SKALD: A Scalable Architecture for Feature Extraction, Multi-User Analysis, and Real-Time Information Sharing

George D. Webster[§], Zachary D. Hanif, Andre L. P. Ludwig, Tamas K. Lengyel, Apostolis Zarras[§], and Claudia Eckert[§]

[§] Technical University of Munich

Abstract. The inability of existing architectures to allow corporations to quickly process information at scale and share knowledge with peers makes it difficult for malware analysis researchers to present a clear picture of criminal activity. Hence, analysis is limited in effectively and accurately identify the full scale of adversaries' activities and develop effective mitigation strategies. In this paper, we present SKALD: a novel architecture which guides the creation of analysis systems to support the research of malicious activities plaguing computer systems. Our design provides the scalability, flexibility, and robustness needed to process current and future volumes of data. We show that our prototype is able to process millions of samples in only few milliseconds per sample with zero critical errors. Additionally, SKALD enables the development of new methodologies for information sharing, enabling analysis across collective knowledge. Consequently, defenders can perform accurate investigations and real-time discovery, while reducing mitigation time and infrastructure cost.

1 Introduction

Cyber crime has evolved to the point where teams of criminals command sophisticated tools and infrastructures, while possessing the resources required to stay ahead of the defender [4, 18]. For instance, in 2012, McAfee received over 100 thousand samples per day [3], yet on one day in 2015, VirusTotal received over a million unique samples [25]. Disproportionately, malware analysis systems are struggling to scale to meet this challenge and present a clear picture of criminal activity [22, 23, 26]. A solution to this problem is twofold: (*i*) how can corporations extract features and retain a central repository at scale and (*ii*) how can industry peers collaborate in a timely manner without exposing sensitive data and retain essential context.

Simply put, the rise in malware, has strained the ability of the security teams to run analytics and maintain a central repository of generated information. The reason why is because current analytic tools fall short when being used together in an automated fashion and enabling a collaborative environment [22, 23, 26]. As stated by MITRE, this causes a situation where analysts often regenerate

information and duplicate the work of their peers—a huge waste of time and resources [23]. With respect to collaboration with industry peers, the second problem is that once information is received, it is difficult to be shared with security partners in a manner that is timely, retains context, and protects the collection methods. Although, rapid information sharing is an essential element of effective cybersecurity, and will lessen the volume companies need to process, companies are weary of sharing data for fear of tarnishing their business reputation, loosing market share, impairing profits, privacy violations, and revealing internal sources and methods [6]. As a result, it is now common practice to share only with trusted groups large sets of data with minimal context or select post-processed information. Recognizing the criticality of this issue, the US Government recently issued an Executive Order to promoting the sharing of cybersecurity information in private sector [2].

In an attempt to alleviate the burden on analysts, a number of solutions has been developed to help triage data through feature extraction and create a central repository of the collected information [9,23,28]. Unfortunately, many of these tools struggle to scale and provide the fault-tolerance required to support the sheer volume of data needed to be processed in part due to the linear, monolithic, and tightly coupled processing pipeline. For instance, analysis tools, like CRITs [23] and MANTIS [9], are not separated from the core Django/Apache system and are executed on the same physical host, while VIPER [28] has been developed for a single user with the intention of being deployed on a workstation. Consequently, when these system becomes overloaded, a bottleneck occurs that prevents the feature extraction tools from executing properly. To make things even worse, when one of the aforementioned tools fails, it is difficult to perform a graceful exit or cleanup, and the system becomes overwhelmed with a load of only a few thousand malware samples. This results in a situation in which feature extraction cannot be performed quickly and at scale using current technologies and architectures. Furthermore, none of these tools address the issue of how to make assessments on the extracted features.

In this paper we present Skald, a novel architecture to create systems that can perform feature extraction at scale and provide a robust platform for analytic collaboration and data-sharing. In essence, Skald provides the required infrastructure to extract features at a scale that can: (*i*) cope with the growing volume of information, (*ii*) be resilient to system failures, and (*iii*) be flexible enough to incorporate the latest technology trends. In addition, Skald takes a new approach in terms of how data is shared by providing a platform that grants analysts' tools access to the entire extracted feature set without requiring the analysts, and their tools, to have direct access to the raw malware samples or other primary analytic object, such as a domain or an IP address. This enables correlations, clustering, and data discovery over the entire set of collected knowledge, while still protecting the raw objects, the sources, and the methods used to obtain the data. To this end, we develop an open-source prototype and conduct extensive experiments that demonstrate that our architecture has a near linear growth rate and is able to eliminate critical failures when extracting features

across millions of entries. Furthermore, we show major performance gains with the ability to conduct feature extraction at a rate of 3.1 milliseconds per sample, compared to 2.6 seconds when using existing systems. Finally, we discuss how our methodology provides a platform for analysis on a collective set of raw data and extracted features, which enables more accurate clustering of malicious information and real-time data discovery while minimizing the need for redundant feature extraction and thereby reducing analysis time and infrastructure cost.

In summary, we make the following main contributions:

- We develop a framework for feature extraction that is scalable, flexible, and resilient, while it demonstrates near linear growth with zero critical errors over millions of samples.
- We display major speed improvements over traditional techniques using only 3.1 ms per sample when utilizing 100 workers.
- We exhibit the ability of our approach to allow partner organizations to submit new raw analysis objects in real-time leveraging the infrastructure from multiple organizations on different continents.
- We show Skald's capacity to share resultant extracted features and analysis with geographically and organizationally diverse partners who are not sufficiently trusted to have unrestricted access to the raw analysis objects.

