A Work-Flow for Empirical Exploration of Security Events

Martin Pirker
Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks,
St.Pölten University of Applied Sciences, Austria
martin.pirker@fhstp.ac.at

Andreas Nusser
Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks,
St.Pölten University of Applied Sciences, Austria
andreas.nusser@fhstp.ac.at

ABSTRACT

As the internet continuously expands, information security research is a never ending challenge. It is impossible to know all internet participants, protocols and their applications. Instead, security research focuses on empirically collected real-world data; stores, processes, transforms and analyses the data, in order to learn from it and its anomalies—security issues—as they happen. This paper presents one practical work-flow for collection and processing of security related data. It present a hard- and software setup, experiences made, and estimates future developments. This gives others the opportunity to learn and identify areas for improvement, especially those in the early stages of setting up a research project based on empirically gathered data.

Keywords

security data mining; big data; virtualisation containers;

1. INTRODUCTION

Computer technology advances continuously. The global internet links all sorts of computing devices and produces new scenarios and applications—and challenges, especially in the domain of information security. While the basic building blocks of computers, operating systems and data exchange protocols are documented, nobody can know and comprehend all of them. While clients, users and their behaviour in a local network are usually known and manageable, once a global audience and their devices connect with global services and interact, locally learned knowledge is no longer sufficient. Instead, one must learn from practical experience and data, and improve and change on-the-fly.

Consequently, this leads to novel and unexpected security problems and scenarios. Due to the complexity of today's technologies and the many ways they may interact, it is impossible to plan or predict for all possible future security challenges. Security research is not solely imagining of developments and events ahead, but also demands a strong foundation in and alignment with practical experiences. This demands an approach of empirical collection of security data and the setup of a data storage and analysis work-flow for researchers active in computer and internet security.

Contribution This paper presents the experience of setting up a work-flow for processing of security events. The specific security research work is not discussed in detail, rather, the focus is on the data collection and processing components in the work-flow. The presentation and documentation of this setup provides the opportunity for others to learn and identify areas for improvement, or just as inspiration on what is possible. This is especially interesting for when one is in the early stages of setting up a new research project in information security.

Outline Section 2 introduces a simple scenario and derives a set of work-flow requirements from it. Section 3 then presents the work-flow in detail. This comprises the hardware and software components and how they interact with each other. Finally, Section 4 reflects on the current state, and possible future developments and challenges.

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2888607 .

2. SCENARIO

2.1 A Search For The (Yet) Unknown

The domain of information security is too complex so that all possibilities and interactions are always considered. Rather, too many security incidents are explained retroactively as "who would have though of *that* to be/turn into a problem". Concurrently, a whole branch of security research accepts as base insight that the challenge is no longer the development of more efficient detectors of known problems. Instead, novel, unknown attacks do and will happen, and consequently the challenge is to detect whether "something" still works as expected. There is "normal behaviour" and there are "anomalies", which may be evidence of actual security problems.

Several previous efforts considered this in the domain of intrusion detection systems. For surveys and overviews of previous approaches of anomaly detection in intrusion detection see, e.g., [3, 5, 9, 16]. These works share the common insight that fixed/known patterns for detection of intrusions only go so-far, the global internet produces attacks faster and faster. As an alternative, in order to detect intrusions, a selection of security related events are observed and recorded. With a larger, collaborative scale, this produces a grander view on the data, and consequently "anomalous" intrusion security events may then be spotted in the data.

We follow the thinking in these works that it is impossible to know what will happen, but when it happens the collected

data will be somehow different. Obviously, by definition, when something different happens it must also produce some difference in the collected data. Naturally, what to look for in detail depends on the specific security research question.

What is clear from this simple scenario of searching for the yet-unknown is, however, that security research requires a proper setup for security data (or events) collection, processing, and iterative refinement of the research question(s) as insights from practically obtained data becomes available and consequently further exploration.

2.2 Requirements

We identified the following requirements to be accomplished by our processing work-flow:

A Large Dataset: In order to spot changes in data over time a sufficiently large collection of data is the basis to work with. Unfortunately, depending on the research question it may be difficult to impossible to guess the data size beforehand. However, it is clear that when the research method is to spot features in data, the data to examine will exceed the storage capacity of a standard desktop or laptop.

Consequently, a work-flow includes a remote server with a huge storage capacity. As security research sometimes handles confidential data provided by companies only via non-disclosure agreement (NDA), the use of additional computing resources from cloud providers is not an option.

