[[copy edited by Kezia Endsley]]

TE color code: GREEN checked out OK, YELLOW needs attention, RED (this) is an error.

Chapter 11: Building Interactive Security Visualizations

[AU: Global for whole chapter, if any color is needed for the code in this chapter, please add it (and insert the new listing numbering you’ve suggested). If you have questions, please contact me right away. Thanks, Kevin (PjE)]

[AU: And please remember that if the code is changed, make the appropriate change in the code download as well and submit a new version of the code download for the chapter with your AR. Thanks, Kevin (PJE)]

“Many in the design community understand that design must convey the essence of a device’s operation; the way it works; the possible actions that can be taken; and, through feedback, just what it is doing at any particular moment. Design is really an act of communication, which means having a deep understanding of the person with whom the designer is communicating.”

Donald A. Norman, *The Design of Everyday Things*

The main purpose behind any of your data visualization efforts should be to help consumers understand and learn from the data. In other words, *effective communication* is the primary goal of your visual creations.

As you’ve seen in previous chapters, developing simple and successful *fixed* tables and charts requires knowledge, skill, and practice, but can provide substantive illumination of a topic, issue, or problem if executed correctly. In most cases—probably 95 percent of the time—these fixed views are all that is needed to achieve the goal of communication. There are situations, however, when static views of data are either insufficient or just not practical, requiring the move to a more dynamic medium to help consumers explore the messages the data has to offer. This chapter helps you understand when the move to interactive visualizations makes sense and introduces you to some of the resources and techniques that will help you craft effective messages, dashboards, and exploration tools.

type="note"

The skills, art, and science surrounding interactive visualizations span a multitude of disciplines across many decades. As a result, this single chapter serves more as a survey and reference for further study for the topic as a whole. It provides practical guidance for where to apply interactivity within the scope of information security.

Moving from Static to Interactive

Assuming the “95 percent” premise holds true, your first instinct when planning visualizations should be to “go static.” It will generally take much less time to construct fixed visualizations even with the tweaking and polishing necessary to produce a consumer-worthy graphic. You should also consider sticking with stationary images if the project you’re working on is fairly discrete with a data set having a minimal number of dimensions (that is, rows, variables/columns/fields). As Scott Murray put it in his book *Interactive Data Visualization for the Web,* “A fixed image is ideal when alternate views are neither needed nor desired, and required when publishing to a static medium, such as print.”

If you’re still feeling the “interactive itch,” there are three primary goals to consider when contemplating a new visualization:

* **Augmentation:** If adding interactive capabilities helps speed up or automate tasks consumers would normally perform manually, going interactive is definitely the right thing to do.
* **Exploration:** If the number of dimensions and size/diversity of the data set grow sufficiently large, it may be better to enable consumers to explore the relationships and outcomes on their own rather than trying to guess which set of static graphics will be most useful.
* **Illumination:** If a topic is complex enough, it may help to provide a well-executed, interactive visualization that provides a user-friendly interface for directed/constrained navigation around the data you’ve chosen to present.

Let’s delve a bit further into each of these areas with a focus on information security examples.

Interaction for Augmentation

There are many repetitive, time-consuming, data-driven tasks in information security. Logs must be collected and correlated, alerts must be received and attended to, and anomalies must be investigated. These actions often involve running a variety of utilities over individual pieces of data or sets of data elements to determine whether there truly is an issue on your network. Any tool that helps alleviate this tedium and speeds up reliable detection of malicious activity is a welcome addition to any security engineer’s toolbox.

Recognizing this fact, a research team led by Robert Erbacher worked to understand both the problem domain—situational awareness of malicious network activity—and how incident responders think and process information. This resulted in the creation of VisAlert (http://digital.cs.usu.edu/~erbacher/publications/VisAlertCGA2006.pdf1), a visual correlation tool that facilitates situational awareness in complex network environments. See Figure 11-1.

[AU: Does that superscripted 1 above refer to the reference #1 at the end of the chapter? If so, I’m not sure why the URL for VisAlert is connected to that reference. Can you clarify? Thanks, Kevin (PJE)]

Figure 11-1: The VisAlert Visual Correlation Tool [793725 c11f001.png]

[AU: It would help this illustration if you briefly explained what’s going on in this figure for readers. What does it show? How do you intend to use the illustration going forward in the chapter? Thanks, Kevin (PjE)]

With tools such as Circos (http://circos.ca/), it’s fairly straightforward to build a radial diagram similar to the VisAlert model in Figure 11-1 and add some interactive features. However, it takes more than eye-candy appeal for any visualization—fixed or interactive—to be truly useful. VisAlert’s detailed focus on the following areas makes it notable.

Define the Problem

This is merely an extension of the “start with a question” mantra you’ve seen in many of the preceding chapters. Although there is merit in building visualizations in a vacuum to learn how to work with a new language or framework, it is imperative that you understand what problem you’re trying to solve with a consumer-oriented interactive visualization and who the users will be before you attempt to deliver a finished product. Even if you’re an established practitioner, your personal experiences may give you insight into only one aspect of a problem domain, and collaboration with others—especially those who you believe to be the natural consumers of your interactive visualization work—can make or break a project.