2 System Overview

Skald is an architecture for developing analytic platforms for teams working to thwart cyber crime. At its core, Skald dictates the required structure to perform asynchronous feature extraction against submitted objects. It scales these actions horizontally, at a near linear rate, across millions of objects while remaining resilient to failures and providing the necessary flexibility to change analytic methods and core components. Skald additionally provides the necessary infrastructure to perform advanced analytics over the extracted features while empowering analysts to retrieve and share information using their preferred tools and scripting interface. Furthermore, Skald's design allows the sharing of data with partners without requiring the release of raw data. This is because Skald's *Access Control Layer* (ACL) and intelligent core components allow analytics to be executed across a combination of central instance and in-house replicas. Ergo, the information can be easily shared, enabling analysis over a more complete and collective set of data, which overcomes biases caused by informational gaps.

Skald's achievements are primarily due to the approach of logically abstracting the system into "loose coupled" themes and core components, as depicted in Figure 1; allowing the creation of systems that are scalable, flexible, and resilient. This level of abstraction between system elements is critically missing in widely-used systems such as CRITs, VIPER, and MANTIS. This has lead these systems to become monolithic and tightly coupled in design; creating a major hindrance in allowing them to evolve and scale by leveraging distributed computing techniques. However, Skald apart from the scalability it offers, it enables

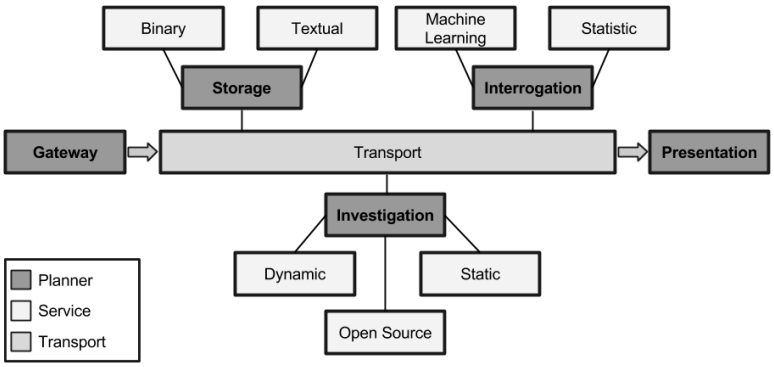

Figure 1. Organization of Skald's components and core themes.

systems to evolve so they can meet the challenges of future cyber criminals by allowing components to be easily exchanged, added, or subtracted. Thus, if a newer, better, or simply different method is discovered, this can be easily incorporated alongside existing methods or simply replace the old ones. This also allows system components to be outsourced to a company, institution, or organization specializing in that work. Finally, this design allows Skald to orchestrate tasking which create efficient system by substantially reducing the infrastructure and network overhead for transmitting data to and from multiple Services.

The structure of Skald is composed of three main components: *Transport*, *Planners*, and *Services*. As Figure 1 illustrates, Transport is the main orchestrator and moves data and tasking to the Planners. Then, Planners allocate infrastructure, enforce security, and oversees the execution of Services. Services in turn perform the requested work and provide the resultant response along with pertinent meta information, such as error messages, back to Planners. This is further described in the following sections.

2.1 Planner

The Planner's primary purpose is to serve as an intelligent orchestrator for Services. At its core, it manages tasking, allocates resources, enforces the ACL, and provides an abstraction between the Services and other aspects of Skald. The Planner also informs the Transport what Services are available for tasking and provides status information back to the system core. As previously mentioned, Planners are loosely coupled with other parts of Skald. In this way, they provide flexibility by ensuring that changes to the core aspects of a Planner will not affect other parts of the system. This helps to improve resiliency by allowing the Transport to delegate tasking to redundant Planners during system failures [5, 19]. Furthermore, this allows the system to horizontally scale by permitting the Transport element to instantiate additional Planners under heavy load while also allowing the Planners to schedule the tasking of Services and allocated additional resources on internal servers and cloud infrastructure.

Communication with Services. The Planner communicates with Services through the HTTP over TLS protocol. We select this method for three main reasons. First, HTTP is widely understood and is capable of transferring a variety of data-types. Second, the wide adoption of the HTTP protocol allows analysts to be able to add new analytics in the language they feel most comfortable. Third, HTTP communication allows Services to be deployed either locally or across network partitions. However, SKALD does not dictate the format of messages transmitted over HTTP over TLS. That said, our prototype provides developers with interfaces for typed JSON message parsing for stronger message safety and a loosely-typed Map data-structure when message safety is not a concern. We choose this because when evaluating other formats in the past, such as Google's Protocol Buffers [8], we found the overall speed increase did not justify the greater level of difficulty.

SKALD provides two methods for the Planner to communicate objects to Services. For local analytics, the object is delivered via a RAM disk, with fail-over to local disks based on size. This creates a fast and easily-available data-store for analytic file reads. For external Services, the Planner will deliver the object via HTTP. When interacting with Services that do not require a local file, the Planner will attach the pertinent meta-data to the tasking message. We select this method for transmitting objects as many existing analytics require a local file to be read. As such, this allows existing tools to maintain relevance through leveraging SKALD's ability to distribute and scale workloads. Furthermore, this improves performance since each Service does not require a costly network transmission to access an object.

