect variable is used to represent which effect to display on the teensy. This is the 98th argument
$effect = 0
# LEDcount is used to keep track of how many LED's have been addressed so far. We run into a problem when
# there are more alarms than LED's so we can only address the first 32 alarms. If more than 32 alarms exist
# their values will be discarded and the top LED will be made to blink indicating that more alarms exist.
$LEDcount = 0
# Build LED string to send to the teensy
# The Teensy expects a red, green, blue, value for each of the 32 LED'score, a blink value and an effect
# ending in a comma. Example:
# "@r,g,b,r,g,b,...,r,g,b,blink,effect,"
# Acceptable values for red,green,blue parameters are 0-127.
# Acceptable values for the blink parameter are 0-32. If 32 is used, no LED will blink as they are addressed 0-31
# Acceptable values for the effect parameter are 0-9. Refer to the Teensy sketch documentation to determine
# what each effect code will do as this may change independently of this PowerShell script.
if ($numAlarms -eq -1)
{
    # There are no alarms. Great job! Your reward is a random effect followed by turning the strip off.
    $effect = Get-Random -minimum 2 -maximum 9
}
$LEDcode = "0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0," + $effect + ","

else

  # Here, we're adding the string "0,32,0," which is a dim green LED however many times we incremented the "$green" counter earlier when we looped through all the alarms $RBP.
  for ($i=0; $i -lt $green; $i++)
  {
    if($LEDcount -lt 32)
    {
      $LEDcode = $LEDcode + "0,32,0,"
      $LEDcount++
    }
  }

  # Same as above for blue, orange and red
  for ($i=0; $i -lt $blue; $i++)
  {
    if($LEDcount -lt 32)
    {
      $LEDcode = $LEDcode + "0,0,64,"
      $LEDcount++
    }
  }

  for ($i=0; $i -lt $orange; $i++)
  {
    if($LEDcount -lt 32)
    {
      $LEDcode = $LEDcode + "32,8,0,"
      $LEDcount++
    }
  }

  for ($i=0; $i -lt $red; $i++)
  {
    if($LEDcount -lt 32)
    {
      $LEDcode = $LEDcode + "32,0,0,"
      $LEDcount++
    }
  }

  # Fill in the remaining LED's as off.
  for ($i=0; $i -lt (31-$numAlarms); $i++)
  {
    if($LEDcount -lt 32)
    {
      $LEDcode = $LEDcode + "0,0,0,"
      $LEDcount++
    }
  }

  # Here we're saying if the number of alarms has increased, display effect 2 and then show the alarms. Otherwise, just show the alarms
  if ($numAlarms -gt $prevalarms)
  {
    $effect = ",2,"
  } else
  {
    $effect = ",0,"
  }

  $blink = $numAlarms
  $prevalarms = $numAlarms

  # If we have more alarms than will fit on the LED strip, blink the top LED to indicate this.
  if ($blink -ge 31)
The LogLED
An LED-Based Information Security Dashboard

Paul Ackerman, PAckerman@VacciNetLLC.com
```csharp
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    \$port.close()
    catch [TimeoutException]
    {}
}

# Wait 30 seconds and then repeat. You can adjust this time to be more or
# less but the lower limit should be
# a second or two greater than any effect that displays on the teensy or
# you might end up trying to send data
# to the teensy while isn't listening. For example, if a rainbow effect
takes 20 seconds to complete on the
# teensy, you shouldn't make this value any lower than 22 seconds.
start-sleep -s 30
}```
# Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>Training Event</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS San Francisco Fall 2019</td>
<td>San Francisco, CAUS</td>
<td>Sep 23, 2019 - Sep 28, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Kuwait September 2019</td>
<td>Salmiya, KW</td>
<td>Sep 28, 2019 - Oct 03, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Cardiff September 2019</td>
<td>Cardiff, GB</td>
<td>Sep 30, 2019 - Oct 05, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Northern VA Fall- Reston 2019</td>
<td>Reston, VAUS</td>
<td>Sep 30, 2019 - Oct 05, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Riyadh October 2019</td>
<td>Riyadh, SA</td>
<td>Oct 05, 2019 - Oct 10, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Doha October 2019</td>
<td>Doha, QA</td>
<td>Oct 12, 2019 - Oct 17, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Cairo October 2019</td>
<td>Cairo, EG</td>
<td>Oct 19, 2019 - Oct 24, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Santa Monica 2019</td>
<td>Santa Monica, CAUS</td>
<td>Oct 21, 2019 - Oct 26, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Orlando 2019</td>
<td>Orlando, FLUS</td>
<td>Oct 28, 2019 - Nov 02, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam October 2019</td>
<td>Amsterdam, NL</td>
<td>Oct 28, 2019 - Nov 02, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Houston 2019</td>
<td>Houston, TXUS</td>
<td>Oct 28, 2019 - Nov 02, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS DFIRCON 2019</td>
<td>Coral Gables, FLUS</td>
<td>Nov 04, 2019 - Nov 09, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Mumbai 2019</td>
<td>Mumbai, IN</td>
<td>Nov 04, 2019 - Nov 09, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Dallas Fall 2019</td>
<td>OnlineTXUS</td>
<td>Sep 23, 2019 - Sep 28, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>

Last Updated: September 21st, 2019