Chapter 11: Building Interactive Security Visualizations

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 data. In other words, *effective communication* is the primary goal of your visual creations.

As we’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% 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 will help you understand when the move to interactive visualizations makes sense and introduce 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 while providing practical guidance for where to apply interactivity within the scope of information security.

Moving From Static To Interactive

Assuming our “95%” premise holds true, your first instinct when planning out 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 minimal number of dimensions (i.e. 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 to visualization will help 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 give the consumer the ability to explore the relationships and outcomes on their own versus 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 if there truly is badness occurring 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, 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, which 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.

Figure 11.1 The VisAlert Visual Correlation Tool [793725c11f01.png]

It’s fairly straightforward, with tools such as Circos (http://circos.ca/), 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, and it was the detailed focus on the following areas that makes VisAlert 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. While 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 only give you insight into 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. So, while the problem scope is fairly 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 our conceptual models of the way things work or our understanding of how to interact with the world or systems around us. 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 (e.g. inclusion of salient parts of network diagrams and automatically highlighting specific protocols and paths) and automation (e.g. DNS lookups, 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 in a process they describe as a “modified hermeneutic circle”—the movement back and forth between the parts and the whole—which is shown in Figure 11.2.

Figure 11.2 The VisAlert Visual Correlation Tool Design Methodology [793725c11f02.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 one 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 VizAlert tool has been a feature in papers and security-oriented conferences since 2006 but has not manifest itself as a commercial or open source product as of this book’s publication.

Interaction For Exploration

Most networks contain their fair share of vulnerabilities and the Nessus (http://www.tenable.com/products/nessus) vulnerability scanner is one of the de-facto 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.

Figure 11.3 Sample Nessus Detailed Vulnerability Report [793725c11f03.png]

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 the book’s web site—has over 2,000). While it’s possible to spin the data multiple ways and produce reams of static visualizations, this is definitely a perfect example of where an interactive tool could really 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 we’ll be focusing 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.

Figure 11.4 Nessus Vulnerability Explorer Interactive Treemap Interface [793725c11f04.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 (plugin) 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 on-screen 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.

Figure 11.5 Excel Pivot Table With Linked Charts [793725c11f05.png]

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 and 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).

Interaction For Illumination

While everyone may seem to be carrying an i-device of some sort and 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, business processes may be well-understood but the complexity of the entirety of 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 thirty developers, supported by over fifteen operations administrators, span three firewall zones and have components that reside on sixteen 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:// informationisbeautiful.net/).

Figure 11.6 World’s Biggest Data Breaches Interactive Visualization [793725c11f06.png]

Data breaches, as seen 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 reporting almost a breach-a-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.

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 (Figure 11.6) which is 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’s Global Security Report (http://www2.trustwave.com/rs/trustwave/images/2013-Global-Security-Report.pdf) plus online databases such as DataLossDB (<http://datalossdb.org/>) and the Privacy Rights Clearinghouse (http://www.privacyrights.org/data-breach) and now the VERIS Community Database (covered in chapter 7) have covered breaches for many years yet tend to be read and mined mostly by information security professionals. What has made this interactive tool more appealing and useful to a much broader audience than these established resources?

Making 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 real-estate 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 in this chapter introduce some of the technologies that make these visualizations possible, but they will 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 on a web site. 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.

Figure 11.7 The Decline Of Flash [793725c11f07.png]

While Flash still commands a presence on around 17% of web sites (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% share of the web.

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

* the never-ending vulnerability, breach and security update cycle along;
* the rise in popularity of platforms such as the iPad, iPhone and other touch environments that do not provide support for web site elements built with these tools, and
* 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.

Facilitating 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, on-screen world and a fixed graphic. What design choices made the “Breaches” visualization easy to explore?

**Critical exploration elements and operations were 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 were 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**. While 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% undoable, either via the controls on the visualization or with a quick hit 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. 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*2 since our intuition would argue that we 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.

