Open Science, Open Security

Scott Campbell
National Energy Research Scientific Computing Center
Lawrence Berkeley National Lab
Berkeley, CA, USA
scampbell@lbl.gov

***Abstract*— We propose that to address the growing problems with complexity and data volumes in HPC security wee need to refactor how we look at data by creating tools that not only select data, but analyze and represent it in a manner well suited for intuitive analysis. We propose a set of rules describing what this means, and provide a number of production quality tools that represent our current best effort in implementing these ideas.**

Keywords—Intrusion Detection, Security, High Performance Computing

I. Introduction

Balancing the needs of researchers with HPC site security and stability makes practical and effective computer security in an Open Science environment very interesting. Virtual organizations like KBase [9] and the Materials Project [1], web 2.0 based interfaces for job submission and reporting, and more data centric design patterns continue to change the landscape. Given the increasing porous nature of the environment, these changes are forcing security analysis to be both faster and more flexible when looking at security.

For many years, the HPC security community has worked diligently to gather large amounts of diverse data in order to better understand not only who is logging in (or attacking) the system(s), but also what the attackers are doing. These efforts include network monitoring, systems logging, login analysis, process accounting and batch job analysis. In [2] we began to describe a more systematic way for data *acquisition*, focusing principally on structural ideas and design patterns to identify locations where high value data will be found.

The process of gathering all this data has turned us into experts in generating vast quantities of data. In terms of security issues as well as general systems problem solving this has turned out to be tremendously beneficial. Not surprising is that by itself the data does not provide much insight into understanding attack minutia, and we have come to the painful conclusion that getting the data is the easy part. What you do with the data is much harder.

Similarly, knowing what sorts of questions to ask is also more complex than it ought to be. We have observed that for many organizations simple and well-defined techniques are not implemented because security infrastructure does not provide *security primitives*. We think of security primitives as fundamental operations on a data set like rate of change or variance. These primitives can be used in the application of first principal analysis that amount to defining rules, which describe immutable security characteristics.

Our objective here is to explicitly design and implement tools that scale for both data volume and analytic complexity and provide them to the security community free of charge. We have defined a series of heuristics for this goal, and describe efforts toward tool creation.

II. Solution Method

A. Overview

Our method can be broken down into three parts, each building on the one before it. Each will be briefly introduced, then explained in greater detail.

1. **DATA REDUCTION and REPRESENTATION**: Gather and normalize data without bias. Actively filter and reduce to machine friendly form.
2. **DATA ANALYSIS**: Rethink analysis in terms of *first principles,* and *security primitives*.
3. **DATA ACCESS**: Analysis of data needs to be simple. If a tool can't be part of a simple workflow, the user will be incentivized to *not* use it.

After working with these rules and implementing a toolset using these principles, we have found that some basic design components have had to be rethought, but the results have been quite refreshing.

B. Data Reduction and Representation

Given the volume of data generated by an average network and system, it can be quite tempting to pre-filter what gets passed up to analysis based on what you *suspect* represents the security threat. Taking our lead from the original design of the Bro intrusion detection system [13], we differentiate between data and security related decision making. This discourages filtering based on assumptions.

On first look, this runs contrary to data gathering at scale – it is not practical to observe, analyze and store every byte, request or user action. Instead of filtering based on presumed threat, filtering can be done based on the ability of the data to inform decision making, thus avoiding the introduction of excessive bias before the analysis phase. An example of this would be flow shunting, which is the removal of high volume information flows after analysis identifies no threat. [7]. By filtering based on the ability of the data to inform decision making, you avoid the introduction of excessive bias before the analysis phase.

This work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC02-05CH11231.

Reducing data volume by abstracting can help tremendously. The highest resolution data source for network traffic would be a full packet network trace, but problems with data volume and indexing/interaction make this data source difficult to work at volume. In the absence of outrageous hardware resources, we need to balance the typical use case for network traces – short term high information resolution vs. long term connection abstraction. The key here is that with thoughtful identification of how data will be stored and used, data volume can be reduced without excessively impacting analysis.

Once data is identified, *how* it is represented (both pre and post analysis) needs to be considered so the various components can share data and processed results. Many tools historically used for security analysis have an output format that does not lend itself well to either machine reading or being placed in an analysis toolchain.[1] Output is designed for human consumption rather than machine sharing creates a natural stopping point for automation. The canonical example of this would be a report that is emailed out without the results cleanly recorded. Keeping data available in machine parsable format allows you to both generate reports a swell as pass the results along ad-hoc through further analysis.