For the VisAlert team, this ultimately meant their goal was to aid analysts’ decision-making processes by providing a robust visual correlation mechanism. Rather than try to build a new intrusion-detection system or deliver a “toy model” solution that works only with perfect and limited data sets, they chose to design a system that works at-scale with real-world data volumes and types that security analysts already use in their daily workflows. Although the problem scope is narrowly defined, it has sufficient breadth and scope to be useful as well as visually appealing.

Seek Domain Expertise

The VisAlert team started with real-world information security analysts to understand their *mental models* of how they go about identifying badness. Mental models are conceptual models of the way things work or people’s understanding of how to interact with the world or systems around them. Security analysts develop domain-specific mental models through their training and practical work experiences. These models evolve with each successful (or failed) identification and eradication of malicious activity. With each investigation, analysts learn which processes provided the most value and these are automatically added to their existing mental framework. By working with these individuals throughout the design process, the team was able to identify what parts of the analysts’ workflows would benefit from enhanced visualizations (for example, inclusion of salient parts of network diagrams and automatically highlighting specific protocols and paths) and automation (for example, DNS lookups and targeted correlations).

Take an Interdisciplinary Approach

The team drew on the talents and works of experts in the fields of information architecture, cognitive psychology, application development, and computer science—along with the domain experts—to build and refine the tool. They called this process a “modified hermeneutic circle”—the movement back and forth between the parts and the whole. It’s shown in Figure 11-2.

Figure 11-2: The VisAlert Visual Correlation Tool design methodology [793725 c11f002.png]

Their methodology has a strong resemblance to the Agile development process (http://agilemanifesto.org/principles.html), where all those involved are equal partners, each working together to yield a successful finished product. If your organization has an application development team and you’re not familiar with Agile, you would do well to invite a member to lunch to understand how Agile works in the real world. (Plus, you’ll have made a friend in the development community and can hopefully help them understand application security a bit better as well.)

Fundamentally, both concepts employ highly effective and efficient feedback loops to help ensure your project stays on the rails and arrives at the desired destination as quickly as possible. You may be the one building the finished product and you may be a savvy practitioner, but you should also regularly seek input and feedback from others in and outside your domain to ensure you’re constructing the right elements.

The VisAlert tool has been featured in papers and security-oriented conferences since 2006 but has been developed as a commercial or open source product as of this book’s publication.

Interaction for Exploration

Most networks contain their fair share of vulnerabilities. The Nessus (http://www.tenable.com/products/nessus) vulnerability scanner (by Tenable) is one of the most commonly used tools that can help you find them. If you’ve ever seen the output from a detailed Nessus report (Figure 11-3), you know that each host will have a listing of vulnerable components and each component will have many attributes, including basic and detailed descriptions, overall rating, and CVSS score. A full report can be hundreds of pages long and makes for excellent nighttime reading if you’re having trouble sleeping.

[[Author: Will readers know what a CVSS score is without explaining above? Kezia]]

Figure 11-3: Sample Nessus detailed vulnerability report [793725 c11f003.png]

[AU: Addition below okay to point readers specifically to where it is on the website. Also, do you have permission to include this content on the website? Thanks, Kevin (PJE)]

Even a small network, such as the one created for the VAST 2011 visualization challenge (http://hcil.cs.umd.edu/localphp/hcil/vast11/), can have thousands of vulnerability findings. (The VAST network data—included on this book’s website as vast\_2011.nbe at www.wiley.com/go/datadrivensecurity—has over 2,000.) Although it’s possible to spin the data multiple ways and produce reams of static visualizations, this is definitely a perfect example of how an interactive tool can help security analysts explore and prioritize how they will attack the problem of which vulnerabilities to remediate first.

Tenable does provide interactive reporting tools, but this chapter focuses on an innovative open source tool released in 2013 by John Goodall called the Nessus Vulnerability Explorer (NV) ([http://ornl-sava.github.io/nv/#](http://ornl-sava.github.io/nv/)). NV allows you to take an export from your Nessus scans, drag the file right into your browser, and begin exploring the vulnerabilities contained within. See Figure 11-4.

Figure 11-4: Nessus Vulnerability Explorer interactive treemap interface [793725 c11f004.png]

The interface is based on a treemap, which is a visualization that enables presentation of hierarchical data in a very compact way through nested rectangles, with the size and color of each rectangle being mapped to categorical or quantitative variables within the data set. Treemaps take a bit of getting used to, but once you learn how to decode them they can become valuable allies in targeted visualizations.

Goodall’s interactive treemap lets the consumer rearrange the structure of the hierarchy through a simple drag-and-drop action, so you can present a traditional IP address-centric view of the vulnerabilities or switch to a view based on Nessus vulnerability (plug-in) ID or even by port. Through a single click, nodes can be sized by volume or potential impact and vulnerability details are revealed through single clicks on individual rectangles.

The view in Figure 11-4 has over 240 nodes, yet it’s very straightforward (and quick) to see all nodes with similar vulnerability profiles. All necessary information is kept onscreen and the bar charts at the bottom of the display provide a useful high-level overview to help guide exploration. A traditional summarized report view would no doubt require much scrolling and panning to provide the same type of information and it would be almost impossible to discern patterns in the environment.

