An Integrated Visualisation Framework for Intrusion Detection

Huw Read, Andrew Blyth

Abstract—This paper builds upon earlier work [1][2] regarding the need for advanced visualisation techniques applied within the intrusion detection arena. Individual visualisation tools can tell us a lot about the way different attacks have been initiated, but we cannot pass interesting sets of data from one tool to another to get a different perspective on the attack. While much work has concentrated on novel visualisation techniques, we explore ways to bring different tools together to work seamlessly with one another. This research explores the need for a framework upon which different visualisation tools can sit and communicate with one another to aid analysts in the intrusion detection process.

In this paper we present our ideas and our proposition for the framework.

Index Terms—Data mining, framework, intrusion detection, site security monitoring, visualisation

I. INTRODUCTION

As Yurcik et al [3] state, “Commonly understood perquisite infrastructure to facilitate data mining includes mass storage, processing power, data mining software, and a database management system”. Building on this, we base the visualisations upon the work by Avourdiadis et al [2]. Our designs make extensive use of the work by Avourdiadis. The work is based around the “use of a method that will allow information from a number of heterogeneous distributed sources to be logged into a single database”. By having the means of storing and cataloguing multiple types of data into a single storage, an analyst can work with data from any intrusion detection system or “IDS” (so long as the interface is setup) giving a greater level of detail in the visualisation tools.

They present SoapSy, a “lightweight secure access mechanism” which allows data from several sources to be logged effectively into one database. At the heart of the database design are two different parts, namely the CORE and the EXTENSIBLE.

The CORE part is static, the EXTENSIBLE is dynamic. The core stores information that is always available from the IDS sensors, information such as the sensor (that logged the event), source and destination and attack signature. The extensible stores many of the objects that describe the various IDS sensors.

The visualisation framework builds upon the core of the database. The extensible is outside the scope of this project.

Fig. 1. “Tiers” The three main tiers in the architecture include the database, the middleware, and the visualisations.

We regard there being three major tiers in the system. As listed in Fig. 1. these are the Visualisation Tier, Middleware and the Database. As we are making use of Avourdiadis’ [2] designs, we are using a relational database. As there is a separation between the middleware and the database (the SoapSy Interface) we could see future implementations of other database types/designs. All that would need to change is this interface.

An overview of the proposed system is as follows. Visualisation tools sit by themselves in the visualisation tier. They do not directly interact with each other in this tier; rather communicate indirectly through an XML interface. The XML interface allows a degree of separation so that changes to the middleware should not directly affect the tools themselves. It is the middleware tier where the major interactions of the system are carried out. The middleware receives requests for information from tools, obtains the information from the database via a SoapSy interface, and passes the retrieved data back to the tool that made the request. This SoapSy interface allows a level of abstraction between the middleware and the database so that the middleware is not wholly dependant upon any one database design. If changes are made to the core of the database design for example, the SoapSy interface is changed to reflect this and the middleware carries on as normal. Part of the middleware we have called the “Function Engine”. The importance of this can be seen a
little later; for now it is the area in the framework that contains the user-editable requests that a tool can perform on the database.

In section II we discuss related work in the field, how others are contributing to the visualisation of intrusion detection data. In section III we introduce the framework and how the interactions that take place allow the passing of data from one tool to another. In section IV we draw our conclusions and in section V we discuss future work, where to go from here.

II. RELATED WORK

There have been other recent developments into the foray of visualisation of IDS data [4][5]. Komlodi et al [6] has many recognisable and imaginative patterns [12] which would but one must remember this is an independent tool which is (TNV) certainly exhibits the ability to data-mine intrusions, using this as a framework for others to create tools that are

However, although these fundamentals are sound, they are using this as a framework for others to create tools that are independent of each other as their more current work [7] shows. Their “Time-based Network traffic Visualisation” (TNV) certainly exhibits the ability to data-mine intrusions, but one must remember this is an independent tool which is not affected by what other tools see; one cannot adjust – by means of another tools – the information being passed to TNV beforehand.

Our work, on the other hand, takes a different approach to Komlodi et al, by providing an environment where tools can seamlessly communicate with each other by passing information back and forth between them to give an analyst several insightful visualisations. However, it could be beneficial for some collaborative work allowing the insertion of such existing tools into our framework in the future.