Flexibility in Processing and Modification: Common databases store and process data. Today's SQL-based databases provide a rich set of operations to operate on data. However, each SQL database vendor implements a different set of operations in their SQL dialect. A mapping of all data processing to a specific SQL dialect binds one to one specific database vendor and further, individual researchers have to learn the intricacies of database administration. A survey of SQL knowledge in the research team motivated the need that a researcher can implement ideas and experiments solely with a general purpose language, and the SQL knowledge required is kept minimal.

Replication and Portability: A data processing work-flow should be run-able by anyone, easily. Consequently, this motivates the use of primarily open-source software, as a base operating system as well as for the individual data processing stages. Open-source enables the inspection, customisation and replication of the work-flow by anyone.

Further, instead of installation and customisation of software packages directly on a researcher's host operating system, a more flexible way is to aim for a virtualisation-based solution. With virtualisation everyone on the team can use the identical base components with the centrally stored data.

Visualisation And Interaction: The capacity of a human's mind is limited, one can only hold so many abstracted pieces of information at the same time in our mind. Consequently, vast amounts of data demand a reduction of information. Fortunately, our human visual system is very good, as the common saying "A picture says more than thousand words" reminds us.

The work-flow is thus not complete without a webbrowser-based interface, as webbrowsers are a widely deployed solution to display pictures and provide interaction with the user. This naturally enables an "everywhere, anytime" capability, once a specific data mangling function is web interaction enabled.

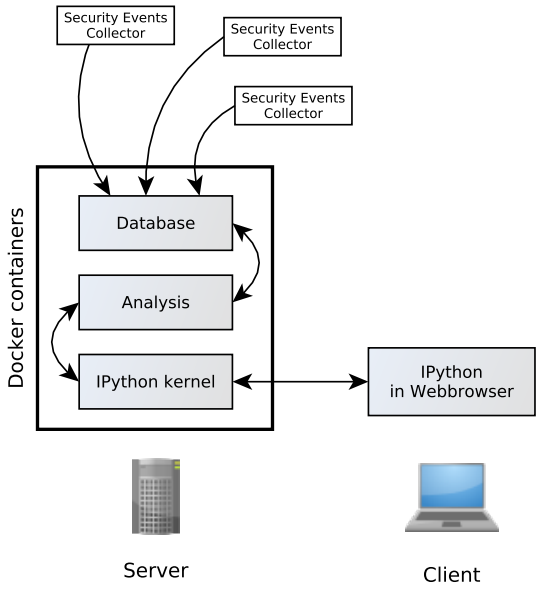

Figure 1: Components and their relations in the work-flow. External event collectors stream their security events to a central server. The server hosts multiple Docker-based containers. From the outside researchers connect via an IPython-based interface running in a standard webbrowser to interact with the data on the server.

3. WORK-FLOW

Building on the scenario reflections in Section 2.1 and the requirements drafted in Section 2.2, we now present an exemplary implementation of our security data collection, storage, processing and interaction work-flow. Figure 1 provides a high-level overview of the major components.

3.1 Hardware

Remote Server: A standard off-the-shelf brand-name server plays the role of the central data processor. It has more (server-class) CPU cores, more gigabytes of main memory, and many terabytes more storage than a standard PC.

Local Workstations: Every researcher has a normal laptop or desktop PC on his desk. Nowadays even standard PCs have quad-core CPUs, gigabytes of memory and fast storage via solid-state drives (SSDs). Consequently, today's PCs often have sufficient computing power to locally develop and test algorithms on a limited data set, before a deployment of them on the full data at the remote server.

3.2 Software

Base/OS Layer: Our server runs with vSphere Hypervisor [17]. It provides the low-level virtualisation layer so the server's resources can be partitioned for individual research problems (or subgroups) as needed.

As base OS running in one virtualisation we chose Ubuntu LTS [2]. The long-term stability (LTS) edition provides ease of administration as the software packages are security maintained and regular updates usually apply without problems.

Ubuntu LTS also comes with support for Docker [4] containers by default. Docker containers provide a separation of resources for different applications. However, all containers run on the same Linux host kernel and are more efficient

than traditional virtualisations that require a full operating system copy. Inside containers it is possible to run a different Linux distribution than the one of the host.

The Ubuntu base remains a bare installation. Instead, several Docker containers host different stages in the data processing. For example, the whole database runs in one container and only exports one network port as interface. The database network port connects to another Docker container that hosts the next data processing layer.

Another container hosts the custom collection of data processing packages (see next sections). This container runs a custom build of Gentoo Linux [6]. The choice of Gentoo is motivated by a) Gentoo is by design easy to customise from scratch and b) the need to be very selective in the installation and combination of software packages (versions).