However, all exploratory interfaces do not need to be this elaborate. Figure 11-5 shows a simple Excel workbook of a firewall log extract that includes filtering controls at the top of the log entry data table. It also has two pivot tables showing views by firewall and port (respectively), with matching bar charts that dynamically change as you manipulate the pivot table values. More modern versions of Excel do not have the workbook size limitations of previous offerings and can comfortably fit over a million rows and 16,000 columns, provided you have a robust enough system to support such a large workbook. You might be surprised just how useful it can be to simply provide intelligently summarized tabular views of data sets—paired with basic visualizations—that can be easily sorted on demand by the consumer. It may sound simple, but remember: you still need to do the hard work of finding, cataloging, acquiring, cleaning, augmenting, and processing the data (ah, the glamorous life of a security data scientist).

Figure 11-5: Excel pivot table with linked charts [793725 c11f005.png]

Interaction for Illumination

Although everyone may seem to be carrying an i-device of some sort and is constantly plugged in to everything, the truth is that most individuals still have only a surface-level understanding of the digital world they live in. For instance, they know that their Instagram app requires an account with a username and password before they can post pictures for their friends to see, but the details of the binary world below that process—where hue, saturation, and brightness are digitized, network packets are exchanged, and information is transported and stored potentially thousands of miles away—remains as much a mystery as does most of the inner workings of a modern automobile engine.

Even in our workplaces, where business processes are more often understood, the complexity of the information technology components that make those processes possible can be somewhat overwhelming to IT specialists, let alone business professionals.

Consider that a modest application has code that might be touched by over 30 developers, supported by over 15 operations administrators, span 3 firewall zones, and have components that reside on 16 disparate systems. It’s incredible we have as much security as we do in such diverse and complex environments and a bit more understandable why all of those individuals involved in the process don’t fully grasp all the nuances of how to ensure that security is a primary emergent property of the system as a whole.

Understanding how complexity is masked, hidden, or ignored should make it easier to see why topics we security-folk are passionate about⎯such as encryption, system/data integrity, and data privacy⎯are faint blips on the radars of most individuals. However, our cause and profession have merit, and we *can* help raise awareness of these important topics. One good way to do this is through the use of interactive visualizations.

A great example of *how* to do this is the “World’s Biggest Data Breaches” visualization (http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/), created by David McCandless and Tom Evans of Information is Beautiful (http://www.informationisbeautiful.net/). See Figure 11-6.

Figure 11-6: World’s Biggest Data Breaches interactive visualization [793725 c11f006.png]

Data breaches, as discussed in Chapter 7, are a reality yet are not well understood outside of the security domain (perhaps not even fully *within* the security domain). When the technical and general news media report almost one breach per week, it can be difficult for people to keep up, let alone digest the diversity of the attacks. David and Tom—who are visualization and development experts, not information security professionals—set out to build an easy-to-use tool that would help consumers gain a better understanding of the quantity, variety, and magnitude of breaches that have made headlines over the past few years.

[AU: Is that a source that should be included in the references? Thanks, Kevin (PJE)]

By following a paradigm of “overview first, zoom and filter, then details-on-demand” put forth by Ben Shneiderman back in 1996 in his “Visual Information Seeking Mantra,” they created an interactive bubble chart (see Figure 11-6) organized vertically by year. Consumers can filter the display to show breaches by organization type or method of leak and can also change the factors that make up bubble size and color.

Publications such as the Verizon Data Breach Investigations Report (http://www.verizonenterprise.com/DBIR/2013/) and Trustwave Global Security Report (http://www2.trustwave.com/rs/trustwave/images/2013-Global-Security-Report.pdf), plus online databases such as DataLoss DB (<http://datalossdb.org/>) and the Privacy Rights Clearinghouse (http://www.privacyrights.org/data-breach), have covered breaches for many years, yet tend to be read and mined mostly by information security professionals. What has made David and Tom’s interactive tool more appealing and useful to a much broader audience than these established resources?

Make Interfaces Accessible

There’s nothing quite like a never-ending, scrolling table filled with security jargon and wrapped in cold, official language to make the average person head for the nearest cat picture. Even a well-crafted, comprehensive report can be daunting to pick up and look through when the topic is so far removed from the daily experience of even the most tech-savvy business executive.

The World’s Biggest Data Breaches visualization succeeds because it presents the data within a familiar and friendly setting—a web page—and makes excellent use of color, style, and design to present a tool that has an intuitive look and feel with no fear of “breaking” anything. The “buttons” look and behave as expected. The filtering interface has plenty of whitespace and steers clear from too much jargon or too little context. Mouse movements and actions provide instant, game-like feedback; and, even without instruction, the interface is almost instantly usable.

Imagine if this had been released as a Microsoft Excel file (yes, you can make clickable bubble charts with Excel) with macro warnings popping up on open and the ribbon and column headers consuming prime display space and with your operating system switching between Excel and your default browser whenever you clicked to see the news story behind the detail. The basic functionality would have been the same, but the user experience would have been radically different.

