Chapter 7: Learning From (security) Failures

In times like these when unemployment rates are up to 13%, income has fallen by 5% and suicide rates are climbing I get so angry that the government is wasting money on things like the collection of statistics!

Hans Rosling, quoting a caller on a radio talk show, *The Joy of Stats*

When organizations experience a security event, their natural reaction is to focus on getting back to normal as fast as possible. They see the event as a sign of failure or an embarrassment and everything they do is to minimize the impact and put the event behind them. In that environment, an important task is often overlooked and the silver-lining is often missed. During such an event, a rich set of a data is generated and just waiting to be collected and analyzed. Think of it, if we could somehow gather that data, make sense of it, perhaps even compare and contrast it with other security events, we could learn how we may prevent the next attacker. Maybe even more than that, perhaps we could identify trends and patterns, enough so that we could address multiple common attacks with a single project. Achieving that kind of benefit is the goal of this chapter, we want to figure out what data to collect and how we can collect and manage it. We will also discuss how we can analyze this data and even share it to get the most benefit from the data.

In order to tackle the challenge of learning from breach data, we’re going to leverage the Vocabulary for Event Recording and Incident Sharing (VERIS) framework. One of the authors of this book (Jay) and the RISK team at Verizon have been developing and evolving VERIS in order to produce there Data Breach Investigation Report (DBIR)[[1]](#footnote-1). In an effort to promote adoption and use, Verizon has opened up VERIS and all the details of its use and implementation are hosted at <http://veriscommunity.net>. We won’t just be focusing on implementing VERIS though. We will use it as a foundation and case study as we discuss developing a data collection and analysis effort for breach data.

Besides being an open framework, using VERIS as a case study has another benefit. There is a project called the VERIS Community Database (VCDB), which offers a free and downloadable data set of publicly disclosed security events using the VERIS format. This means we’ll have thousands of VERIS records we can download and analyze throughout this chapter. At the time of this writing, VCDB is being housed at Github (<https://github.com/vz-risk/VCDB>).

Setting up the Research

First and foremost, we want to approach our breach analysis as a research project. If we think of this as a “metrics program” or a “security project”, we may fool ourselves into thinking this is somehow unique to information security and it isn’t. This is all about data collection and analysis, something that has been done countless times before across many different disciplines and generations. Approaching this as if it is a unique project and trying to reinvent the (data analysis) wheel, would not only be wasteful of our time and resources, but we’d be laughed at and ridiculed by all the grown-up data scientists. Let’s avoid all that and call this what it is, a research project.

Most of our work in this book has been of an exploratory nature. We worked with the data to see what it contained and then formed the questions we want to answer with the data and went back into the data. This effort is different because we are starting with no data at all. If we jumped right in and started to collect the breach data that seemed good, we’d waste countless resources, capture data that we’d discover later to be meaningless and we’d end up wishing we had data we didn’t collect. Therefore, we’re going to set a frame for this effort and define a handful of questions we’d like to explore. From that, we will be able to determine what data points we want to collect.

VERIS was developed to support the strategic decision process. Where can we focus our limited resources to get the biggest benefit for our security spending? Given a list of audit findings or remediation projects, how can we prioritize those, so we fix the most critical first? Perhaps even more importantly, we also want to find the opposite of those questions. In other words, can identify areas and tasks where do we *not* want to spend our time and money? Supporting these questions therefore, is our goal in this chapter and can be summarized as:

**Our goal in collecting and analyzing breach data is to support the decision making process within security leadership.**

Notice how used the word “support” in there. This research will exist to *support* a decision process. It is not intended to be or replace the decision process. We need to have the wherewithal to recognize that security prioritization is a complex issue and we are just beginning to scrape away at it. At this point in that scraping, where we have very little data, we should not make the assumption that we’ll get it perfectly right out of the gate. Even though there may be influencing variables that we don’t collect, we have to support the decision process, and we do that by reducing the amount of uncertainty in that decision.

type="note"

Breach Data is for Reducing Uncertainty

While it would be great to collect breach data to create a perfect and prescriptive list of priorities, it just won’t happen. The data will simply help us know a little more than we currently do, but it won’t be able to definitively show us the path forward. This raises the question for some whether or not it is worth it. Is it worth spending the time and resources to create information that doesn’t tell us what to do?

The answer is an emphatic yes.

Uncertainty exists in the gap between what we know and what we would need to know to make the best decision. While it’s tempting to toss out imperfect information because it contains uncertainty, the value of the information should be assessed by comparison. Not between the perfect information we’d want and the information we’ll get, but instead between the information we *currently have* and the information *we will have*. This is where we see the value of this type of data analysis. Data will help us reduce our uncertainty by reducing the gap between what we know and what we need to know. Plus we will be making progress and setting a foundation for further reduction. This is how science has evolved our knowledge: a series of small steps each reducing our uncertainty a little more. Therefore, our goal should be to use data to reduce our uncertainty, not to give up when the data is less than perfect.

Considerations in a Data Collection Framework

As VERIS has evolved, we’ve developed some guidelines that help us as we consider new data points or evaluate the existing questions we’re asking. These guidelines are not just for VERIS or even just for breach data. If you are looking to collect data manually, these guidelines should help shape that effort and make the process easier for all involved.

First and foremost we want to **aim for objective answers**. If we ask questions that require opinion we are going to get a lot of variety in the answers. In some cases this may be okay because inconsistent answers may be better than no answers at all. However most of the time we want to ask things that are observable of deducible. For example, it’s far better to ask if malware was involved in the attack and the functions it performed instead of asking how advanced the malware was. The investigator during the breach can answer yes or no (see below on “unknown”) to whether or not malware was used. If the investigator has the resources to do malware analysis (or the malware is identifiable), there isn’t a lot of guesswork around what’s it’s capable of.