C. Data Analysis

Analysis here describes interacting with the data to solve problems. Ideally a tool encourages flexibility and exploration from the onset, avoiding any number of issues with a fragile and complex analysis chain. We describe the Bro IDS application design, and why it allows maximum flexibility. We then introduce the two concepts of first principle analysis and security primitives.

For logging and analysis, we have chosen to use the Bro IDS. Its design can be broken into two components. The first is the event generator which accepts data from either a standard pcap interface or serialized event objects from an external source. This event generator takes data characteristics and processes them into a series of agnostic "events". Second is a policy component where the event stream is interpreted and logged by a domain-specific scripting language. This language is used to expresses local site security policy. A particularly useful element is how the scripting language handles events. When an event fires, multiple handlers can be assigned to it, allowing parallel analysis chains to be trivially created. These parallel events are used to build the design shown in Fig. 1 which is motivated by the need to have systematic logging, local security policy and asynchronous interaction.

The 'Log' level is designed with the sole purpose of recording a log of the observed event, whether network connection, user keystroke, or some other small unit of information. The log provides an unbiased transcript of what has transpired and is used for forensic analysis or later research and experimentation.

[1] An example of this would be the way that both snort and bro represented log results in native format.

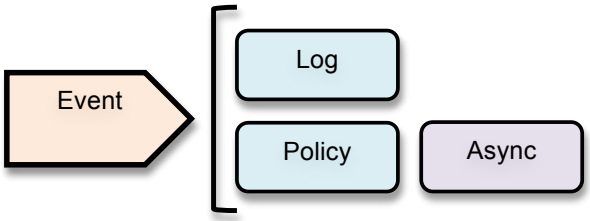

Fig. 1. Representation of event being handled by the logging, local policy and async feed codes.

The 'Local Policy' is where the analysis takes place, and is where we will be looking at introducing security primitives and first principle analysis. The relation between these two are that the first principle is the check on an {object,value} pair and the security primitive is the generalized means to check that relationship. For example you might want to know when your site creates an order of magnitude more outbound network connections. The assertion about site behavior is the first principle, and the mechanism to test will be built from a set of primitives.

An example of a security primitive would be rate of change values for one or more objects. If some value (x) increases by an order of magnitude, or the rate if change (d/dx) or (d^2/dx^2) grows, having a notification for this built into the table behavior would be a tremendously powerful thing. We have prototyped this, but it is not part of our current production.

An example of first principle analysis rests on the idea that except for a small number of well defined mechanisms like an suid executables, a user process should not take on the identity of a privileged account.

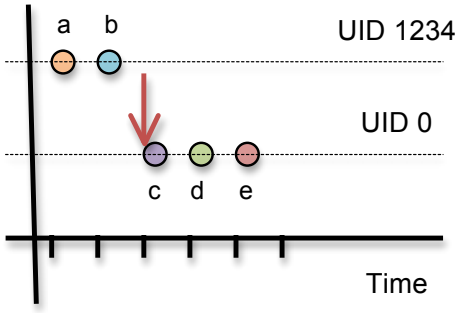

Fig. 2. User executing a set of commands, one (b) which moves their location in permission space.

In the case of Fig. 2, a regular user executes a series of commands represented as dots {a,b,...,e}. The second command transitions from uid 1234 to uid 0, which is quite interesting from a security perspective. The most common way for this to occur is via a suid/sgid binary. On a well-administered system, the set of known and *expected* suid programs can be generated. This set represents permission gateways that an average user would be expected to use in their day-to-day work behavior.

D. Data Access

Data access is a reminder that for something to be useful, it must be worth the time for an analyst to change their working

habits. Since we are building tools to be used, this is a critical point. The security landscape it littered with the remains of visualization and analytics tools which might technically do a better job, but do not fulfill a sort of activation energy that is required for them to be put into use.

III. Related Work

The related work can be broken into several groups. Most HPC related security work is focused on static hardening from an architecture and network point of view. Another set is derived from a compliance and modeling viewpoint. In total there was tremendous focus on highly technical component wise problem solving.

The most relevant work was Yurcik (et al) [17], which describes cluster security as an emergent property derived from the aggregate interaction of the various components. This is driven both by the diversity of architecture components, as well as the *interaction* of these components with one another. The notion that cluster security is an emergent property, irreducible in principle to the sum of the smaller components is totally consistent with the ideas that we are forwarding. Both first principle analysis and the idea of security primitives can be applied to this idea without modification. Some ideas proposed as best practices such as high quality monitoring and moving compute nodes off into RFC 1918 (or otherwise non-routed) address space have become standard design as clusters grow in size.