Your consumers live in the browser and that’s where most (if not all) your creations should be targeted for deployment. Latter sections of this chapter introduce some of the technologies that make these visualizations possible, but they do not include phrases like “Java applet” or “Adobe Flash.” Relying on the native capabilities of modern browsers and web frameworks will help you reach the largest possible audience in the most compatible and accessible ways possible. It will have the added benefit of making you sympathize a bit more with the complexities faced by user-interface developers (whom you should also take to lunch on occasion to trade security knowledge for useful front-end coding tips and techniques).

type="general"

The (Slow) Demise of Flash and Java

There was a time when Java and Flash applets were the only way to add “decent” visual interactivity to a website. Java was (and is) a formal language taught in many schools, which had made it an especially easy choice for academic visualizations. Flash was (and is) easy to learn with friendly development tools that have made it highly popular among the general web development community.

Although Flash still commands a presence on around 17 percent of websites (Figure 11-7), the use of it as a visualization medium is in a slow, steady decline. In contrast, Java applets hold on to a razor-thin 0.1 percent share of the web.

Figure 11-7: The decline of flash [793725 c11f007.png]

TE: I think that this graph should be redone with a vertical axis range from 0% to 30%. The current range (17% to 29%) does not accurately convey (visually) the absolute change in share. Unless there is a VERY good reason, all time series graphs intended to communicate about trends should have a range starting with zero.  
If you end up editing it, I suggest you change y-axis to “Percent *Flash* Usage Across Web”. Right now, it doesn’t say “Flash” anywhere. Kezia]]

[AU: If you redo this figure, please provide the new version when you submit the author review of this chapter. Thanks, Kevin (PJE)]

The fading of each technology can be attributed to many factors, including:

* The never-ending vulnerability, breach, and security update cycle
* The rise in popularity of platforms such as the iPad, iPhone, and other touch environments that do not provide support for website elements built with these tools
* The increased native platform capabilities due to widespread adoption of HTML5, CSS, and JavaScript across the most used browsers

To reach the broadest audience, it’s best to avoid proprietary technologies or visualization toolkits that require browser extensions.

Facilitate Directed Exploration

Donald Norman coined the phrase “the tyranny of the blank screen” in his book *The Design of Everyday Things.* The perfect, illuminating, interactive visualization lies somewhere between this fully open, onscreen world and a fixed graphic. Which design choices made the World’s Biggest Data Breaches visualization easy to explore?

* **Critical exploration elements and operations are prominent and visible.** Through consistent colors, shapes, and prominent placement, the controls for the visualization are immediately discernable. By having the filter controls come up right after the visualization loads, there is the immediate reaction of “Oh, I can click this!” on the part of the consumer. Color also draws attention to what the creators feel are especially compelling stories.
* **All components and actions are consistent and deliberate.** Mouse movements highlight elements and mouse clicks select options and provide detail. There is no jumping between mouse and keyboard or switching between dragging and clicking. The interface becomes immediately predictable with no surprises, apart from interesting and engaging stories.
* **Feedback is instant and all operations are safe.** Although the site loads fairly quickly given all the data and resources it uses behind the scenes, there is a slight delay and this is where the helpful feedback starts. A familiar “loading” message appears but quickly fades directly to the core visualization. Every click produces instant feedback that is 100 percent undoable, either via the controls on the visualization or with a quick click of the browser reload button. This feeling of safety puts consumers at ease and encourages them to explore.
* **Actions are limited.** The interface provides options to change color and size of bubbles and highlight certain organization and breach types. However, you cannot group elements together and generate a bar chart or select individual organizations out from a list of thousands. These constraints make the interface much less daunting—a condition referred to as the *paradox of choice*—since some argue that people want more freedom and more tools and ways to explore. Limiting actions also enables you to shape or guide the exploration in a particular direction. Considering how fixed graphics represent the extreme in limiting actions, you should be able to think back to what made the data interesting to you as you explored it and come up with a set of constrained, exploratory actions that lie somewhere between the freedom of an RStudio window and the constraints of a static graphic.

type="note"

Barry Schwartz writes about *The Paradox of Choice: Why Less Is More* (Ecco, 2004) in more detail in his book.

[AU: Addition of note here okay? Thanks, Kevin (PJE)]

Include Appropriate Detail

Breaches are complex entities, as illustrated by the breadth and depth of the VERIS taxonomy explained in Chapter 7. This level of technical detail is completely inappropriate for the mass-consumer audience of the World’s Biggest Data Breaches visualization. Rather than bombard the consumer with multi-level taxonomy details, McCandless and Evans opt for simple summaries and succinct descriptions available upon clicking, while making detailed news stories also available on demand.

The level of detail you choose to provide in this type of visualization is highly dependent on the target consumer. Including VERIS-level taxonomy details within a similar tool released at a conference of security professionals focused on metrics (Metricon, http://securitymetrics.org/) is both appropriate and expected by the audience. You must have a solid grasp of who will be using your creations and what their level of expertise and expectations are in order to build a truly successful interactive visualization.