Next we want to **constrain the possible answers** to a short list or small set of options. If we ask for a sentence or description, we should do so knowing that is won’t be useful directly in the data analysis without a lot more effort. Most of the time, free text fields are helpful to record unique aspects or to set context if we ever want to understand why these data points look like they do. With this in mind, we will make judicious use of “notes” fields and a field for the overall “summary” of the event, but all of the real data about the event will be contained in lists or numbers. Having the data constrained to a limited set of values this will make the analysis easier in the long run.

For most every constrained list of answers we create, we’ll usually want to **allow “unknown” and “other” answers**. Even though we may think something is so easy we should always know it, the world will always create a circumstance to prove us wrong. We will want to separate the times we don’t know from the times we know and the list isn’t applicable. This is a subtle distinction, but one that can really mess us up during the analysis. There are a few rare questions we can leave “unknown” off from the answer, but those are rare and you’ll know it when you see it.

The second field we’ll want to add is “other” or depending on the question a “not applicable” answer. We want to avoid from making an exhaustive list. Not only would exhaustive lists become unmanageable (which will slow down data entry), but also we only need to capture *most* of the answers. The common answers create trends and statistics, while the uncommon answers make for interesting stories. Therefore, we want to capture the common things for data analysis and relegate the uncommon to the “other” category and the notes field. We should keep an eye on anything marked “other”, but if we create our list well enough, they should show up few and far between.

It’s okay to be lazy when creating our lists of answers and **seek out standards** to leverage. For example, we will not create our own list of industries to gather. We will leverage the good work of the U.S. Census Bureau and their North American Industry Classification System (NAICS). We won’t attempt to define all the countries, but leverage ISO 3166-1 and store 2-digit codes for the countries. Not only does this offload some of the work, it may also typically be better than anything could dream up.

This last two points may seem subtle, but we want to **avoid conflation and drop the minutiae** where possible. These two concepts are opposites and we have to find the middle ground between them. Conflation occurs when a question (and its answers) are combining more than one concept. For example, the breach types used by Dataloss DB (<http://datalossdb.org/analysis>) conflate the actor, actions and assets into the type. They list a type of “Hack” for a “computer based intrusion” (no asset or actor defined), or “Snooping” which is an “employee … accessing confidential records” (conflating the actor and action) or we can specify “Stolen Media” or “Stolen Drive” or “Stolen Tape”, which are all unique options conflating and repeating the action (physical loss) with the asset. The assignment of a single conflating “breach type” should not be thought of as wrong or bad, it just represents a different goal within the research. Just be aware that conflation of terms like this will create a challenge during data analysis. With conflated terms we will find it challenging to do anything more than simply count the frequency of each breach type.

Where conflation combines more than one concept into a single variable, we have to be careful of the opposite where we split a single concept with minute details. We want to get just enough detail and separation in the list to support our goal. An example of that is when we try to collect how incidents are discovered we want to classify the discovery moethd. While it may be interesting to know if it was an external security researcher, and perhaps amusing to know what color hat they wore (white, black or even grey), those details wouldn’t change our goals. We have split one concept (an external security researcher) across multiple selections. In this case, maybe we just want to drop the distinction of an external security researcher altogether and create one broad field of “an external unrelated party”. But don’t be afraid to go into detail where necessary. As an example, the list of possible assets within VERIS is split into several categories and dozens of detailed assets under each category. There are times we’ll want to split and times we can combine, the trick is getting that balance right.

Luckily, these are all things we’ve been considering as VERIS has been evolving. One of the biggest challenges is saying no to new questions. We’ve found there is always more we’d like to know, but we know that each data point we try to collect has a cost (see below).

type="note"

Consider the Cost per Datum

During a manual data collection effort, it is very tempting to dream up all sorts of questions we’d like answered. Creating such a list isn’t bad and it may even be good to lay out all the questions you’d want to answer. But choose the questions you are going to ask very carefully because every question adds exponential cost across the lifetime of the data. Even before the question gets answered we have to build a method to collect it, so every question must be built into the data collection. We may have data validation going into the method, and then data validation coming out. As a single incident is being entered, each question will require some thought and perhaps even some research before it can be answered, again adding time and effort. That data point may require processing and clean up, and will need to stored and managed. Anytime we want to parse the data (and you’ll want to parse this in many different ways), we may have to consider this field, or worse, consider the interactions of all the fields. Beyond that, there are dozens other subtle interactions that will increase the cost of each data point beyond what we can imagine as we form the research questions.

It’s helpful to pretend you are about to take a long journey to a wise sage who lives on top of a mountain. You will have a limited amount of time to ask questions before the sage says something mysterious and vanishes. What questions will have the greatest impact? You’ll want to identify a handful of questions you really want answered, maybe a handful of questions you’d like to have answered and then you’ll have a mountain of questions you wish you had time to ask, but you’ll just have to make do. The same is true with manual data collection. If the post-incident questionnaire asks too many questions or is too painful, people will lose interest quickly and the answers will end up being of poor quality. You must choose your questions wisely.