Service Orchestration and Management. The Planner leverages configuration management and containers to package Services together in order to manage the tasking of Services and optimize their execution. This provides three core benefits with respect to speed, flexibility, and resiliency [7]. Regarding speed, the packaging of multiple isolated Services on one worker reduces the volume and frequency of data passing through the network. This in turn is a major advantage with distributed and cloud based systems because it eliminates the need for multiple large file transfers. Furthermore, packaging Services increases the flexibility of SKALD-based systems by allowing the rapid deployment of new Services without concern for complex dependency management while ensuring discrete versions of Services and configurations. Finally, containers easily allow Quality of Service (QoS) operations to automatically re-instantiate critically failed Services.

Access Control Layer Enforcement. The Planner is the primary element responsible for managing the ACL by ensuring that tasking is authorized. The Planner also limits potential exposure caused by analyzing an object by enforcing Service execution restrictions through the use of ACL meta-tags. This is done by allowing tasking to state that the execution should only run, for example, on internal hardware, without Internet access, and restrict DNS lookups. This is critically missing in current system designs.

2.2 Planner Themes

SKALD breaks down Planners into five discrete themes as Figure 1 illustrates. This prevents systems from becoming monolithic through the separation of core parts of the system into categories based on their area of influence. Planners are categorized as members of one of the following: Gateway, Investigation, Storage, Interrogation, and Presentation.

Gateway. The Gateway's primary purpose is to receive taskings and push it to the Transport. When tasking is received, the Gateway first performs an ACL check to guarantee that the tasking is authorized. If authorized, Gateway then ensures taskings are valid, scheme compliant, and that the pipeline can handle the requested work. The Gateway can also automatically assign tasking based on an object type. Together this ensures that pipeline resources are not wasted and provides the first level of system security.

Investigation. This theme is responsible for performing feature extraction against objects. When tasking is received, it schedules the execution of its Services which are capable of performing static and dynamic analysis as well as gather data from third parties. During scheduling it optimizes the execution of Services by packaging them together and directly provides them the data. As taskings are executed, it performs the two-fold QoS strategy by monitoring the health of Services and validating received results. Additionally, it enforces the ACL and ensures Services adhere to the meta-tags configured restrictions.

To help illustrate the goals of the Investigation Planner, we describe an ideal execution flow. The Transport layer T2 (see Section 2.4) submits an object for Investigation along with a set of taskings and ACL tags. The Planner first identifies which Services are available for tasking and configures them according to the tasking request and ACL meta-tags. Next, it either packages an object together with a set of Services for execution on one node, or sends the object to a preexisting node dedicated to a Service. The Planner will then monitor the health of the Service and perform any remediation action as needed. For instance, if a Service is unable to gather data from a source due to a query cap, the Planner will reschedule the Service's execution once the cap has expired. When the results of a Service are received, the Planner passes them to the Transport layer which then commands the Storage Planner to archive them. Additionally, if a Service returns new objects, the Planner will submit the object back to the Transport layer with the pertinent ACL tags for storage and tasking to the Gateway.

Storage. This Planner controls how data is stored and retrieved in SKALD. At its core, it is an abstraction layer for the database Services. It enforces a standard storage scheme and passes the requests to the appropriate database elements, identified by UUID4. This enables SKALD to be storage system agnostic and utilize a single or hybrid data storage scheme for resultant data and objects of analysis. The benefit of this approach is that data can be stored in databases optimized for the data type while also easing the inclusion of legacy archives. For example, objects can use Amazon's Simple Storage Service (S3) while the features can be stored in a system optimized for textural data such

as Cassandra [16]. This approach additionally has a major benefit of allowing industry partners to perform in-house replication of selected sets of data while enforcing restrictions on more restricted datasets. For instance, in our prototype, raw objects are stored in restricted datasets hosted by the originator while the extracted features are replicated across all partners. When restricted data is required, the Planner provides contact information to the requester and once approved the Planner will automatically configure access.

The design also enabled the Storage Planner to perform storage-based optimizations, for example, performing deduplication, compression, and deconflict updates and deletions of temporally sensitive datasets. Finally, the Planners overarching view of the system allows it to automatically perform QoS-based operations such as automatically instantiating additional storage shards and coordinating multi-datacenter replication.

Interrogation. The Interrogation Planner focuses on how to turn the extracted features into intelligence in two distinct forms. Aside these forms, the Planner is responsible for scaling the number of Services to balance system load. In the first form, Interrogation Services process system data for retrieval through mechanisms such as an API, plugin, or website. This deviates from previous systems by separating the feature extraction process from the rendering of data. Thus, displaying information is not bound to the extraction method and can change based on what a user's desire. For example, a Service can display the VirusTotal score along side any Yara rule matches. In the second form, an Interrogation Service orchestrates the execution of advanced analytics, such as Machine Learning. To do this, the Planner distributes the work load across available Services to complete the task at hand. This load balancing allows the integration with mini-batch training techniques to achieve large scale model training and generation and serve as a distribution layer for pre-trained models.

Presentation. This Planner provides a standard mechanism for interacting with stored data and the data the Interrogation layer generates. When requests are received, it first ensures that the request accessing information is authenticated and scheme compliant. As this Planner theme is the most likely to vary based on individual use cases, we opt to keep its definition as minimal as possible. That being said, the Presentation layer can be imagined as a microservice fog surrounding the datastores queried by the Interrogation layer.