There are tools that have been created and used with a degree of success [8][9][10]. Of such note is the comically named “Spinning Cube of Potential Doom” (henceforth CUBE). The CUBE visualises intrusion data captured with the Bro Intrusion Detection System [11]. Depending on the form of attack / intrusion, the CUBE does create some recognisable and imaginative patterns [12] which would certainly help an analyst in spotting certain types of attack clearly without fear of misinterpretation. However, the CUBE is limited to working with the Bro IDS. Lakkaraaju et al [9] again have the similar ideas as Komlodi [7] above with their NVisionIP tool [9][10]; exhibiting “traffic in multiple integrated views”.

The data mining process within this tool are limited to this tool only. The views of the data are limited to what this one tool provides.

III. VISUALISATION FRAMEWORK

The middleware performs all the legwork in the visualisation system. It allows different visualisation tools to interact amongst themselves, retrieve information from the database, and lets new tools be hot plugged into the system without needing to restart the whole system.

Whenever a tool makes a request for data, the middleware retrieves it from the database, returning XML formatted data the tool can read. When an analyst wishes to look at a group of data in more detail they would click a context sensitive area of the visualisation and select another tool to display a different view. It is the middleware that takes care of the actual passing of information. Whenever a context-sensitive area is clicked, a sub-menu pops up with the other possible tools. Behind the scenes, a real-time scan takes place to locate other relevant available tools.

![Fig. 2. Dataflow Diagram](image-url)

Fig. 2. is a brief look at the process behind the traversal from one visualisation tool to another. The general process of moving from one tool to the next involves an analyst clicking on a context sensitive area (“object”). These are objects on the visualisation representing either individual or collections of intrusion data. We could click on an object representing an individual IP address which may then be passed to a tool that gives us the output of a WHOIS lookup – thus providing us with the location and contact details of the domain owner of this IP.

Despite the obviousness in the name middleware, this middle tier in the overall system plays an integral central role in the data mining process. One of the core ideas we need to discuss is how the middleware works between everything and still manages a high degree of loose coupling.

By this we mean that components that work with and hook into the middleware are not affected by changes in the other components that hook in i.e. the tools are not broken (in a programming sense) when the database changes. By maintaining these interfaces (the XML interface between the tools and middleware, and the SoapSy interface between...
the middleware and the database) when changes take place, we end up with a robust system that is open enough to allow new tools, as well as databases to be plugged in and work alongside the existing components.

The Imperative Paradigm

“Logical programming languages … are often thought of as defining “what” is to be computed, rather than “how” the computation is to take place, as an imperative programming language does.” [13]

We initially looked at how we could pass information from one tool to another and decided that using XML as a container for the data was the best solution. XML is open and durable enough such that any tool should be able to interpret the data stored within.

The question that really became apparent was how do we package the information within the XML document? This caused considerable debate. One of our initial ideas involved embedding as much data as we could about the events in the XML.

As the XML document is passed from one tool to another, each tool would “whittle” down the file keeping only the data it needed. What this would ultimately mean is that for each tool to understand what data is currently present in its version of the document there would need to be a way of a tool identifying what it could use.

This leads to the conclusion of having a “type” system where each item of data is regarded as a “data type”. The XML would be constructed around this idea and containers of different types would then be created. A tool would check its own configuration to see what types of data it could accept, match this against what is available in the document, and pull out the relevant data.

By creating the typed system in a methodical way, we could construct a type hierarchy where the different types exhibit inheritance from their parent type, similar to the traits exhibited by popular programming languages such as C++ and Java. As an example, a general IP address object was defined as xxx.xxx.xxx.xxx and source / destination IP addresses were defined beneath it.

A tool could then accept a source IP address in two circumstances; the tool accepted a source IP exclusively or it accepted the more general IP address type (through inheritance). We would allow the modification of “types” by developers so that new types could be integrated and defined. Tools would continue to operate as normal with only the types they knew of, but we would see the automatic integration of similar, inherited types. See Fig. 3.

Fig. 3. Example of object hierarchy, demonstrating type inheritance

A general “Object” type is broken down into the constitutional data types in the database. One could envisage a tool accepting a “Country” type as input and thereby, by inheritance, also accepting “County”. A developer in the future may add a “State” underneath “Country”. The tool could then also accept “State” by default as input.