Docker also enables an easy transfer of containers. A container image may migrate to a desktop PC, or to a laptop for work while travelling. A container image may also be published so that other researcher can recreate the same setup. Naturally, another benefit is that everyone on the team uses the same image, it is just one `docker load` command away.

Data Collection: The data collectors are deployed on different machines and stream their security events via a simple network TCP connection to our server. In this paper the focus is on the data work-flow, the nature of the data collected and the actual research questions are out of scope for this paper. We leave this discussion to another paper.

Storage / Database: The database container receives data via a network connection. The stream of data packets is stored in an SQL database.

Initially, this task was implemented with MySQL [11]. However, we encountered a) data corruption issues and b) performance in some cases depended heavily on the order in which SQL join queries were issued to the database (and obviously not smartly optimised by the database).

Consequently, we abandoned MySQL and instead deployed PostgreSQL [13]. We are unable to report any issues, so far.

Data Mangling: For a general purpose programming language we chose Python [14]. Python provides a very rich environment of ready-to-use libraries. Of special note, Python hosts the "SciPy" ecosystem of open-source software for mathematics, science, and engineering problems. Also, the "PyData" conference series provides this community a forum and several commercial companies push the state of the art of Python for big-data processing.

One unfortunate feature of Python is the split between the end-of-development Python 2.x and the current Python 3.x series. We chose to base our work on Python 3.x right from the start and consider ourselves lucky that all Python packages we use already provide mature Python 3.x support.

For data retrieval we rely on SQLAlchemy [15], it is a full database toolkit to interact with the major SQL databases.

Data processing in pure Python would be too slow. However, the Numpy [18] and Pandas [10] packages provide C-backed acceleration for the processing of large data sets.

The NetworkX [7] package performs in-memory modelling of data relationship graphs. It also provides a set of common graph manipulation functions. Figure 2 depicts an example graph of process call relationships.

Visualisation: For data visualisation Matplotlib [8] is the established Python plotting library and commonly used by scientists. Several packages take advantage of Matplotlib's basic primitives to build more sophisticated (or prettier) visualisations of data (e.g., Seaborn [19]). Matplotlib comes from a time of static graphs (and now also provides interactive ones), however, for large datasets and web-based interaction a new generation of library is needed.

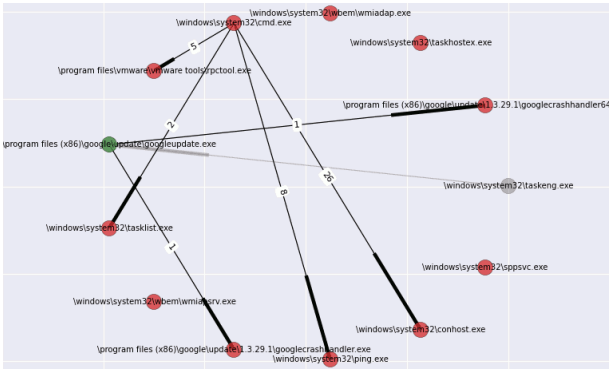

Figure 2: A NetworkX graph with process relationships. Different node colours depict the security status of the process and connecting edges show who called who how many times.

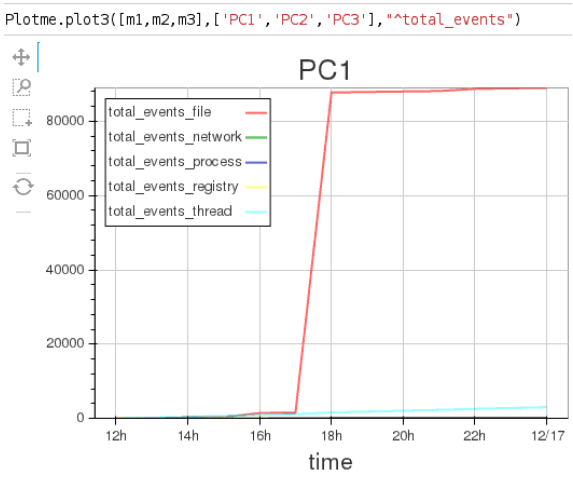

Figure 3: From an IPython notebook dialog a remote call to a Python routine on the server triggers the drawing of a graph with Bokeh. The icons on the left provide interactive zoom, resize and other interactions with the graph in the webbrowser. Obviously, between 17:00 and 18:00 a notable amount of file accesses happened.