Two architecture related papers - McMahon and Hutchison "An Architecture for HPC Facilities" [10] and Nowak (et al) "Security in HPC Centers" [12] were similar enough in design advice that they get grouped together. Both embody the notions of static defenses – firewalls, local encryption etc. Since these are design oriented rather than analysis documents, there applicability is mostly in understanding design concepts.

The last paper discussed here is Pourzadi (et.al.) [14] which focuses on a design for highly secure carrier class clustered systems. The system would be broken out into management and security service components. System and network (inter)connectivity permissions would be fully controlled via the central security server as well as correlations with intrusion detection systems. With such an aggressively default deny model for a system, security analysis should be much simpler once the model for application characteristics has been sufficiently vetted. For a single application with limited input this might be possible, but for a site running over a hundred basic codes being driven by custom code and arbitrary data this seems impossible.

IV. Examples

Since we are building publicly accessible production grade tools, the examples presented here are publicly available off of the author's github repository [8]. NERSC has been running iSSHD in production for a number of years, while the Auditd Bro-Framework and user-abstractions are still in alpha format. Building the example programs has informed our opinions about what works and what needs to change.

A. Instrumented SSHD

One of the first attempts at gaining some insight into user activity involved the instrumentation and analysis of local ssh server instances. A complete description of the architecture and analysis used is outside the scope of this paper, but was presented at LISA [3]. In general terms, instrumented sshd allows for the recording and analysis of a significant portion of user input/output, authentication data, and sshd metadata such as port forwarding, channel creation, tunneling and remote command execution.

From a design standpoint we have learned a tremendous amount from the successes and failures of this project. This has carried over into how we design and implement more current projects. As a tool to identify attackers and determining their corresponding success, it has proven to be invaluable. To a large degree it provides well-structured data for identity and various metadata operations (like port forwarding). This follows the desire for highly normalized output data. On the other hand the majority of content runs across tty and non-tty channels where most analysis is still done via regular expressions. For now user activity is not reducible to a normalized form – it is at best semi-structured text which does not lend itself well to machine consumption.

B. Auditd Framework

The AuditD framework is a collection of tools that collect raw auditd kernel data. This section provides a short description of the current implementation highlights. While the isshd implementation was designed during our first attempt at identifying on system user behavior, the auditd analyzer is the first project based on the design principles mentioned in section II. For brevity, we will focus on how the application fulfills the design specs described here more than project internals. A number of well written references exist describing the design, use and configuration of the Linux Audit Framework (auditd) including [6] and [15]

The linux audit framework provides the ability to audit system behavior based on a series of rules that are passed to it via the auditd service. This service allows linux kernel auditing to log data directly into userspace. While the ecosystem of things that live in user-space is fairly complex, we provide a configuration for the auditd service as well as the rule set defining user and system activities which should be logged. This auditd configuration is fairly standard, and is available in the source repo.

The set of auditable objects is quite formidable including system calls, file system objects, authentication services, and a variety of security related services. A complete listing of available object classes is defined in the audit.h header file. For our needs we selected a subset of system calls relating to privilege escalation, networking, and modifications to sensitive files. Given the inherent overhead related to monitoring system calls, we opted for a less complete set to avoid performance issues. Initial testing placed the overhead for an average users activity at approximately 0.5% .

```
6855:5:1 SYSCALL_OBJ SYSCALL 1391192813.866 host-g 214150 32434 execve SYS_EXEC uname /bin/uname 1888710
                    1888d50 188ffd0 sc sc sc sc sc sc sc sc 26589 26588 pts1 yes 0
6855:5:2 EXECVE_OBJ EXECVE 1391192813.866 host-g 214150 26589 2 %20uname%20-m
6855:5:3 PLACE_OBJ CWD 1391192813.866 host-g 214150 26589 /home/scottc NULL -1 -1 -1 -1
6855:5:4 PLACE_OBJ PATH 1391192813.866 host-g 214150 26589 NULL /bin/uname 10356738 0100755 root root
6855:5:5 PLACE_OBJ PATH 1391192813.866 host-g 214150 26589 NULL %28null%29 11665433 0100755 root root

1395458894.644 214150 host-g 32434,sc,sc,sc,sc,sc,sc,sc,sc
   6855:5:1 host-g 26589 214150 EXECVE SYS_EXEC execve uname /bin/uname NULL NULL NULL NULL (null)
   /home/sc 1888710 1888d50 188ffd0 uname -m 26588 pts1 NULL yes 0 root root
```

Fig. 3. Example of first two passes of normalizing for auditd data. The first is output from the host normalizer, and the second is the single line output which bro generates as it's permanent record and the data structure to operate on. The red part is metadata and green is the identity.