2.3 Service

Services perform the work being orchestrated by Planners. In Skald, Services are "loosely coupled" and only interact with their parent Planner. As discussed by Papazoglou et al. [19], this highly decentralized model allows services to be platform independent and scale as needed. Additionally, this improves fault-tolerance as no Service is reliant on another Services. Furthermore, the atomic nature of Services provides a great level of flexibility by allowing them to be exchanged as new technology emerges and requirements change. In essence, a

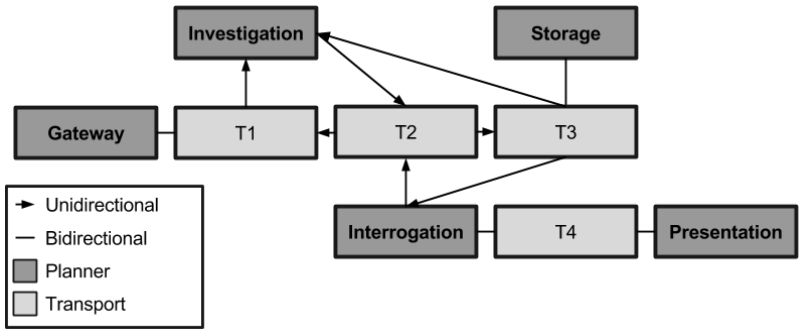

Figure 2. Interaction between the Transports and Planners.

Service is attached to a Planner and performs work associated with that Planner's theme. For example, under the Investigation Planner, Services can gather data from VirusTotal, generate a PEHash [27], and perform dynamic analysis through Cuckoo [10] and Drakuf [17]. An Interrogation Service, on the other hand, will perform an action across a set or subset of the data through data gathering, machine learning, or other data mining and exploration techniques and provide a mechanism to display the resulting information [14].

2.4 Transport

Transport's main task is to move data among the Planners. In addition, the Transport layer monitors the health of the Planners and performs remediation actions to support QoS. This enables a robust level of resilience by ensuring that requested work is always stored in a queue and that results and taskings are never lost. Additionally, the Transport layer improves SKALD's ability to scale by reducing adverse effects of system overloads by allowing the distribution of work across multiple Planners. This is a huge benefit over current available cybersecurity systems that are only design to scale vertically.

The Transport consists of four main sections (T1, T2, T3, and T4) as Figure 2 illustrates. This permits the selection of optimized technology to handle the interaction among Planners. In the following we introduce these sections.

Transport - T1. T1 is focused on moving data to the Investigation Planner for analysis. To do this, T1 is required to perform three primary actions. The first action is to receive tasking from the Gateway Planner and schedule their transmission to the Investigation Planner. The second is to receive tasking from T2 and submit them to the Gateway Planner for validation. The final action is to monitor the health of the Interrogation Planner and perform QoS management. When implementing T1, we recommend the utilization of a distributed message broker such as Kafka or RabbitMQ.

Transport - T2. T2 is focused on receiving objects and results from the Investigation and Interrogation Planners. To do this, T2 has two primary actions. The first action is to receive data from the Investigation and Interrogation Planners and schedule their submission to the T3 for Storage. This is separated from T3 to allow a message queue service to be implemented to help throttle the storage of data during peak loads as storage operations can be costly but are often not time critical. The secondary action is to receive objects from the Investigation and Interrogation Planners and pass them to T1 for further analysis. Like T1, we recommend the use of a distributed message broker.

Transport - T3. T3 is focused on submitting and retrieving data from the Storage Planner. As such, T3 is responsible for three primary actions: (*i*) provide data from the Storage Planner directly to the Investigation and Interrogation Planner, (*ii*) receive information from T2 and pass the data on to the Storage Planner, and (*iii*) manage the QoS of the Storage Planner. We recommend the first two actions to be implemented with no message queues between the Storage Planner and the databases. This permits database Services to rely on their own optimization frameworks during the retrieval of data; databases are often heavily optimized for retrieval of data and implement their own form of message queues.

Transport - T4. T4 handles the exchange of information between the Interrogation and Presentation Planners. The first action is to provide a conduit for communication between the Presentation and Interrogation Planners. The second is to monitor the health of the Investigation Planner and perform QoS management. As the Interrogation Planner will typically provide data through HTTP calls, we recommend implementing T4 as HTTP load balancers.

3 System Wide Aspects

In this section we present system wide aspects of Skald. We first introduce the QoS strategy we follow and then discuss the ACL requirements.

3.1 Quality of Service

Malware authors are incentivized to thwart analysis and as a result the failure of Services should be expected. To counter this, Skald automatically recovers from issues arising from the execution of Services by leveraging a robust QoS pattern. It does this through a two-fold QoS philosophy of monitoring the actual Service and the resultant response from a Service. Thus, Skald accounts for the scenario of when the returned work has failed or even if the actual Service has failed and accounts for them differently.

To implement the QoS strategy, the Planner monitors the health of the Service worker using container status messages and HTTP response codes. The Planner then evaluates the availability, response time, and throughput of each attached Service. If the evaluation responds by stating the Service is operating within normal bounds, the Planner will then send the results to the second stage

to evaluate the returned work. This allows Service authors to specify deep level checks and perform automated remediation actions that are Service specific. If a failure occurs at either step, the Planner will determine if the Service has entered a failed state and perform remediation action, such as restarting the Service container. If the Service appears to be healthy, the Planner will re-queue the work that has demonstrated failure while saving otherwise successful Service results.