However, there were two problems with this solution:

- An enormous amount of data would be moving around the system
- We would be unable to alter the “data mining path”

The amount of data being passed from one tool to another initially would be enormous. We would essentially be passing around an XML version of the entire database, not only giving rise to bandwidth considerations, but also negating the database itself.

This said however, after progressing down the data mining path we would see substantially less data traveling between visualisations.

To better describe this, take a typical scenario. We are visualising on a global map where events are coming from. After seeing increased activity from a given country we select this country and select a regional view tool to map the data onto. The new visualisation appears showing the events on a regional scale. In this simple one-step scenario we can already see that the total number of events we are passing around the system has drastically decreased.

This leads us onto the second failing of the initial idea. We are unable to alter the current data mining path. Taking another scenario, we have a graph of events listing where the attacks are coming from; we then select one source IP address to visualise further. The attackers IP address is passed onto another tool which then visualises all destinations on the network that this IP has attacked. We

1 “Data mining path” – The route through which an analyst traverses the data by using different tools for different visualisations
see that one destination (i.e. computer on our network) has been hit several times.

The problem we have now is as follows. If the analyst wants to step back to broaden the search to all attacks on this one destination he cannot as the event data has since been discarded. This is a fundamental drawback of the Imperative paradigm. We would have a very linear approach to data mining; we couldn’t expand on certain areas of interest during the data mining path. Data persistence simply does not exist.

**The Functional Paradigm**

"Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions." [14]

By taking a step back and looking at the different programming paradigms, we discovered that taking a functional approach to the problem could help. By passing around, between tools, just the identifiers of the individual intrusions 2, we make it much easier to keep track of which actual events we are looking at (as opposed to just looking at raw data as described in the imperative paradigm).

By comparison, we could conceivably call this a “typeless” system, as only one data type is being passed around. In addition, since we now know which event we are looking at, one can perform queries or functions on said event to obtain the information we wish to look at. This rids us of some of the side effects of the imperative paradigm, particularly where large amounts of data were initially being passed from one tool to another.

A further constraint inherent within the Imperative paradigm, the inability to broaden the scope of one’s search, is no longer an issue. The running instance of a tool will have a set of events it used to create its visualisation. By saving these sets at given stages, analysts can easily review previous visualisations in the data mining path to replay ideas, broaden the search scope or merely try another path if the current one runs dry. All is needed is a request (via the middleware/SoapSy interface) to the database for further information. The idea of this data mining path is represented by Fig. 4.

---

2 In Avouridias’ database design, these are referred to as Event ID’s.
By adopting a functional approach to the specification for implementation of the middleware we can adapt a lambda calculus approach to the functional specification of behaviour and the development of a semantic understanding of the problem domain.

Lambda calculus also supports the development and validation of function behaviour concerning the manipulation and transformation of data contained in the database. In fact, it offers us the ability to develop a pure formal logic specification, devoid of monotonic logical operators, for the data mining of large data-sets.

Framework summary

![Dataflow Throughout Framework](image)

**TABLE I**

<table>
<thead>
<tr>
<th>No.</th>
<th>Type of data being passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>XML: Event ID(s) &amp; function(s)</td>
</tr>
<tr>
<td>2.</td>
<td>XML: Event ID(s) &amp; data type</td>
</tr>
<tr>
<td>3.</td>
<td>XML: Requested data from database</td>
</tr>
<tr>
<td>4.</td>
<td>XML: Event ID(s) &amp; function(s)</td>
</tr>
<tr>
<td>5.</td>
<td>XML: Event ID(s) &amp; data type</td>
</tr>
<tr>
<td>6.</td>
<td>XML: Requested data from database</td>
</tr>
<tr>
<td>7.</td>
<td>XML: Event ID(s) &amp; data type</td>
</tr>
<tr>
<td>8.</td>
<td>XML: Requested data from database</td>
</tr>
<tr>
<td>9.</td>
<td>SQL Query</td>
</tr>
<tr>
<td>10.</td>
<td>Result of SQL Query</td>
</tr>
</tbody>
</table>

Fig. 5. demonstrates the various interactions that can occur within the framework. Table I lists the contents of the file used in these exchanges. The framework is an intricate affair; the strengths in the system lie in the degrees of separation between the three layers (visualisation, middleware, database). The tools will not be affected if the underlying database changes; the function engine would instead be the only part of the framework to change to include the new database design.