Normalizing the data and converting it from its native human oriented format proved to be challenging. When something auditable happens, the overall act is called an *event*. These events are composed of a series of *records*, each as a collection of key:value pairs called *fields*. Events can have many (1-10) records, each of which has from 10-30 fields. To address the tremendous diversity of data, groups based on event types were created. These groups are: users, places, who, socket, execve and internal. Each type is composed of a single data structure built to contain the entire set of information that we were interested in analyzing. All fields are converted into well structured key:value pairs, with all string type values URI encoded [4] for both ease of use and security.

An example of this processed data for the command 'uname –a' can be found in Fig. 3. For all auditd events, the first entry defines the basic action (i.e. what happened), and the remaining lines fill in additional information about the action.

Ultimately we decoupled the identity/ownership with an event from the action and reduced the ad-hoc data structure to a well defined key:value set collection. While action data tends to be transient, identity is useful to track throughout the lifetime of the session (and possibly longer). What we now have is a set of well defined data types containing normalized data that is both clean and associated with an identity.

The local site policy provides the ability to do detailed analysis. Code has been provided which tracks execution, network socket activity (creation of listeners and connections), and the canonical example of tracking a user as they traverses through identity space. Here the various notions of identity: auid, uid, gid, euid, egid, fsuid, fsgid, suid, and sgid are used. The auid is assigned on user login and remains immutable.

We now have the ability to define sets of activity and quickly and clearly test for them in near real time across a large production cluster. Things we can look for include unexpected identity transitions, file permission errors, execution history including flagging unexpected executable locations, and mapping individual network connections to a given user on a given system. We are implementing the second part of the security primitives with prototyping in place for select tables.

V. Conclusion and Future Work

We propose that to address the growing problems with complexity and data volumes a number of basic changes need to take place. Amongst these is the need to refactor how we look at data by creating tools that not only select data, but analyze and represent it in a manner well suited for intuitive analysis. We propose a set of rules describing what this means, and provide a number of examples that represent our current best effort in implementing these ideas.

Moving forward, we are prototyping a number of changes in the auditd analyzer that will build statistical and comparative measures directly into data tables. This would simplify detection of commonly desired qualities like unusualness.

References

[1] A. Jain, S.P. Ong, G. Hautier, W. Chen, W.D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, K.A. Persson. *The Materials Project: A materials genome approach to accelerating materials innovation*
Applied Physics Letters Materials, 2013, 1(1), 011002.

[2] Experiences with Intrusion Detection in High Performance Computing. Scott Campbell, Jim Mellander. Cray User Group, 2012, Anchorage AL

[3] Campbell, Scott. "Local system security via SSHD instrumentation." *Proceedings of the 25th international conference on Large Installation System Administration*. USENIX Association, 2011.

[4] Nick Galbreath, stringencoders: A collection of high performance c-string transformations, http://code.google.com/p/stringencoders/

[5] J. Gonzalez, V. Paxson, and N. Weaver, Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention, Proc. ACM CCS, October 2007.

[6] Steve Grubb, Linux Auditd Main Page, http://people.redhat.com/sgrubb/audit/index.html

[7] J. Gonzalez, V. Paxson, and N. Weaver, Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention, Proc. ACM CCS, October 2007.

[8] https://github.com/set-element Repo for software referenced in work.

[9] KBase, DOE Systems Biology Knowledgebase. https://kbase.us/

[10] McMahon, Peter, and Andrew Hutchison. "A security architecture for high performance computing facilities." (2006).

[11] Nominé, Jean-Philippe, and François Robin. "Security in HPC Centres."

[12] V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time. Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998

[13] M. Pourzandi, I. Haddad, C. Levert, M Zakrewski, and M. Dagenais, "A New Architecture for Secure Carrier-Class Clusters," IEEE International Workshop on Cluster Computing, 2002.

[14] SUSE Linux Enterprise Server Security Guide, Part V, The Linux Audit Framework.

[15] http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/book.security.html

[16] Yurcik, W., Koenig, A., Meng, X. and Greenseid, J. "Cluster Security as a Unique Problem with Emergent Properties: Issues and Techniques". *5th LCI International Conference on Linux Clusters*, Presentation, May 2004