The unique aspect of this strategy is that SKALD views the tasking of each Service as single elements and process them individually. When the Service or returned work fails, SKALD will only discard failed work as opposed to abandoning combined tasking and queued work. This is a key difference between SKALD and previously systems. For example, the methodology used by systems which rely on the HDFS and MapReduce model for their data and task distribution, such as BinaryPig [11] and BitShred [13], will cease processing or discard successful results when percentages of work fail. However in SKALD, even if there is a high number of failed jobs, any successful result will be saved and failed work will be reattempted. Furthermore, in the event of pathological failure, SKALD will store the tasking in a separate queue for human intervention. Our evaluation showed this new approach provides significant performance benefits and was outright required when executing large, historical, analytic tasks across cybersecurity data as the data is often designed to confound investigations and cause failures.

SKALD's QoS system also accounts for congestion during times of peak load as well as Service shutdown and instantiation. To do this, the Planner enforces throttling of Services using a pull-based pattern with an "at least once" message delivery scheme [12]. In this scheme, the Planner is aware of the system's current message load and pulls a configurable number of messages, in which it is capable of completing, from the Transport layer. Each requested message is then tracked within the Planner as a discrete entity. Upon completion of work, the Planner notifies the Transport of the work status, pushes the results to the queue, and pulls additional tasking. This allows SKALD to prevent the overburdening of Planners while also ensuring that no work is lost due to component failure. This approach also reduces the chance that a failed Service state will replicate to other parts of the system and cause a work stoppage due to being overburdened.

3.2 Access Control Layer

The nature of handling unique and sensitive data within SKALD requires the incorporation of a complex ACL system. Unfortunately, the standard ACL model used in cybersecurity only allows for isolation based on the source of the raw data or a user's role. However, the problem is that this does not address access to sensitive capabilities, differentiate Services from users, or separate raw data from features and analytics; creating an "all or nothing" approach to access. Therefore, the systems cannot be designed to easily allow analysts or Services to derive intelligence across a collective set of information without granting the analysts access to all sources and sensitive capabilities.

To overcome this, SKALD creates a new ACL model by granting access based on *User*, *Capability*, *Source*, and *Meta-tags*. The *User* defines the users and com-

ponents of the system. While *Capabilities* map to Services and their derived information, *Source* maps the origin of the raw data. Finally, *Meta-tags* provide Planners with Service execution restrictions. For example, these tags can specify that dynamic analysis can only execute without Internet access. While implemented in our prototype, the full definition of the ACL is left to future work.

4 Evaluation

To evaluate SKALD, we created an open-source prototype and performed a series of experiments. We acknowledge that the SKALD focuses on the structure of a system and does not prescribe implementation methods. While this can create varying performance metrics, we feel it is prudent to present a baseline implementation to demonstrate the significant improvements afforded by the SKALD architecture. Throughout this section, we use our prototype to evaluate the architecture's (*i*) scalability, (*ii*) resiliency, and (*iii*) flexibility.

4.1 Experimental Environment

We leveraged three hardware profiles for our evaluation. The first profile was used as a control and deployed CRITs using their recommended setup with the following nodes: (*i*) *Ingest VM*: 2 cores 4GB RAM, (*ii*) *MongoDB VM*: 10 cores 32GB RAM, and (*iii*) *CRITs VM*: 6 cores 32GB RAM. CRITs was selected for our control as it is an industry standard for performing multi-user analytics. Additionally, we made the assumption that CRITs performs similarly to other systems such as MANTIS [9], as the architectures are remarkably similar and Django-based. The second profile deploys our prototype using a similar hardware profile as the control. In this profile we deployed the prototype in a cloud-based environment using the following nodes: (*i*) *Workers*: three AWS EC2 M3 large instances, (*ii*) *Transport*: One AWS M3 xlarge instance, and (*iii*) *Storage*: One AWS M3 medium instance. Finally, the third profile was used to evaluate the horizontal scalability of the SKALD architecture. In this profile we used the following nodes: (*i*) *Workers*: 100 AWS EC2 M3 large instances, (*ii*) *Transport*: One AWS M3 xlarge instance, and (*iii*) *Storage*: One AWS M3 medium instance.

In all experiments we used a diverse set of malicious PE32 samples from Virus Share, Maltrieve, Shadowserver, and private donations. These binaries encompasses traditional criminal malware, highly advanced state-sponsored malware, and programs which are not confirmed as malicious but are suspicious.

4.2 Scalability

In order to provide a meaningful evaluation of SKALD's scalability, we studied the ability to ingest PE32 samples and then execute a series of three Investigation Services on our profiles. We selected PE32s as this provides a direct mapping

Framework	1K	5K	10K	50K
CRITs	2.8000	3.1774	3.3781	1.1929
3 Workers	0.0502	0.0558	0.0616	0.1303
100 Workers	0.0032	0.0032	0.0032	0.0025

Table 1. Average time to process samples in seconds.

to CRITs and our prototype performs at a near identical level when processing Domain, PCAP, IP, and other objects. To this end, we did not perform any experiments related to the storage of Service results because these experiments would vary depending on the data-store used. Furthermore, we omitted experiments on the Interrogation Services as this architectural model is relatively similar to Investigation and it is difficult to perform clear correlations among existing systems.

During our experiments, we used four sets of data: 1000, 5000, 10,000 and 50,000 randomly selected samples. As SKALD is intended to scale to support large datasets, we additionally ran SKALD with a set of one million samples. In both cases we pushed the samples into each system through a linear sequence of RESTful calls with no delays. We did this to evaluate the ability for the systems to queue work and simulate batch queues that we regularly encounter in our work. Additionally, to ensure that the evaluation was as fair as possible, we levered existing CRITs Services and added a RESTful wrapper around Services to make them compatible with our prototype. These Services gather PEInfo data, check the file against the VirusTotal private API, and run each sample against 12,431 Yara [1] signatures provided by Yara Exchange [20].