Including 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 would have been completely inappropriate for the mass-consumer audience of the “Breaches” visualization. Rather than bombard the consumer with multi-level taxonomy details, McCandless and Evans opt for simple summaries and succinct descriptions available on-click 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, say, 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 wish to develop interactive visualizations. Most often, you will be required to roll up your sleeves and actually 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 web sites 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

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 (mac support is expected soon) that was heavily influenced by research conducted by Jock Mackinlay in the automation of the design of graphical presentations of relational information (http://cs171.org/2008/papers/mackinlay86.pdf). 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 we look back to 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 to 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 [793725c11f08.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.

Figure 11.9 Awareness Survey Results Presented With Tableau [793725c11f09.png]

After looking at the data (which is available on the book’s web site), 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, used Tableau to create the interactive dashboard shown in Figure 11.9 and viewable at http://public.tableausoftware.com/views/UserAwareness/UserAwareness. The whole process—from data import to finished dashboard—took about twenty minutes.

Figure 11.10 Tableau Details On Demand [793725c11f10.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 be selectable and provides even further levels of detail when inspected (Figure 11.10). 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 web site) 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 wish to create more specialized interactive visualizations or prefer not to be locked into a commercial 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 an entire 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); and,
* have a solid understanding of the Document Object Model (DOM) [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 gathering expertise along the way.

Unlike most proprietary technologies, all D3 visualizations can be dissected and inspected just by doing a “view-source” in 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 very 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>

*// setup 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 setup 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 assignes*

*// 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>

The code is available on the book’s web site 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 &

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

Figure 11.11 Basic D3 Bar Chart [793725c11f11.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 actually 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) and inspect the results (Figure 11.12).

Figure 11.12 Viewing D3 Created Elements in the JavaScript Console [793725c11f12.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”3, then Vega (http://trifacta.github.io/vega/) is D3’s counterpart. 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 all the translating the specification into the appropriate D3 code. To see the difference, compare raw D3 bar chart example above 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"}

}

}

}

]

}

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

Creating An Interactive “Threat Explorer”

The SOC Analysts found the “badness” graphs in Chapter 4 very interesting and asked if there was a way to use the same type of visualization to view which internal hosts are talking to external hosts on a port-by-port basis. This will require adapting the graph code a bit to work with firewall data, but that should be a simple exercise at this point.

You find this request intriguing and sit down with them 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.

During your interaction with them, you also 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 but 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 browser-based JavaScript is very good at. By targeting the browser environment, you are able to take advantage of all the other open web development resources to help simplify and accelerate the development process. You can also with directly with these events in low-level D3 code.

Figure 11.13 The “Threat View” Interactive Visualization [793725c11f12.png]

The entirety of code is contained in the index.html file. 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:

*// 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*

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

*// 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 tool tips*

*// 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 tool tip for now*

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

});

Each “port” visualization has it’s 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.*

*// tool tips 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, lookup 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 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.

You can find the complete working “threat-view” example on the book’s web site and interact with it by starting the Python web server in the threat-view directory.

In 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.

You should avoid proprietary solutions whenever possible to ensure your creations can be viewed the largest audience and make note of characteristics in other visualizations you find to be effective so you can duplicate their best parts in your own work.

While 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 so you can be sure that you are delivering the most effective tool possible with just the right amount of functionality to make it a success.

For Further Reading

Norman, Donald A. *The design of everyday things*. Basic books, 2002.

Murray, Scott. *Interactive Data Visualization for the Web*. O'Reilly Media, 2013.

Cairo, Alberto. *The Functional Art*. Peachpit Press, 2012.

Foresti, Stefano, and James Agutter. "VisAlert: From Idea to Product." In*VizSEC 2007*, pp. 159-174. Springer Berlin Heidelberg, 2008.

Maclean, Malcom. *D3 Tips and Tricks: Interactive Data Visualization in a Web Browser*. http://leanpub.com/D3-Tips-and-Tricks

References

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

2Schwartz, Barry. "The paradox of choice: Why less is more." New York: Ecco (2004).

3Wilkinson, Leland. The grammar of graphics. Springer, 2005.