An Introduction to VERIS

When there is a security event investigated, a narrative naturally emerges from the process. The investigator will typically try to answer, “Who did what to what (or whom) with what result?” which is a good core set of data points to collect. Therefore, as a starting point, we want to focus on those four points, “Who (threat actor) did what (action) to what or whom (asset) with what result (attribute)?” But that’s not all we may be interested in; we may also want to know how we discovered and responded to the incident and if possible the impact we experienced as a result. Finally, we’ll have some housekeeping items (an identifier, summary, status, etc) and if we aggregate breaches or may share the information, we’ll want to record some victim demographics. Overall, we can break down the following sections of data we want to gather at a minimum.

|  |  |
| --- | --- |
| VERIS Section | Purpose |
| Incident Tracking | Meta-data about the incident for management and tracking purposes. |
| Threat Actor | One or more people that that cause or contribute to an incident |
| Threat Actions | What the threat actor(s) did or used to cause or contribute to the incident |
| Assets | Information assets that were compromised or affected during the incident |
| Attributes | What was affected about the asset during the incident |
| Discovery/Response | Timeline, discovery method and lessons learned |
| Impact | What was the overall effect of the incident to the organization |
| Victim | Demographic information like industry and organizational size |
| Indicators | Optional indicators of compromise (ip addresses, malware hashes, domains, etc.) |
| Plus | Optional section for extending VERIS |

Table 7.1: Sections within VERIS

While it’s tempting to dig into the data (and we will), it’s important to understand the significance of these fields so we don’t misapply them. Therefore we will go through each part of VERIS in more detail and discuss the fields in each section. Keep in mind that the separation of these sections is for people to think about the structure, in the data there is nothing denoting the “incident\_id” field as helping with incident tracking for example.

type="warning"

While we are covering VERIS with some depth, we will not go into every field, and we won’t be able to cover every detail about the framework. For example, we won’t call out all the places the framework specifies a “notes” field (which is almost every section), and we won’t cover the indicators section in detail. Just keep in mind, that the framework is actively maintained and evolving. This chapter is discussing the 1.2.1 release, so be sure to refer to <http://veriscommunity.net/> for all the details and current specification of the VERIS framework.

Incident Tracking

Some of the fields within VERIS exist to simply describe or track the incident. These fields help us keep records straight by identifying each with a unique identifier, tracking the source of the incident and any related incidents. You’d use the source\_id field to compare your unique “source” of incidents to something like the VCDB (which has “vcdb” in that field). Required fields are marked. If something has a value of “**factor**” that means it is a restrictive list and only those values are expected.

|  |  |  |
| --- | --- | --- |
| Field | Value | Description |
| schema\_version | string | VERIS version (currently 1.2.1) |
| incident\_id | string | unique identifier (VCDB uses GUID) |
| source\_id | string | origin of the data (VCDB data has "vcdb") |
| reference | string | URL or internal ticketing system ID |
| security\_incident | factor | Confirmed, Suspected, False positive, Near miss |
| confidence | factor | High, Medium, Low, None |
| summary | string | free text summary of incident |
| related\_incidents | string | free text, other incident\_id’s |
| notes | string | free text |

Table 7.2: Incident Tracking Fields

Just looking down the list, there is only one or two fields here we will use during analysis and those are the two “factor” variable (again, this means they are restricted to a list of expected answers). The security incident is required and will help us split our analysis on whether the event was a confirmed security incident (an asset has a security attribute affected) or not. The confidence rating is a rare subjective field. It enables the analyst to record their subjective assessment of how confident they are in the accuracy of the data they entered. This optional field is not heavily used and won’t appear much in the VCDB incidents we’ll look at.

Threat Actor

Earlier in this chapter we talked about the challenge of conflation. This is something we want to be aware of especially in these next three sections (actor, actions and assets). We talked about the framework DataLoss DB used with a single conflated breach type and we see the same thing with the framework used by Privacy Rights Clearinghouse. Their framework also uses a singular “breach type” to define each event and again it will mix in the actors and actions into the one label. For example, they have an “Insider (INSD)” type, which is defined as “Someone with legitimate access intentionally breaches information - such as an employee or contractor.” And then a type of “Physical loss (PHYS)” which is defined as “Lost, discarded or stolen non-electronic records, such as paper documents.” These simplified labels can quickly become confusing during data entry if, for example, an insider steels paper documents. We may see insiders breach systems, drop malware and social engineer, just as an external actor would and we want to separate the two clearly in the data. VERIS tackles that by separating who from what they did and what was affected. We should mention that we shouldn’t think of the method Privacy Rights Clearinghouse uses as right or wrong. It just has a different focus and represents different priorities and goals. We would like to inform and support security decisions, which benefits from more detail than a single “breach type” label.

|  |  |  |  |
| --- | --- | --- | --- |
| actor | Field | Value | Description |
| external | motive | factor | Helps understand intentions, same enumeration for all instances |
|  | variety | factor | Shapes resources, capability of external actor |
|  | country | factor | ISO-3166-1 2-digit country field |
| internal | motive | factor | Helps understand intentions |
|  | variety | factor | Shapes resources, capability of internal actor |
| partner | motive | factor | Helps understand intentions |
|  | industry | string | U.S. Census NAICS code |
|  | country | factor | ISO-3166-1 2-digit country field |

Table 7.2 Threat Actor Fields

The threat actor section also introduces us to the nesting feature of VERIS. At the top level we are talking about the actor, so we have a section in the data for “actor”, then there are three classes of actors defined, external, internal, and partner, all of which are optional. Within each of those classes we want to add details about that type of actor. Looking down the values we have in this section we see all factors. That means we should be able to include any of these or use them as pivot points. In other words, if we want to support a threat modeling exercise that is comparing different threat communities, we could extract the actions for financially motivated actors and compare that to disgruntled employees.