Our first scalability experiment revealed that SKALD outperforms existing systems, as shown in Table 1. Our prototype, which was deployed with 100 workers, was able to process each sample at an approximate rate of 3.1 ms. Additionally, this hardware setup began to show speed improvements, due to the scheduler caching, at about 50,000 samples.

An interesting finding is that even with similar hardware, the design still greatly outperformed existing systems and was able to process each sample at an average rate of 74.5 ms. In terms of speed, this is a significant improvement over CRITs' average rate of 2.64 seconds per sample. Furthermore, these results highlight one of the critical issues plaguing existing systems. The perceived rate increase with CRITs at 50,000 samples was caused by an overload of the system. When this occurred, the operating system began to randomly kill processes before completion, producing a false appearance of speed improvements.

Having established SKALD's performance against existing systems, we then studied the prototypes ability to scale to meet the demand of very large sample sets, as Figure 3 depicts. Unfortunately, a direct comparison with CRITs was not possible because CRITs could not maintain stability beyond 50,000 samples. Nevertheless, this experiment clearly showed that our prototype was able to scale to meet the demand of processing a million samples and performed at

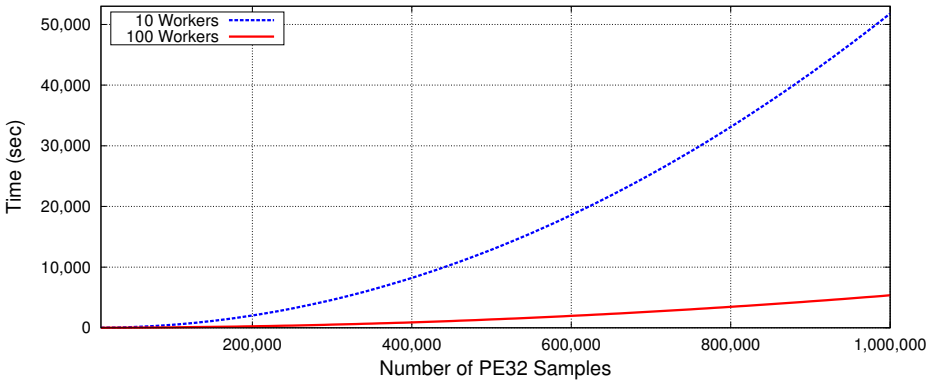

Figure 3. SKALD's speed in processing one million samples.

an approximate rate of 7.5 ms per sample using 100 workers. When analyzing the results, we identified that the slowdown, with respect to the previous experiment, was caused by the Transport requiring disk reads due to tasking size. This however, was a constant rate and we are confident that SKALD performs at a similar rate irrespective of the volume of samples.

4.3 Resilience

Next, we executed two experiments to study the resiliency of SKALD. For the first experiment, we wrote 26KB worth of random bytes across 20% of the samples. This was done to generate files that represent work that will potentially confuse and fail analysis tasks. This allowed us to evaluate SKALD's ability to cope with failed, long-running, and troublesome work. For the second experiment, we studied how SKALD-based systems perform while core components of the infrastructure are unavailable or entered a failed state. We did this by running a script that randomly killed and restarted the machines, in isolation and in unison, hosting the Planner, Transport, and Service components of the system. During both experiments, we reran newly generated file sets through each hardware profile using the same method as used in the scalability experiments.

It was immediately apparent that SKALD's design and QoS paradigm greatly outperformed current systems. In the first experiment, as Table 2 shows, our prototype encountered zero critical errors, defined as a Service failing to complete tasking. While the Services did fail, the Planners successfully recovered from all errors encountered by Services and continued processing other unaffected Services. This is in direct contrast with CRITs reporting 17,012 critical errors using a set of 50,000 samples. In an investigation of the results, the high critical error rate was because the system was unable to handle the load caused by failed Services and, as a result, these Services entered a locked state, consuming valuable resources, or were killed by the operating system before results could be submitted for storage.

Framework	1K	5K	10K	50K
CRITs	0	0	151	17,012
3 Workers	0	0	0	0
100 Workers	0	0	0	0

Table 2. Critical failures in sample processing.

During the second experiment, SKALD remained tolerant of faults. While the overall processing speed was decreased, our prototype's QoS paradigm was able to identify failed states and re-queue the tasking without any operator interaction. The final outcomes revealed that no work was lost and 100% of the tasking were completed with no critical errors.

These results showed the benefits of SKALD. During these experiments, our prototype was not only resilient to handling poorly performing Services, but also easily handled failures to critical components of the system. Furthermore, these results revealed the benefits of SKALD's QoS structure in that work was never lost and the system never entered a failed state. By contrast, CRITs' failed Services remained in a failed state and human intervention would be required to clean faulty results from the database, reset service, and re-task the system.

4.4 Flexibility

In order to examine SKALD's flexibility, we first evaluated its ability to incorporate existing feature extraction methods. To do so, we created an Interrogation Service by modifying the existing CRITs PEInfo Service. This required the modification of less than 50 lines of code in order to remove CRITs specific commands and provide a RESTful HTTP wrapper. During evaluation, we noted that the wrapped Service performed with no discernible difference when compared to natively written SKALD Services. The prototype was able to seamlessly incorporate the Service to include performing QoS operations. In summary, this demonstrated SKALD's ability to provide flexibility by showing a minimal level or required work to incorporate a none native Service.