Developing Interactive Visualizations

Even with the elimination of Flash and Java as options, you are still faced with the aforementioned paradox of choice when it comes to deciding on how you want to develop interactive visualizations. Most often, you’ll have to roll up your sleeves and write code, especially since you will usually be dealing with sensitive data that cannot be published on the public Internet. The vast majority of Internet-accessible “point-and-click” tools store data in the “cloud” and use public websites for the presentation layer, but there are desktop tools that can be of great assistance when fixed visualizations are not sufficient.

Building Interactive Dashboards with Tableau

[AU: Should Mackinlay’s paper be listed in the references as well? It’s note currently. Thanks, Kevin (PJE)]

One standalone, Office-like tool that excels at building assisted/directed interactive visualizations and dashboards is Tableau (http://tableausoftware.com/). Tableau is a Windows-only application that was heavily influenced by research conducted by Jock Mackinlay in automating the design of graphical presentations of relational information (http://cs171.org/2008/papers/mackinlay86.pdf). A foundational premise of Tableau, therefore, is to have the system analyze your data and provide suggestions for the best way to visualize it. If your goal is to build interactive, user-friendly dashboards or quickly provide an interactive exploratory interface for a complex data set, Tableau should be your “go to” tool of choice.

If you look back at the security awareness use case in Chapter 10, one way to build such a survey is to use an in-house tool such as Microsoft SharePoint or look to a commercial solution such as SurveyMonkey to present the desired survey questions. The raw survey results will look something like the almost endless series of data points shown in Figure 11-8. Slicing and dicing that data into generate static views *is* possible, but is neither practical nor useful for communicating the messages contained within the data set.

Figure 11-8: Raw data from the security awareness survey [793725 c11f008.png]

Tableau can easily digest this data, analyze the types of variables it contains, and guide you through selecting the most appropriate visualizations to encode the individual elements or relationships between elements. That’s great for producing fixed graphics, but Tableau can also be used to quickly generate interactive visualizations that can be distributed to other Tableau Desktop users or be presented to web browsers via Tableau Server.

After looking at the data (awareness-survey.csv, which is available on the book’s website, www.wiley.com/go/datadrivensecurity, as part of Chapter 11 download materials), we decided it would be most helpful to provide views of each survey answer by business unit, years employed, and employee level (management or individual contributor), since we could then attempt to discern if any of those factors stood out (which will help us tailor messages in future awareness initiatives). With this goal in mind, we used Tableau to create the interactive dashboard shown in Figure 11-9. It’s viewable at http://public.tableausoftware.com/views/UserAwareness/UserAwareness. The whole process—from data import to finished dashboard—took about 20 minutes.

Figure 11-9: Awareness survey results presented with Tableau [793725 c11f009.png]

Rather than build a giant, scrolling web page, we chose to let consumers explore individual survey questions and had Tableau automatically pivot the compact detail views on demand. Each visual component in each section is also selectable and provides further levels of detail when inspected (Figure 11-10).

Figure 11-10: Tableau details on demand [793725 c11f010.png]

[AU: Which file is that in the web materials? Name is so readers can go right to it. Thanks, Kevin (PJE)]

This all required nothing more than a few mouse clicks and drags. We never entered even one line of code, yet produced an interactive tool that can be used by anyone with a web browser. Plus, we can give the entire workbook (also available on the book’s website) to other analysts to produce other customized views—provided they also have the Tableau Desktop software.

Tableau is great for producing straightforward fixed and interactive visualizations using standard charting components. However, if you want to create more specialized interactive visualizations or prefer not to be locked into a proprietary desktop tool, you’ll need to head to your favorite text editor and start coding.

Building Browser-Based Visualizations With D3

There is a vast landscape of tools, languages, and techniques available to help you craft engaging, web-based, fixed, and interactive data visualizations. It would be impossible to cover them all in one book, let alone part of one chapter, so we’ll highlight one of the most flexible and popular visualization libraries available today—D3—and show you a fully working example using a meta-language built on top of D3⎯Vega.

D3 (http://d3js.org/) is a powerful JavaScript library created by Mike Bostock that makes it possible to dynamically transform and manipulate the contents of web pages based on data. To fully bend D3 to your will, you’ll need to

* Become proficient in the web trifecta⎯HTML5, Cascading Style Sheets (CSS), and JavaScript
* Be familiar with the structure of Scalable Vector Graphics (SVG)
* Have a solid understanding of the Document Object Model (DOM); see http://www.w3.org/TR/1998/WD-DOM-19980720/introduction.html

However, you can begin to learn D3 without deep knowledge in those areas, just by viewing and exploring the plethora of examples found on the “official” D3 GitHub site (https://github.com/mbostock/d3/wiki/Gallery) and by gathering expertise along the way.

Unlike most proprietary technologies, you can dissect and inspect all D3 visualizations just by choosing “view source” from your web browser. Since D3 visualizations are fully driven by the data being visualized, the data itself is also available for download and should be in a recognizable format—usually CSV, TSV, JSON, or hardcoded HTML tables and JavaScript arrays.