Figure 7.1 Known Motives Across All Actors (percentage of events) [FILENAME 793725c07f001]

Threat Actions

This section collects variables to describe what the threat actor(s) did or in some cases, used during the event. Again we have nest variables under top level categories which are:

* **malware**: malicious software, script, or code run on an asset that alters its state or function
* **hacking**: person (at a keyboard) attempting to access or harm an asset without authorization
* **social**: exploiting the human element (phishing, pretexting, etc)
* **misuse**: abusing resources or privileges contrary to that which was intended
* **physical**: personal actions involving proximity, possession, or force
* **error**: anything done (or left undone) incorrectly or inadvertently
* **environmental**: natural events and hazards within the immediate environment or infrastructure of assets

We have to be careful as we work with these categories. There are many opportunities for misinterpretation and misclassification across categories. These categories and the factors in each category are explained in detail along with use case examples at the VERIS website (<http://veriscommunity.net/>). Once you spend some time and look at a few examples though, these get to be a bit easier and eventually will become intuitive.

|  |  |  |  |
| --- | --- | --- | --- |
| action | Field | Value | Description |
| malware | variety | factor | functionality of malware |
|  | vector | factor | how the malware was installed/infected |
| hacking | variety | factor | type(s) of hacking action |
|  | vector | factor | path of attack |
| social | variety | factor | type(s) of social action |
|  | vector | factor | path or method of communication |
|  | target | factor | role of targeted person |
| misuse | variety | factor | type(s) of misuse action |
|  | vector | factor | path or access method for misuse |
| physical | variety | factor | type(s) of physical actions |
|  | vector | factor | method of physical access |
|  | location | factor | physical location of action |
| error | variety | factor | type(s) of error actions |
|  | vector | factor | cause of error |
| environmental | variety | factor | type(s) of environmental actions |

Notice how “variety” and “vector” are repeated over and over? Every action category has variety field with unique enumerations for each category. All but the environmental actions have a vector field, again with unique enumerations for each category. Finally social actions also ask for the target of the social action and physical actions ask for a location of the action and that explains the whole section! Again notice how every field here is a factor, meaning we’ll be able to split, pivot and/or filter based on these fields.

Figure 7.2 Top 10 Varieties of Threat Actions [FILENAME 793725c07f002]

type="note"

Multiple Events in the Attack Chain

Anyone who has been around information security knows that breaches tend to not be simple and single events. Often times the attacker will perform multiple actions and this complicates the recording process. Most of the factors in the VERIS framework support multiple answers. On one hand, this is very nice, because we don’t have to pick “the one best answer” for a complex security event, but on the flip side, this adds complexity for data management and analysis. As we’ll see later in this chapter, this isn’t as hard as it first seems.

As an example, suppose an attacker sends a phishing email to an executive’s assistant and quickly follows that up with a phone call pretending to be a business partner who sent the email. These are two actions and we should see both “Pretexting” and “Phishing” selected in the social.variety field. If the phishing email contained malware that is installed, we’d also see the malware action along the variety and vector of “Email”, since it was installed via the phishing attack. When we represent this data, if we count up the actions, we’ll usually have more actions then events. This naturally precludes the use of pie charts, which ends up being beneficial for all parties involved.

There is also a common notion within information security of the “attack chain” or “kill chain”. The concept is to establish the actions of the attacker in the order they happened. While VERIS allows multiple actions, it does not record the order they occurred in. This was a conscious trade off of cost versus benefit. Attempting to put the events in order created substantial overhead for the analysts and was taking too long to enter. Most of the time the order of attacks in reports and tickets (definitely in media articles) are either vague or missing completely. As a result VERIS simply records the presence of attacks in order to reduce the amount of effort during data collection.

Information Assets

Assets are the information containers (servers or other devices) that we are trying to protect. Like the others we have a top-level category, which are as follows:

* Server (S): system providing service(s)
* Network (N): infrastructure device or appliance
* User Device (U): end user equipment (laptop, desktop)
* Media (M): data storage devices or physical documents
* People (P): since people can be affected
* Kiosk/Public Terminal (K): public-use devices

Within each category there are several varieties of assets, but the category and variety are stored in the same field. For example a Mail Server is stored as “S - Mail” and a desktop computer is a “U - Desktop”. Associated with each asset is an optional “amount” field, which allows us to record multiple assets with the same variety when they are involved in one event.

|  |  |  |  |
| --- | --- | --- | --- |
| asset | Field | Value | Description |
| assets | variety | factor | specific type of asset, prepended with letter for category |
|  | amount | integer | count of the above asset |
| asset | accessibility | factor | how accessible the assets are |
|  | ownership | factor | who owns the assets |
|  | management | factor | who manages the asset |
|  | hosting | factor | where (physically) is it hosted |
|  | country | factor | location of assets (if different from victim) |
|  | cloud | factor | type of cloud service, if cloud |

There is quite a bit packed into the assets and these are relative recent additions to the VERIS framework. There is a lot of focus around mobile devices and employees bringing their own device into the corporate environment. Also, there may be unique exposures from cloud hosted applications and assets, so that will be captured here as well. Note also that these are all factors so there are only a handful of possible answers, we cannot write in “very” for accessibility of the asset as an example.

Figure 7.3 Asset Categories [FILENAME 793725c07f003]

Attributes

The attributes of the above assets are what we work hard in information security to not have affected. Attributes are based on the C.I.A. triad, which stands for confidentiality, integrity and availability. For a while VERIS extended these three with three more attributes to record the Parkerian Hexad (named after their originator, security pioneer and long time security researcher, Donn Parker). The extra three attributes included possession and control, authenticity and utility. But the added fields just did not yield enough benefit for the added cost of separate categories so they were combined with the three top categories. For simplicity, when a VERIS record is stored, the sections are just labeled with the three primary categories (in bold below). The three main sections of attributes are:

* **confidentiality**, possession and control: data was observed or disclosed to an unauthorized actor, owner may no longer have exclusive custody
* **integrity** and authenticity: asset is incomplete or changed from authorized content and function, conforms to expected state
* **availability** and utility: asset is not accessible, useful or fit for use

The use of these categories can be quite helpful in separating out the areas to focus on for a security team. The Verizon Data Breach Investigation report has exclusively focused only on breaches where the confidentiality attribute was affect and there was a confirmed data disclosure.