Next, we wanted to study SKALD's flexibility in changing a core component. We did this by testing the ability of our prototype to directly work with the CRITs database. This experiment was selected because the original implementation of SKALD's Storage Planner used Cassandra and Amazon S3 as the database back-end. In contrast, the CRITs framework uses MongoDB Documents and GridFS. Thus, we were not only required to change the underlying scheme for storing data but also the core database technologies. To do this, we made three modifications to the original prototype: (*i*) we created a Storage Service that parsed the results into the CRITs database scheme, (*ii*) we introduced an additional Storage Service that queried the existing CRITs database system to retrieve raw objects, and (*iii*) we included logic in the Storage Planner to identify which Storage Services to select when handling tasking. In total, we

were able to make these changes using less than 100 lines of code. Furthermore, no modifications to the other parts of Skald were required.

In summary, we are confident that Skald provides the flexibility required to relatively easily change Services as well as core components of a system. We are confident that these results are applicable to other parts of the system.

5 Use Cases

In this section we present use cases to demonstrate how Skald can overcome many of the current limitations in sharing data among industry partners. These use cases are based on our experience in using our framework to manage a collective set of millions of objects, including their associated analysis, across three globally distributed organizations.

5.1 Sharing Resources with Geographically Distributed Partners

Security partners rarely share infrastructure for fear that this will lead to informational exposure. This fear is well founded as it has in the past tarnished business reputations, reduced market share, impaired profits, caused privacy violations, and revealed internal sources and methods [6]. This poses a problem where processing resources become duplicated and large capital is required for their maintenance. While we acknowledge that Skald cannot overcome the reasons behind why data is restricted, our framework can overcome the problem of how to develop a shared infrastructure.

This is because Skald allows each partner to leverage their own Storage Planner to restrict the transfer of raw and restricted data and configure the Transport to acknowledge internal and external Planners. This in turn allows partners to leverage a collective set of hardware resources while allowing the creation of Investigation Services capable of performing data discovery, clustering, and colorations over the entire set of collective knowledge.

5.2 Sharing Derived Information with Partners

Cybersecurity analysis is mostly retrospective in nature and based on identifying previously observed attack patterns [6]. Sharing information is vital in this process as it allows analysts to make better correlations and see a more accurate picture of what is happening. Unfortunately, current methods for sharing information among partners require too much time to be truly effective. This is because information first needs to be identified as important, processed, validated as sharable, and then transmitted to partners often in the form of *Indicators of Compromise*. This is an inherently slow process but an even more critical issue is that feature extraction methods are not uniform and vary among industry partners [21]. This causes the receiving party to rerun extraction methods before the information can be leveraged by their analysts.

Skald overcomes the above described issue by providing a platform that supports breaking the traditional paradigm of how information is shared. This is accomplished by providing the infrastructure to allow partners to directly share a central repository of extracted features and jointly perform analysis against the information. This approach allows partners: (*i*) to have a common set of extraction methods that is understood by all parties, (*ii*) to have real-time access to current information, and (*iii*) to have a wider view of the malicious activity during the investigation.

Skald makes the aforementioned setup an easy task. In our deployment, we created this setup with globally distributed partners by adding a Storage Service that contained the logic required to replicate features between industry peers while restricting sensitive raw data. Thus, this created a uniformed platform that allowed all partners to leverage a collective pool of information. Furthermore, when working with Investigation tasking, the taskings can leverage the ACL meta-tags, discussed in Section 2.1, to inform the system if it should only use in-house resources or use collective resources.

6 Related Work

The need for a large scale malware analysis framework has been well discussed in prior work. Bitshred is a prime example and can perform malware correlation based on hashes of features, thus greatly increasing the throughput of the analysis system [13]. However, it only addresses the problem of dealing with already extracted data through clustering correlations on these hashed features. It does not address the full analysis lifecycle and is reliant on other systems for feature extraction. Hanif et al. [11] attempted to solve the problem of performing scalable feature extraction in their work on BinaryPig, by performing feature extraction in a distributed fashion through the use of a Hadoop based back-end. Unfortunately, BinaryPig only performs static analysis of malware binaries, it does not support deriving intelligence over the features, and the QoS schedule cannot appropriately handle failed work.

Significant research has been put forward in distributed setups addressing the issues of scaling, flexibility, and resilience. Service Oriented Architectures (SOA) breaks systems down into business processes, i.e., Services, and integrates them in an asynchronous event-driven manner through an Enterprise Service Bus (ESB) [15]. By establishing disjoint components, SOA enables distributed systems by defining how Services communicate versus their implementation. The xSOA architecture expands upon traditional SOA by allowing multiple Services to be combined under a single composite Service [19]. This composite server then provides a management layer which can perform Service orchestration, routing, provisioning, as well as integrity checking. Verma et al. [24] has taken xSOA a step further in their work on large-scale cluster management with Borg. They developed a xSOA like system and optimized it by introducing a BorgMaster. The BorgMaster serves as the master for Services and schedules their execution across Borglets. Together, the BorgMaster and Borglets intelligently manage

the execution of Services and perform necessary actions to improve resilience, flexibility, and scalability. Additionally, Borg packages Services together for execution to improve efficiency by cutting down on transmission time, network bandwidth, and resource utilization. However, these architectures are not meant to deal with cybersecurity work as the data is often designed to cause component failures and Skald's additional abstraction is required.