Getting started with D3 requires only three things⎯a text editor, the D3 JavaScript library, and a web server. To prove this, read through this annotated, basic example of a static bar chart (Figure 11-11) to see what it’s like to code in D3.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<style>

**rect.bar {**

**fill: #54278F;** */\* fill color for the bars \*/*

**}**

**.axis text {**

**font: 10px sans-serif;** */\* 10-pt text for axis labels \*/*

**}**

**.axis path, .axis line** { */\* line style for the axes \*/*

**fill: none;**

**stroke: #000;**

**shape-rendering: crispEdges;**

**}**

</style>

*// Load the D3 js library*

**<script src="http://d3js.org/d3.v3.min.js" charset="utf-8"></script>**

</head>

<body>

<script>

*// set up the data that will generate the bar chart*

**var data = [3, 3, 5, 9, 15, 18];**

*// define that margins for the plot and document*

**var margin = {top: 40, right: 40, bottom: 40, left: 40},**

**width = 960,**

**height = 500;**

*// we can use many scales with D3, but we'll stick with a basic*

*// linear scale for the X axis that is based on the values*

*// contained in our data set. in ggplot parlance this would*

*// be akin to using scale\_x\_continuous()*

**var x = d3.scale.linear()**

**.domain([0, d3.max(data)])**

**.range([0, width - margin.left - margin.right]);**

*// for the Y axis, we'll use an ordinal scale since these are really*

*// just individual factors being displayed. in ggplot parlance, this*

*// would be akin to scale\_y\_discrete()*

**var y = d3.scale.ordinal()**

**.domain(d3.range(data.length))**

**.rangeRoundBands([height - margin.top - margin.bottom, 0], .2);**

*// apply the scales to each axis, setting attributed for text*

*// text alignment and tick marks*

**var xAxis = d3.svg.axis()**

**.scale(x)**

**.orient("bottom")**

**.tickPadding(8);**

**var yAxis = d3.svg.axis()**

**.scale(y)**

**.orient("left")**

**.tickSize(0)**

**.tickPadding(8);**

*// create an SVG element at the top of the the document body*

*// that will hold the bar chart visualiztion, setting basic*

*// layout parameters*

**var svg = d3.select("body").append("svg")**

**.attr("width", width) *// 'attr' sets DOM element attributes***

**.attr("height", height)**

**.attr("class", "bar chart")**

**.append("g")**

**.attr("transform", "translate(" + margin.left + "," + margin.top + ")");**

*// this creates all the bars in the chart using SVG 'rects'.*

*// try chaning the number of entries and values in the 'data' array*

*// above to see how it affects the display*

**svg.selectAll(".bar")**

**.data(data)**

**.enter().append("rect") *// 'enter+append' creates new elements***

**.attr("class", "bar") *// each 'rect' will use the CSS 'bar' format***

**.attr("y", function(d, i) { return y(i); }) *// scaled y coordinate***

**.attr("width", x) *// width based on the x value***

**.attr("height", y.rangeBand()); *// bar widths dynamically scaled to fit***

*// display the axes we set up earlier*

**svg.append("g")**

**.attr("class", "x axis")**

**.attr("transform", "translate(0," + y.rangeExtent()[1] + ")")**

**.call(xAxis);**

*// we could have embedded labels in array, but this just assigns*

*// A-Z+ character codes, which helps show how to make almost any*

*// D3 element dynamic*

**svg.append("g")**

**.attr("class", "y axis")**

**.call(yAxis)**

**.selectAll("text")**

**.text(function(d) { return String.fromCharCode(d + 65); });**

</script>

</body>

</html>

[AU: If you could name the file it’s in, again, that would be a good help to the reader. Thanks, Kevin (PjE)]

The code is available on the book’s website and you can test the visualization in your browser by using the built-in HTTP server found in Python standard library and executing:

python -m SimpleHTTPServer 8888 &

Execute this in the directory containing the example D3 HTML file and point your browser to http://localhost:8888/.

Figure 11-11: Basic D3 bar chart [793725 c11f011.png]

If the syntax looks a bit daunting, remember that it’s just a web page with formatting and JavaScript. You can start to get more comfortable with this code (or any D3 example) by experimenting with changing small things like the bar color and axis fonts. Then add, remove, and modify elements in the data array. If you use Google Chrome or Mozilla Firefox, you can bring up the Developer Tools JavaScript console and interact directly with the document elements. For instance, you can see all of the objects that were created by D3 when you told it to make the bars by typing svg.selectAll(".bar") in the console (once the visualization displays). You can inspect the results (Figure 11-12).

Figure 11-12: Viewing D3-created elements in the JavaScript Console [793725 c11f012.png]

More complex and interactive D3 code can take a bit of getting used to, but there are ways of using D3 without always having to interact with code on this level.

Going Meta with Vega

If the ggplot library is the R incarnation of the “grammar of graphics,” then Vega (http://trifacta.github.io/vega/) is D3’s counterpart.

type="note"

For more on the grammar of graphics, read Leland Wilkinson’s *The Grammar of Graphics, Second Edition* (Springer, 2005).