|  |  |  |  |
| --- | --- | --- | --- |
| attribute | Field | Value | Description |
| confidentiality | data\_disclosure | factor | status of confidentiality breach |
|  | data\_total | integer | number of records (see below) |
|  | data.variety | factor | type of data disclosed |
|  | data.amount | integer | number of records |
|  | state | factor | state of data when disclosed |
| integrity | variety | factor | nature of effect |
| availability | variety | factor | nature of effect |
|  | duration | time range | duration of availability/utility loss |

There is a new field type here of “time range” which is actually two fields, a “unit” of time and a “value” for that unit of time. The unit has general measurements of time: seconds, minutes, hours, days, weeks, months and finally years. The value represents how many of those, so 3 weeks or 6 months. Within VERIS it was found that knowing a specific amount of time first was often difficult to get precisely and this method of generalizations was usually possible to discern between. As an example, it may be known that the server went offline during the DDoS attack, but the specific quantity was more than 60 minutes and definitely not a full day. In that case, we would see “hours” in the unit, and if the specific number of hours is known we’d see a value, otherwise it may be blank if the precision is unknown.

type="note"

Counting records

One of the more common pieces of information in publicly disclosed breaches is the number of records. Perhaps reporters and the general public demand this and the victims are forced to provide a number, even if it’s all the records in the database. Records can be relatively easy to count when the data comes with obvious separation. Payment (credit) cards, identities or medical records are quite clear in their separation and lend themselves to being counted. But when we get into more complex types like classified information or trade secrets, the ability to count records becomes a little less defined. Perhaps the number of physical documents could be used, or number of files disclosed, but oftentimes it’s difficult to count them. Overall, analysts struggle to record a precise number for the data varieties of classified or internal information and trade secrets and we will have to account for that in the analysis and visualizations.

Discovery/Response

We just saw our first time range in the availability attribute and we’ll see that a lot more in this section for the timeline data. Some of these fields are not in a section like the previous four sections, but the timeline does have its own section.

|  |  |  |  |
| --- | --- | --- | --- |
| section | Field | Value | Description |
|  | discovery\_method | factor | how event was discovered |
|  | control\_failure | string | free text field to describe what, under the victims control, failed |
|  | corrective\_action | string | free text field, describing what the victim should do |
|  | targeted | factor | targeted or opportunistic attack |
| timeline | incident\_date | date | date of incident |
|  | compromise | time range | time to initial compromise |
|  | exfiltration | time range | time from initial compromise to data exfiltration |
|  | discovery | time range | time from initial compromise to discovery |
|  | containment | time range | time from discovery to containment |

We have a new field of “date” here and this is not a standard date field. Because VERIS has to account for unknown values the date fields are in separate variables. Too often then we’d like for our analyses, the precise date of the incident isn’t known or isn’t reported clearly. The framework assumes at least the year is known, but the month, day and time fields are all optional in that date field. The other fields in the timeline are the same time range values we saw in the availability attribute.

Notice also that the control failure and corrective action suggestions are free text. This makes them difficult to include in our data analysis without more effort. Finally, the discovery method is one of the rare enumerations that cannot have multiple answers. The framework assumes that once the incident was discovered it could not be discovered again, so only one method of discovery is allowed.

Impact

The impact section is perhaps one of the most, if not the most, sparsely populated section in the incidents. This has nothing to do with the framework and everything to do with the lack of accurate data to collect and record about the impact. The result is that this section has some subjective measurements and estimates.

|  |  |  |  |
| --- | --- | --- | --- |
| section | Field | Value | Description |
|  | currency | factor | ISO 4217 currencies for monetary estimations |
| overall | rating | factor | qualitative rating of overall impact |
|  | min\_amount | number | min estimated monetary amount |
|  | amount | number | most likely estimated monetary amount |
|  | max\_amount | number | max estimated monetary amount |
| loss | variety | factor | specific category of loss |
|  | rating | factor | qualitative rating of overall impact |
|  | min\_amount | number | min estimated monetary amount |
|  | amount | number | most likely estimated monetary amount |
|  | max\_amount | number | max estimated monetary amount |

Notice the repeating rating and monetary estimations. There is a dedication “overall” field here for those fields, but the loss section is defined in the data as an array. This means that the analyst can add multiple loss sections in the data for each variety of loss being recorded. The loss varieties are specific types of loss, for example “response and recovery” costs or “legal and regulatory”.

Victim

The last section we should cover is the victim section. If VERIS is being implemented inside a single organization, the fields in this section could be skipped (or hard-coded) since the victim will always be the same. But for cases like the VCDB where it is aggregating across many victims, this section is vital. We want to capture data about the victim with the intention of contrasting and comparing breach data when we split on these fields.

If you remember back to Chapter 5 where we touched on regression analysis, we attempted to find independent variables that could help describe the outcomes we observed. The data we are collecting about the victim can go a long way to describe the types of threat actors and their actions. For example, in the 2013 DBIR, Verizon saw state-affiliated espionage in at least 3 out of every 4 cases within the manufacturing industry and yet none in the retail industry. While industry alone is not a perfect predicting variable, it does help reduce our uncertainty, and remember that’s what we are after here.

|  |  |  |  |
| --- | --- | --- | --- |
| victim | Field | Value | Description |
| victim | victim\_id | string | identifier or name of victim |
|  | industry | string | U.S. Census NAICS code |
|  | employee\_count | factor | Label for number of employees |
|  | country | factor | ISO 3166-1 2-digit country code |
|  | state | string | state, province or region in country |
|  | locations\_affected | integer | number of locations affected |
|  | revenue | integer | annual revenue of the victim |
| secondary | victim\_id | string | list of secondary victim\_id or name(s) |
|  | amount | integer | and/or count of secondary victims |

The most recent change to the VERIS framework (version 1.2.1) changed how this section is stored. In version 1.2 and before, the entire victim section could be repeated for each victim involved in the incident. For example, if an organization is breached and they were processing data on behalf of another organization, they would become a victim of the same breach. This was found to be confusing though and the victim was reduced to just supporting one single victim. The fields in the “secondary” section where added in 1.2.1 to capture what was treated as a multiple victim breach.