7 Conclusions

In this paper, we designed Skald: an architecture to support the entire lifecycle of analysis against the ever growing volume of malicious activities plaguing computer systems. Skald enables the design of a large-scale, distributed system that is applicable in the security domain. To this end, it supports multiple users and scales horizontally to support analysis of millions of objects. Skald provides mechanisms for implementors to incorporate static and dynamic feature extraction techniques and apply advanced analytic across these features to derive intelligence. Furthermore, it breaks the paradigm that the automated extraction is the ultimate goal, opting to provide a flexible architecture to empower human analysts. In a similar manner, Skald's design overcomes the limitation of current sharing models. It does this by providing the infrastructure needed to allow industry peers to perform analysis across collective knowledge while protecting sensitive data. Empirical results confirm that our solution scales horizontally, at near linear growth, and is able to process objects at a rate of 3.1 milliseconds with zero critical error in contrast to existing system's rate of 2.6 seconds and thousands of critical errors. Additionally, Skald allows analysis across a collaborative set of raw data and extracted features. Thus, it enables more accurate clustering of malicious information and real-time data discovery while minimizes the need for redundant feature extraction systems and thereby reduces analysis time and infrastructure cost.

Acknowledgments

We would like to thank the Technical University of Munich for providing ample infrastructure to support our prototype development. We would also like to thank the United States Air Force for sponsoring George Webster in his academic pursuit. In addition, we thank the German Federal Ministry of Education and Research for providing funding for hardware under grant 16KIS0328 (IUNO). Lastly, we would like to thank the members of VirusTotal, Yara Exchange, and DARPA for their valuable discussions and support.

Availability

The prototype used to evaluate the Skald framework is open-source under the Apache2 license. It can be accessed at: `http://holmesprocessing.github.io/`

References

1. V. M. Alvarez. Yara 3.3.0. VirusTotal (Google, Inc). `http://plusvic.github.io/yara/`, 2015.
2. O. Barack. Executive Order No. 13691. Promoting Private Sector Cybersecurity Information Sharing, 2015.
3. Z. Bu, T. Dirro, P. Greve, Y. Lin, D. Marcus, F. Paget, V. Pogulievsky, C. Schmugar, J. Shah, D. Sommer, et al. McAfee Threats Report: Second Quarter 2012. 2012.
4. K.-K. R. Choo. The Cyber Threat Landscape: Challenges and Future Research Directions. *Computers & Security*, 30(8):719–731, 2011.
5. F. Cristian. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*, 34(2):56–78, 1991.
6. DARPA. Cyber Information Sharing - DARPA Cyber Forum, Oct 2015.
7. J. Estublier. Software Configuration Management: A Roadmap. In *Conference on the Future of Software Engineering*, 2000.
8. Google. Protocol Buffers. `https://developers.google.com/protocol-buffers/`, Nov 2015.
9. B. Grobauer, S. Berger, J. Göbel, T. Schreck, and J. Wallinger. The MANTIS Framework: Cyber Threat Intelligence Management for CERTs. *Boston, US, Jun*, 2014.
10. C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser. The Cuckoo Sandbox. `http://cuckoosandbox.org`, 2012.
11. Z. Hanif, T. Calhoun, and J. Trost. Binarypig: Scalable Static Binary Analysis Over Hado Op. *Black Hat USA*, 2013.
12. HiveMQ. MQTT Essentials Part 6: Quality of Service 0, 1 & 2. `http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels`, 2015.
13. J. Jang, D. Brumley, and S. Venkataraman. Bitshred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *Conference on Computer and Communications Security (CCS)*, 2011.
14. B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert. Adaptive Semantics-Aware Malware Classification. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.
15. D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall Professional, 2005.
16. A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
17. T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias. Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System. In *Annual Computer Security Applications Conference (ACSAC)*, 2014.
18. G. Ollmann. Behind Todays Crimeware Installation Lifecycle: How Advanced Malware Morphs to Remain Stealthy and Persistent. Technical report, Damballa, 2011.
19. M. P. Papazoglou and W.-J. Van Den Heuvel. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB journal*, 16(3):389–415, 2007.
20. M. Parkour and A. DiMino. Deepend Research - Yara Exchange. `http://www.deependresearch.org/2012/08/yara-signature-exchange-google-group.htm`, May 2015.

21. W. Shields. Problems with Pehash Implementations. https://gist.github.com/wxsBSD/07a5709fdcb59d346e9e, Sep 2014.
22. A. Stamos. The Failure of the Security Industry. http://www.scmagazine.com/the-failure-of-the-security-industry/article/403261/, Apr 2015.
23. The MITRE Corporation. Collaborative Research Into Threats (CRITs). http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/collaborative-research-into-threats-crits, Jun 2014.
24. A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-Scale Cluster Management at Google with Borg. In *European Conference on Computer Systems (EuroSys)*, 2015.
25. VirusTotal. File Statistics. https://www.virustotal.com/en/statistics/, May 2015.
26. P. Vixie. Internet Security Marketing: Buyer Beware. http://www.circleid.com/posts/20150420_internet_security_marketing_buyer_beware/, Apr 2015.
27. G. Wicherski. Pehash: A Novel Approach to Fast Malware Clustering. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.
28. L. Zeltser. SANS - Managing and Exploring Malware Samples with Viper. https://digital-forensics.sans.org/blog/2014/06/04/managing-and-exploring-malware-samples-with-viper, Jun 2014.