With Vega, you describe a visualization using very readable JSON and simply use Vega’s parse() function to read the file and display the visualization. The Vega library takes care of translating the specification into the appropriate D3 code. To see the difference, compare the raw D3 bar chart example given previously with this Vega version:

{

"width": 500,

"height": 960,

"padding": {"top": 40, "left": 40, "bottom": 40, "right": 40},

"data": [

{

"name": "table",

"values": [

{"x": "A", "y": 3}, {"x": "B", "y": 3},

{"x": "C", "y": 5}, {"x": "D", "y": 9},

{"x": "E", "y": 15}, {"x": "F", "y": 18}

]

}

],

"scales": [

{

"name": "x",

"type": "ordinal",

"range": "width",

"domain": {"data": "table", "field": "data.x"}

},

{

"name": "y",

"range": "height",

"nice": true,

"domain": {"data": "table", "field": "data.y"}

}

],

"axes": [

{"type": "x", "scale": "x"},

{"type": "y", "scale": "y"}

],

"marks": [

{

"type": "rect",

"from": {"data": "table"},

"properties": {

"enter": {

"x": {"scale": "x", "field": "data.x"},

"width": {"scale": "x", "band": true, "offset": -1},

"y": {"scale": "y", "field": "data.y"},

"y2": {"scale": "y", "value": 0}

},

"update": {

"fill": {"value": "#54278F"}

}

}

}

]

}

[AU: Does this bar chart end up looking different? Should there be a figure of this one? Thanks, Kevin (PJE)]

This is much more readable than straight D3 code and the JSON format makes it easy to build graphics based on templates that are populated with computed data and customized styles. You can also use this flexibility—combined with some extra JavaScript code—to build fully interactive visualizations.

Creating an Interactive “Threat Explorer”

[AU: In light of your revisions to Ch. 4 to remove the term “badness,” you will likely want to revise the below to refer to the language that’s now used in Ch. 4 instead. Please revise. Thanks, Kevin (PjE)]

Imagine that you’ve been asked to visualize which internal hosts are talking to external hosts on a port-by-port basis using the same visualization technique as the “badness” graphs you saw in Chapter 4. This will require adapting the graph code a bit to work with firewall data, but that should be a simple exercise at this point.

[[Author: Edits that TE (and I) made okay above? Kezia]]

You find this request intriguing and sit down with the SOC analysts to get more requirements. After delving into the details with them, you come up with the following objectives:

* The interface must let an analyst choose which port to explore.
* The visualization should—if the metadata is available—identify internal nodes by type (server or workstation) and IP address and also by which data center egress connection attempts were made from.
* External nodes should be easy to identify, with the default direction for graph edges being internal to external.
* The analysts would like to be able to view at least a month’s data at a time.

[AU: Again, revise “badness” to whatever the new language you want to use is. Thanks, Kevin (PJE)]

During your interaction with the analysts, you notice that when they are looking for badness, they often check IP address reputation using external resources. This gives you an idea to have your code perform this lookup ahead of time and color-code external nodes that are found in the AlienVault reputation database. You want to also provide a way for analysts to quickly check all external nodes against other external resources. With the problem domain fairly well defined, you set off to create the tool.

You decide to use Vega for the visualization components and the jQuery (http://jquery.com/) and Opentip (http://www.opentip.org/) JavaScript libraries to add the interactive layer to the core, static Vega visualizations. “Interaction” is just a fancy way of saying “listening and responding to mouse and keyboard events,” something that browser-based JavaScript is very good at. By targeting the browser environment, you can take advantage of all the other open web development resources to help simplify and accelerate the development process. You can also work directly with these events in low-level D3 code.

[[Author: Last sentence above--I assumed you meant “work”? Kezia]]

[[Author, You need a figure reference and some lead-in text for Figure 11-13 so readers know what they are going to be looking at. Thanks, Kevin (DE)

Figure 11-13 The “Threat View” interactive visualization [793725c11f12.png]

The entirety of the code is in the Chapter 11 download materials; it’s contained in the index.html file, in the /support/ch11-threat-view/ directory. Rather than go line-by-line through the file, we’ll highlight some of the core components. The following jQuery routine starts the whole visualization:

[[Author: I made some edits below. Are they okay? Kezia]]

*// The $(document).ready(…) pattern lets us excude a block of code*

*// once all of the HTML in the document has been read and parsed by*

*// the browser. This means we can rely on all the base objects being*

*// ready when we want to start our visualization display*

**$(document).ready(function() {**

*// Opentip is a very flexible tooltip library that we’ll use*

*// to pop-up details of individual nodes on demand*

**Opentip.defaultStyle = "dark"** *// dark-styled tooltips*

*// This tells Opentip to look for mouse events on the vis div*

*// element which can be found in the <body> of the HTML file*

**tip = new Opentip(document.getElementById("vis"));**

**tip.deactivate();** *// hide tooltip for now*

**doParse("22");** *// start visualization with port 22*

**});**

Each “port” visualization has its own pair of files, one for the JSON visualization graph specification (##-vega.json) and one for the actual graph data (##-data.json). This naming convention makes it very straightforward to programmatically change the display—via doParse()—when the port popup registers a new selection.