Figure 7.4 Top 5 Industries in VCDB dataset [FILENAME 793725c07f004]

Anywhere we have an industry (which is here and in the threat actor partner section), they are listed as a “string” but they should not be free text. Following one of our guidelines to leverage other resources wherever possible, VERIS leverages the U.S. Census Bureau’s North American Industry Classification System (NAICS). Doing so adds flexibility and a level of detail not possible with other industry classification systems. If people were going to create a list of industries, they’d probably come up with a dozen or so high level categories and call it good. NAICS started there (20 top-level categories actually), but then made it extendible and enables more and more detail to be put into the industry specification. Industries within NAICS are represented by a 2 to 6 digit integer which is why VERIS stores them as a string and not a factor, the list is enormous (but is listed at http://www.census.gov/eos/www/naics/).

As an example, let’s take the pizza shop down the street. The NAICS code for that is 722511 which represents “Pizza Parlors, full service.” But sometimes, maybe the analyst just knows it’s a restaurant, then they would just record “7225”, or maybe they know the victim offers some type of food or beverage service, then they may enter “722”, but if they are really unsure exactly what type of service establishment it is, we may just see “72” for “Accommodation and Food Services”. When we do analysis on this field we can drill down or up depending on the level of detail we want. But be careful, the more detail we put into this field, the smaller we will be dividing up our samples.

Extending VERIS with Plus

Finally we have the catch all section labeled as “plus”. Within the VERIS framework there technically is nothing specified in this field and the data schema simply allows anything to exist in this section. It exists to allow individual implementations to record additional fields not in the base VERIS schema. If we look at the VCDB repository for example, each incident has a plus section with the analyst who recorded the incident and the time it was created along with a few other fields being considered. Any implementation can apply the guidelines (or not at your own peril) and add their own fields here. If they fields are useful, feel free to suggest the change to the core framework!

Seeing VERIS in action

It’s always nice to take some time before we jump into the analysis to look directly at the data. It helps set the context in our mind and may help shape our approach to the analysis. Since the average incident is about 100 lines of JSON, we won’t include the whole incident. Please take some time to surf around the VCDB repository and look at the data there for full records. As a good example, here are the actor and action sections from an incident from VCDB:

"actor": {

"external": {

"country": [ "SY" ],

"motive": [ "Ideology" ],

"variety": [ "State-affiliated" ]

}

},

"action": {

"hacking": {

"variety": [ "Use of stolen creds" ],

"vector": [ "Web application" ]

},

"social": {

"target": [ "End-user" ],

"variety": [ "Phishing" ],

"vector": [ "Email" ]

}

}

If you have never seen JSON before, this is what it looks like. Rarely if ever would we want to edit the JSON by hand. It’s not that JSON is terribly difficult, but it is terribly easy to mistype something, forget a comma, a quote or something that would prevent the data from loading properly. If you do attempt to create or modify a JSON file by hand, be sure you have a way to check your work, validate the JSON and if possible, validate the values and factors within the data.

The best part about working with JSON is that it typically imports right into native objects in the languages we use. Within python, an incident in JSON is imported directly to a python dictionary. The code to load up a JSON object and view the hacking variety in this object is relatively simple:

# python to load JSON and read hacking variety:

import simplejson as json

# Open the JSON file and read the raw contents into jsondata

jsondata = open("some\_veris\_file.json").read()

# convert the contents into a python dictionary

incident = json.loads(jsondata)

# now access the hacking variety:

print(incident['action']['hacking']['variety'])

Which would print out the python list object for hacking variety and display ['Use of stolen creds'] the example above. In reality we should wrap the json.loads() command with try-except, if the file has any errors in the JSON syntax, they will be caught that way. Plus the hacking action is optional, and we’d want to test if the “hacking” key existed before we attempt to read it. But it helps show how easy JSON can be to load and work with.

Within R, json files are converted to a native list object. Performing the same function of loading the file and printing the hacking variety is done like this:

# use the rjson library

library(rjson)

# fromJSON accepts a filename to read from

incident <- fromJSON(file="~/Documents/json/newfinal/jay.json")

# print the hacing variety

print(incident$action$hacking$variety)

[1] "Use of stolen creds"

The R code returns a one-element vector with the value in the hacking vector. Again, in full-featured code, we’d want better error checking than this, but it does show how easy this data is to load into native objects and work with.

Working with VCDB Data

Before you can follow along in your own environment, you’ll need to grab VERIS data. Head on over to the VCDB github repository at <https://github.com/vz-risk/VCDB> and either fork, copy or download a zip file of the repository. The incidents themselves are quite small, but we’ll manage to still learn quite a bit in spite of the data not being “big”. Feel free to explore the incidents in the repository and get a feel for the files and the data. Keep in mind that all of these incidents are collected from publicly disclosed events, which makes some of the incidents rather light on the details.

For our analysis we are going to leverage the “verisr” package, which was developed by our own Jay Jacobs and is in his github repository (found at https://github.com/jayjacobs/verisr). We should also point out that the verisr package is actively in development, be sure to refer to the latest documentation of the package for the most current description of its functions. By the time you are reading this, there will undoubtedly be all sorts of wonderful features in the package that aren’t there at the time of this writing.

In order to install the verisr package from Github, you will have to load up the “devtools” package first. This is one of many great packages from Hadley Wickham and it allows us to install R packages directly from their Github repository, which is what we’ll do with verisr.