Bokeh [1] provides the visualisation for large datasets, an example is shown in Figure 3. While Bokeh is a rather young package and in active development—which unfortunately sometimes requires code changes from releases to release—its explicit development goal is "high-performance interactivity over a large or streaming dataset". So while this young library is still a bit unstable, in the future it will improve and fit well with growing datasets (and requirements).

User Interface and Interactive Exploration: The early stages of data import are mundane work. There is no perfect data stream and some corrections and transformations have to be made, e.g., data timestamp synchronisation errors due to intermediate network errors. This kind

of code is written once and then performs its task in the data processing pipeline, it does not need any further user interaction.

Testing a new idea with the pool of data is, however, an interactive process. A piece of code runs on the data, produces a result, the result leads to a tweaking of parameters and the test runs again. Consequently, this method of data exploration requires an efficient "implement, run, tweak, run again" feedback loop.

IPython [12] provides an interactive interface: an IPython notebook runs in the webbrowser on a local PC. It keeps an open network connection to the remote server where the IPython kernel runs in a Docker container. Code is developed in the web browser, which is then transferred to the server, executed, and the result reported in the webbrowser again. Immediately the code can be edited and re-executed. A result may also be an interactive graph, where the researcher uses the mouse to zoom and scroll in the data.

4. CONCLUSION AND OUTLOOK

The work-flow presented in this short paper serves us well, so far. We are happy that it turned out this way as the overall knowledge of the individual pieces was sparse initially. All team members contributed their knowledge of individual pieces of the puzzle and subsequent assembly of our setup lead to the work-flow as presented. Naturally, it is impossible to plan the complete setup and then implement it. Rather, as more data became available certain parts had to be tweaked for, e.g., a better throughput performance.

With this work-flow, we are able to manipulate data in standard Python 3.4 scripts sufficiently fast, up to approximately 80000 events/s with simple manipulations, and still several thousands event/s with complex analyses. If performance becomes an issue, there are several efforts and packages in the SciPy community that provide alternative, faster algorithm implementations.

We are dependant on the development path of certain libraries, however, we are optimistic that they will continue to work fine. In the worst case our choice of an open-source basis allows to implement changes all by ourselves. Further, we can freely share our setup for reuse and replication of our work. The public availability of datasets depends on how we obtained them; datasets provided by companies under non-disclosure agreement cannot be reexamined by other researchers, obviously.

As our understanding of the recorded security events improves, the strategies for analysis will also adapt. This approach of empirical mining in datasets was enabled by sufficient advances and availability of cheap computing power and storage. We hope novel data analyses with the support of recent software packages also lead to new insights with real-world gathered security event(s) data.

Acknowledgements

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

The work presented in this paper was done at the Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks (TARGET). TARGET is operated by the St. Pölten University of Applied Sciences.

5. REFERENCES

[1] Bokeh Development Team. Bokeh: Python library for interactive visualization. `http://www.bokeh.pydata.org`, 2016.

[2] Canonical Ltd. Ubuntu: The leading OS for pc, tablet, phone and cloud. `http://www.ubuntu.com/`.

[3] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[4] Docker, Inc. Docker: Build, ship, and run any app, anywhere. `https://www.docker.com/`.

[5] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Comput. Secur.*, 28(1-2):18–28, Feb. 2009.

[6] Gentoo Foundation, Inc. Welcome – Gentoo Linux. `https://www.gentoo.org/`.

[7] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug. 2008.

[8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[9] V. Jyothsna and V. V. R. Prasad. Article: A review of anomaly based intrusion detection systems. *International Journal of Computer Applications*, 28(7):26–35, August 2011.

[10] W. McKinney. Data structures for statistical computing in Python. In S. van der Walt and J. Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[11] Oracle Corporation. MySQL: The world's most popular open source database. `https://www.mysql.com/`, 2016.

[12] F. Pérez and B. E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[13] PostgreSQL Global Development Group. PostgreSQL. `http://www.postgresql.org/`, 2016.

[14] Python Software Foundation. Welcome to Python. `https://www.python.org/`.

[15] SQLAlchemy authors and contributors. SQLAlchemy: The data toolkit for python. `http://www.sqlalchemy.org/`, 2016.

[16] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer. Taxonomy and survey of collaborative intrusion detection. *ACM Comput. Surv.*, 47(4):55:1–55:33, May 2015.

[17] VMware, Inc. vSphere Hypervisor. `https://www.vmware.com/`, 2016.

[18] S. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30, Mar 2011.

[19] M. Waskom. Seaborn: statistical data visualization using matplotlib. `https://stanford.edu/~mwaskom/software/seaborn/`.