<div>Select port: <select name="port" onchange="**doParse(this.value)**">

<option value="22">ssh</option>

<option value="23">telnet</option>

<option value="prt">Printers</option>

<option value="161">SNMP</option>

<option value="554">Streaming (554)</option>

<option value="7070">Streaming (7070)</option>

<option value="16464">Port 16464</option>

</select></div>

The doParse() routine does some minor error checking and then calls Vega’s parse() function to do all the work:

**function parse(spec) {**

*// load the visualization specficication (spec) which,*

*// in turn, loads the data file and lets us create the graph*

*// and attach mouse events to the graphic*

**vg.parse.spec(spec, function(chart) {**

*// render the chart in the vis div and give us a handy*

*// reference to it in the graph object*

**graph = chart({el:"#vis"})**

**graph.renderer("svg").update()**

*// when the user mouses over one of the shapes,*

*// build the tooltip on the fly and display it.*

*// tooltips can contain any type of HTML formatting.*

*// here we add whatever metadata we have, including*

*// country flag if available.*

**graph.on("mouseover", function(event, item) {**

**if (item.shape == "circle" || item.shape == "square") {**

**tip.setContent("<div>INFO: " + item.datum.info + "<br/>CC: " +**

**item.datum.cc + " <img src=\"images/flags/png/" +**

**item.datum.cc.toLowerCase() + ".png\"/><br/>DNS: " +**

**item.datum.dns + "<br/></div>");**

**tip.activate();**

**tip.\_storeAndLockDimensions();**

**tip.reposition();**

**tip.show();**

**} else {**

**tip.deactivate();**

**tip.hide();**

**}**

**})**

*// turn off tooltips when the mouse moves out of an element*

**graph.on("mouseout", function(event, item) {**

**tip.hide();**

**tip.deactivate();**

**})**

*// if the user clicks on an external node, look up the selected IP*

*// address on the tcpiputils.com site*

**graph.on("click", function(event, item) {**

**a = item**

**if ((item.datum.group == 4) || (item.datum.group == 5)) {**

**window.open("http://www.tcpiputils.com/browse/ip-address/" +**

**item.datum.name,"\_blank")**

**}**

**graph.update("click", item);**

**});**

**});**

**}**

There are many additions you could make to enhance this basic interactive tool, including:

* Sizing nodes based on the number of connections
* Incorporating other IP reputation resources
* Performing additional metadata queries on internal hosts that have suspicious activity and displaying other layers of information

This should be a good starting point to help you explore both D3 and JavaScript further.

type="note"

You can find the complete working “threat-view” example on the book’s website ( and interact with it by starting the Python web server in the /support/ch11-threat-view/ directory, in the Chapter 11 download materials.

[AU: Edits above okay? Thanks, Kevin (PJE)]

Summary

Creating interactive dashboards and visualizations is a multi-disciplinary endeavor. You must understand both the problem domain and mental models of your consumers, know which goals—augmentation, exploration and illumination—must be accounted for in the finished product, and be certain that interaction is truly necessary for effective communication.

Avoid proprietary solutions whenever possible to ensure your creations can be viewed by the largest audience. Make note of characteristics in other visualizations you find to be effective so you can duplicate their best parts in your own work.

Although there are ways of building useful interactive visuals without coding, you will need to learn the intricacies of modern web frameworks to build highly customized and tailored interactive tools. As you work to fine-tune your finished product, you should endeavor to create a regular feedback loop with those who will end up using your work. This will ensure that you are delivering the most effective tool possible with just the right amount of functionality to make it a success.

Recommended Readings

[AU: Please include a short explanation of the recommended readings below to match the format we’ve settled upon for this section. We’ll have just the title and author here, followed by a short explanation. Then in the References appendix, you’ll include the full bibliographic info for the book. The websites can stand with just short explanations here. Thanks, Kevin (PjE)]

The following are some recommended readings that can further your understanding on some of the topics we touch on in this chapter. For full information on the book included in these recommendations and for any sources we mention in the chapter, please see Appendix B.

***The Design of Everyday Things* by Donald A. Norman**—Insert explanation …

***Interactive Data Visualization for the Web* by Scott** — Insert explanation …

***The Functional Art* by Alberto Cairo**— Insert explanation …

. **“VisAlert: From Idea to Product” by Stefano Foresti and James Agutter**—This is found in *VizSEC 2007*, pp. 159-174. Springer Berlin Heidelberg, 2008. Insert explanation …

*D3 Tips and Tricks: Interactive Data Visualization in a Web Browser* by Malcolm Maclean—See http://leanpub.com/D3-Tips-and-Tricks Insert explanation …

References

[AU: Obviously, you can move the references to the references appendix and delete them here. Please don’t forget to add the ones I suggested by query earlier in this chapter. Thanks, Kevin (PJE)]

1Stefano Foresti, James Agutter, Yarden Livnat, Shaun Moon, and Robert Erbacher, “Visual Correlation of Network Alerts*,” IEEE Computer Graphics and Applications,* vol. 26, no. 2, pp. 48-59, March/April, 2006.

2Barry Schwartz. “The Paradox of Choice: Why Less Is More.” New York: Ecco, 2004.

3Leland Wilkinson. *The Grammar of Graphics.* Springer, 2005.