# load up devtools

library(devtools)

# install the verisr package

install\_github("verisr", "jayjacobs")

# load the versr package

library(verisr)

We can now load up the VCDB data with the verisr package:

# set this to the location you stored the VCDB files:

jsondir <- '../VCDB/incidents'

# create a veris instance with the vcdb data

vcdb <- json2veris(jsondir)

This should load up fairly quickly, but on slower machines it may take a second or two. If you’re on a computer with very little RAM or if VCDB grows exponentially, you may not be able to load all of them into memory. But we’ve loaded over a hundred thousand incidents into verisr, we shouldn’t hit that limit anytime soon. Now that we have loaded up this data, we should get to know the data a little bit. Let’s begin with the summary() command. The verisr package implemented it’s own summary(), so the output is very specific to VERIS data.

summary(vcdb)

## 1737 incidents in this object.

##

## Actor:

## external internal partner unknown

## 1024 556 104 86

##

## Action:

## environmental error hacking malware misuse

## 1 413 466 49 225

## physical social

## 520 33

##

## Asset:

## Kiosk/Term Media Network Person Server Unknown

## 18 546 10 37 724 86

## User Dev

## 447

##

## Attribute:

## availability confidentiality integrity

## 639 1690 190

If you’ve grabbed the latest data from VCDB, you’ll undoubtedly see different numbers than this.

type="note"

JSON Notation

It may take a while to get used to the naming structure in JSON and how the variables are accessed in different settings. If we load VERIS JSON data into a mongo database, we’d use Javascript to query the data and leverage a dot-notation approach to the variables. That dot-notation is something used in the verisr package since the fields are referenced and retrieved by passing in character strings. This means we can access the top-level action data by just referencing “action”. If we want to access the social section within the action we reference “action.social” and the variety data under that is “action.social.variety”. Take some time, look at the raw json and try different approaches in python and verisr and the naming will become second nature.

There are two main functions from the verisr package that we’ll use to dig into the data. The first is a function to create a filter so we focus in certain aspects of the data. The second is a versatile function called getenum(), which will get the enumerated data from the dataset with a variety of options and extensions. Let’s start by looking at the actors. We can replicate the information in the summary above with the following bit of code.

# we should already have verisr loaded and the vcdb object created.

actors <- getenum(vcdb, "actor")

# actors is a data frame

print(actors)

## enum x

## 1 external 1024

## 2 internal 556

## 3 partner 104

## 4 unknown 86

Within this data.frame, we can see the raw numbers, but that isn’t all that helpful. Some incidents will contain multiple actors, so we can’t simply add these up and get a total number of incidents. Luckily, the getenum function can also return the total number of incidents where the field is defined. If we add the “add.n=TRUE” we will get an additional column of the full sample and if we add “add.freq=TRUE” we can the percentage associated with each entry. Let’s look at both of those options in one example.

# ask for the total incidents (n) and percentage (freq)

actors <- getenum(vcdb, "actor", add.n=TRUE, add.freq=TRUE)

print(actors)

## enum x n freq

## 1 external 1024 1737 0.59

## 2 internal 556 1737 0.32

## 3 partner 104 1737 0.06

## 4 unknown 86 1737 0.05

From this we can see that there were 1737 incidents with something defined in the actor section, and external actors were present in 59%. Since this function returns a data frame to us, it’s relatively straightforward to feed into the ggplot2 library and produce any number of visuals from (see our website for how we create the bar charts earlier in this chapter).

The getenum() function is quite versatile, we can pass in any of the variable names within the VERIS framework and get an object we can visualize right away. As an example, let’s create a function that accepts a VERIS variable name, like “action.hacking.vector” and returns an image object we can print or save or whatever. This could be extendable to include in a report or dashboard.

# load ggplot2

**library(ggplot2)**

# take in the vcdb object and the field to plot

**verisplot <- function(vcdb, field) {**

# get the data.frame for the field with frequency

**localdf <- getenum(vcdb, field, add.freq=T)**

# now let's take first 5 fields in the data frame.

**localdf <- localdf[c(1:5), ]**

# add a label to the data.frame

**localdf$lab <- paste(round(localdf$freq\*100, 0), "%", sep="")**

# now we can create a ggplot2 instance

**gg <- ggplot(localdf, aes(x=enum, y=freq, label=lab))**

**gg <- gg + geom\_bar(stat="identity", fill="steelblue")**

# add in text, adjusted to the end of the bar

**gg <- gg + geom\_text(hjust=-0.1)**

# flip the axes and add in a title

**gg <- gg + coord\_flip() + ggtitle(field)**

# remove axes labels and add bw theme

**gg <- gg + xlab("") + ylab("") + theme\_bw()**

# fix the y scale to remove padding and fit our label (add 7%)

**gg <- gg + scale\_y\_continuous(expand=c(0,0),**

**limits=c(0, max(localdf$freq)**\*1.07))

# make it slightly prettier than the default

**gg <- gg + theme(panel.grid.major = element\_blank(),**

**panel.border = element\_blank(),**

**axis.text.x = element\_blank(),**

**axis.ticks = element\_blank())**

**}**

What’s a little funny about that function is that we get all of our data ready in the first line of the function, trim to the to the top 5 entries in the second line, and spend the rest of the function making a pretty picture. But once this is written and loaded up, we can create any number of pictures from our data with a single line of code.

print(verisplot(vcdb, "action"))

Figure 7.5 shows a few of the possible values passed and printed.

Figure 7.5 Various Top 5 views of VCDB data [FILENAME 793725c07f005]

Getting the most out of VERIS data

One of our favorite images from the 2013 Verizon Data Breach Investigation Report was figure 8. It was a heat map that compared the assets and actions overall and then separated out individual comparisons by the type of threat actors. We can create a similar image with the verisr package without too much effort.