As we can see in Table I there is much emphasis on the Event ID being passed around the system. Mentioned briefly earlier in the functional paradigm, this relates to how the underlying database design [2] stores intrusions. By passing the Event ID’s of the intrusions we wish visualised from one tool to another we keep bandwidth constraints in the visualisation tier to a minimum. A receiving tool can then use the Event ID’s to obtain any data it wants about the intrusion. All it needs to know is what data to get.

**Objects, their role in data exchange**

A concept mentioned, but not explained thus far has been the idea behind data exchange between tools. This is made possible by tool developers integrating “context sensitive areas” or objects within the visualisations, i.e. clickable regions within the programs view.

![Clicking on an Object](image)

It would be up to a developer to determine the data they wish to be made available to other tools; these objects would typically be on items of data already present in the visualisation. You see an IP address you wish to investigate further, click on it and pass it to a WHOIS tool to see who owns the IP address. You know the country where an attack originated, click on it and pass it to a tool which displays all attacks from that particular country. The framework is adaptable enough to work not only with single items, but with groups of data too. If you want to graph several MAC Addresses against attacks that have occurred against them, all that is needed is a graph tool that accepts and processes the corresponding information.

At an implementation level, the objects would generate XML documents containing the event ID’s that contain the selected data, and a number of functions.

**Functions, their role in data exchange**

A function is a statement allowing the retrieval of a specific type of data from the underlying database using the Event ID of a specific recorded intrusion. These statements are stored in the middleware layer within the function engine – tools themselves are not authorised to modify them directly for security reasons. This is to prevent rogue tools deliberately programmed to ignore certain sequences of attacks. The tools should have read-only access and
administrators should be the only ones who can change the way functions work.

The broad idea is to have tools developed with certain functions in mind. There would be two lists of functions; one to describe what a tool can accept as input (the list stored in a tools configuration file perhaps), and another which relate to the objects in the visualisation.

Upon clicking on an object (Fig. 6.) a real-time scan is performed to “pair-up” which tools can accept the data the clicked object can provide. Upon completion, the object generates the XML file (Table I, No. 1 & 4) which is then passed to the selected corresponding tool during its invocation.

IV. CONCLUSIONS

As intrusions have increased over the years, we have found better ways to monitor their occurrence and catalogue their types. One of the big issues facing us now is that we are literally faced with too much data for anyone to reasonably process. We now typically have databases full of ip addresses, region codes, time stamps and event types.

This is where visualisations have excelled; helping us to identify attack patterns clearly with recognizable traits, colour and shapes [5][8][9]. There are certainly many novel techniques that are all powerful tools in their own right for finding specific attacks. If only we could integrate them all and pass around the data to make use of their individual strengths!

Our previous work [1] on the subject provides a broader introduction to this framework. This paper looks at the research carried out since then and concentrates on describing the framework. We have presented what we believe to be the next step in IDS visualisation, bringing different visual tools together in a logical and coherent manner to assist in data mining and the overall intrusion recognition process.

By having a framework that gives different developers a powerful and innovative API, new and existing tools can be integrated easily. Subsets of intrusions can be passed around from one tool to another to easily show different views of the same data, or to drill-down and view just a handful in more depth.

Such a system would give analysts greater control over how they view intrusions and gives them the flexibility to pursue some attacks in more detail or to step back and get a broader picture of what is happening to their networks.

V. FUTURE WORK

There next stage is to look at implementing the various components of the proposed framework, and to look at creating prototype tools to test the success of the architecture.

We need to specifically identify and analyse the expected interactions carried out within the system, namely (See Fig. 5.) those that take place:

- Between tools
  How do they communicate? How do they invoke one another?
- From visualisation tier to middleware tier & vice versa
  The request for data, should we tie ourselves into a specific database language (e.g. SQL)?
- From middleware tier to database tier & vice versa
  The function engine needs to issue requests in the native format of the underlying database, and process the results so that they can be passed on up to the visualisation tools.

Although not interactions themselves, there are other aspects that need mentioning here as their absence would cause the interactions in the system not to work. These are:

- Objects
  The context sensitive areas in a visualisation
- Tool configuration
  How an individual tool is configured
- Functions
  A database query stored in XML and invoked by a tool to obtain information, also assists in pairing up compatible tools to pass data around the system
- Type definition
  Determines which functions a tool should carry out
REFERENCES


Last Accessed: November 2005

Last Accessed: November 2005