# get a data.frame comparing the actions to the assets

# this will add zero's in missing squares and include a frequency

**a2 <- getenum(vcdb, enum="action", primary="asset.assets", add.freq=T)**

# trim unknown asset and environment action for space

**a2 <- a2[-which(a2$enum=="environmental" | a2$primary=="Unknown"), ]**

# so we should create a "slim" version without zeros to color it

**slim.a2 <- a2[-which(a2$x==0), ]**

# now make a nice plot

**gg <- ggplot(a2, aes(x=enum, y=primary, fill=freq))**

**gg <- gg + geom\_tile(fill="white", color="gray80")**

**gg <- gg + geom\_tile(data=slim.a2, color="gray80")**

**gg <- gg + scale\_fill\_gradient(low = "#F0F6FF",**

**high = "#4682B4", guide=F)**

**gg <- gg + xlab("") + ylab("") + theme\_bw()**

**gg <- gg + scale\_x\_discrete(expand=c(0,0))**

**gg <- gg + scale\_y\_discrete(expand=c(0,0))**

**gg <- gg + theme(axis.ticks = element\_blank())**

# and view it

**print(gg)**

This will look through all of the incidents and produce the simple colored heat map in figure 7.6. Keep in mind the specifics will vary depending on incidents in the VCDB.

Figure 7.6 A2 Grid Comparing Assets and Actions [FILENAME 793725c07f006]

The real benefit of working with VERIS data is the ability to compare across disparate data sets. If you were to collect your own internal incidents in the VERIS format, it would be a relatively trivial task to run comparisons on very specific slices of data across multiple data sets. Since one of the authors works for Verizon and has access to the DBIR data set, we decided to show this point by example. We should be able to quickly see differences across the two data sets. Remember, VCDB is collected from news articles and various public sources. Generally speaking the details are far less than what we would hope. The Verizon data set is gathered from a variety of primary sources, but primarily from a first hand account of the forensic investigators that were brought in after the security event. This means this data has a bias, it is generally limited to breaches that were complex or big enough for a victim to seek external help, either from law enforcement (many of the contributing partners are law enforcement), or from an incident response consulting company.

Let’s use the same code that generated figure 7.6 and compare four different fields from the VCDB data and the Verizon DBIR data over the last three years. Let’s start with all of the incidents in both data sets in the first row. Then filter out for just confirmed data loss events (where attributes.confidentiliaty.data\_disclosure = “Yes”) in the second row. Then let’s focus on financially motivated attackers with confirmed data loss events in the third row and finally, let’s just look at attackers motived by ideology, curiosity, fun or pride (which covers attackers labeled as “activists”), again with confirmed data loss.

Figure 7.7 The Strength of VERIS: Comparing the same views from to very different sources of data. [FILENAME 793725c07f007]

We can see a rather significant difference that would be worth talking through. Since the VCDB data set is all publicly disclosed events, there are a lot of daily “low hanging fruit” type things that would never make it into the DBIR data. Events like simple lost or stolen laptops, documents tossed in a dumpster without being shredded or envelopes with personal information mailed to the wrong person appear quite often in the VCDB data set. That’s why we see public (theft/loss) and error (disposal error and misdelivery) in the first row for VCDB, and those all but disappear when we filter for confirmed data loss in the second row. Keep in mind a lost or stolen laptop has the potential for data loss.

Another interesting comparison is the malware category. The public disclosures will rarely mention if malware was used, but we know from the DBIR research that malware is often used, either to escalate privilege or to capture and exfiltrated data, yet the malware column is almost completely empty in the VCDB data. We are probably seeing the same type of effect for the user devices. We know that the user device is often leveraged in a breach, but when a company is publicly disclosing information, they’ll leave off what assets were involved and just say something vague like “our database was compromised.” As a result, we see very little recorded events involving user devices.

We could probably go on and on about the subtle differences across these two columns, but it’s pretty clear there is a lot to be learned by recording and comparing breach data.

In Summary

We may never be able to shake the “blame the victim” mentality when it comes to data breaches. This means the victims will always try to be discrete and put their focus on getting back to normal. And that means we may always be fighting for more disclosures and more data when it comes to security breaches. But that is exactly what we need because these events produce a very rich set of data that has yet to be fully explored.

When we break the event down to its atomic components, “Who did what to what (or whom) with what result?” we will be able to do more research, better comparisons and learn so much more than if we applied a label or two on the whole chain of events. Identifying and recording the actor, their action, the assets involved and the attributes affected are a very good start. But remember, every data point comes with a cost and you will have to make some tough trade offs between the time investment and the benefits you’ll be capable of with the data.

Using JSON has some direct advantages, we can quickly load it in a variety of languages and it feeds right into databases that can take JSON (like mongodb). Within R we can use the verisr package to read in VERIS data and rapidly analyze fields and create visualizations. But the real strength of leveraging a framework like VERIS is when are able to make comparisons. Are you unique in your problems? Are others in your industry or across all industries seeing the same trends and attacks? Until recently we would struggle to answer those questions, but as more organizations take a data-driven approach to security we’ll be asking and answer those questions soon.

Recommended Reading

Any Verizon Data Breach Investigation Report at http://www.verizonenterprise.com/DBIR

Categorical Data Analysis by Alan Agresti

<http://veriscommunity.net>

<https://github.com/vz-risk/VCDB>

<https://github.com/jayjacobs/verisr>

1. The Verizon Data Breach Investigations Report leverages the VERIS framework for its data collection and data analysis and may help the reader get a context for this chapter. The most recent report can be found at www.verizonenterprise.com/DBIR/ [↑](#footnote-ref-1)