Internet-Scale File Analysis

Zach Hanif
Novetta
zhanif@novetta.com

Tamas K Lengyel
Novetta, TUM, UConn
tamas@tklengyel.com

George D Webster
TUM
webstergd@sec.in.tum.de

***Abstract*— Malicious file analysis is well beyond the days when creating simple hashes for binaries was sufficient. The use of malicious PDF, Office, and other files present a far more diverse threat than our defensive tools were originally designed to handle. Even PE32 executables have been turned into poli- and meta-morphic binaries with layers of packing applied to hide from detection. To make matters worse, the sheer influx of files to analyze presents a meaningful logistical problem which becomes increasingly complex as analytic methods move from static to dynamic analysis. When the point in time problem is considered - the fact that historical discoveries can be viewed differently in the light of new analytic techniques or information - the problem seems all but intractable.**

To this end, we designed the *Skald* framework, a blueprint for future analytic systems. We leveraged this framework to develop TOTEM, a system which is capable of coordinating, orchestrating, and scaling malware analytics across multiple cloud providers and thousands of running instances. TOTEM makes it easy to add new capabilities and can intelligently segregate work based on features, such as filetype, analytic duration, and computational complexity. TOTEM supports dynamic analysis through DRAKVUF, a novel open-source dynamic malware analysis system which was designed specifically to achieve unparalleled scalability, while maintaining a high level of stealth and visibility into the executing sample. Building on the latest hardware virtualization extensions found in Intel processors and the Xen hypervisor, DRAKVUF remains completely hidden from the executing sample and requires no special software to be installed within the sandbox. Further addressing the problem of monitoring kernel-mode rootkits as well as user-space applications, DRAKVUF significantly raises the bar for evasive malware to remain undetected.

This paper will discuss the design, implementation, and practical deployment of TOTEM and DRAKVUF to analyze tremendous numbers of binary files.

I. INTRODUCTION

Modern malware analysis systems have come a long way over the past years with numerous powerful tools are currently available to researchers. Unfortunately, many of these tools have been created to aid in the analysis of binaries for the purpose of signature generation. While effective at achieving their original goals, they fall short when stretched to deal with the latest analysis trends, technology breakthroughs, and simply being used together in an automated fashion. Furthermore, cyber criminals are growing in sophistication and writing a signature to detect malware binaries is no longer enough to stop the threat. This problem is two fold. The first problem is that cyber criminals have organized themselves. These teams command sophisticated tools, infrastructure, financial networks, and the resources required to stay ahead of the defender [5], [18]. The second problem is that researchers are faced with an increasing rise in malware volume. For example, in 2012, McAfee reported receiving over 100k samples per day [3], yet on May 7 of 2015 alone, VirusTotal received over 1 million unique samples [20].

To put it simply, a single analyst is no longer enough to protect networks and identify malicious activity. A range of specialized skills and tools are now required to perform investigations and keep systems secure. In response, the lone reverse engineer is evolving into teams of analysis to identify:

1) Who is behind the action
2) What are their goals
3) Where is the infrastructure
4) When do they operate
5) Why are they conducting the operation
6) How do we thwart their activities

To answer these questions teams must work together, leverage the knowledge of their industry peers, and be able to process an ever growing and changing volume of data. Sadly, cyber security tools have not evolved to help these teams perform their mission. The cyber security community is in desperate need of tools that can extract features, utilize techniques to make sense of the data, and provide a platform to share this data with our peers. Furthermore, these tools must scale to cope with the growing volume of information, be resilient to system failures, and be flexible enough to incorporate the latest technology trends.

The goal of this paper is to provide a high level overview of the Skald framework for large scale threat intelligence. The framework enables the creation of systems that can support teams of analysts against the current threats. We will then discuss two systems we created, TOTEM and DRAKVUF, that already do leverage and integrate with Skald-based systems. TOTEM is capable of performing triage against objects through static analysis and collect Indicators of Compromise (IOC) information from partners at a scale previously impossible. We will then discuss our solution for dynamic analysis, DRAKVUF, which performs hypervisor based analysis efficiently while remaining stealthy and increasingly resilient to evidence tampering.

II. BACKGROUND

Many of today's security systems exist in a vacuum, designed to "just get the job done", irrespective of how they integrate into a larger security landscape [21]. While current systems represent major steps forward, significant improvements are required in the flow of data analysis if the cyber security community wishes to tackle our present and future challenges. In our experience, many of today's systems

struggle to scale and provide the fault-tolerance required to support the sheer amount of data that needs to be processed. One of the major reasons for this is the linear, monolithic, and tightly coupled processing pipeline of current systems. For example, in the case of CRITs and MANTIS, secondary tools are not separated from the core Apache system and are executed on the same physical host. When the system becomes overloaded, the scheduler forms a bottleneck and prevents the secondary tools from executing properly. As a result, the entire system can become unresponsive under heavy load. Additionally, we observed that when one of these tools fails, a graceful exit or cleanup is difficult to perform and the system can quickly become overwhelmed with a load of only a few thousand malware samples. HTTP load balancing can help alleviate this problem but represents a temporary fix, and not a true solution. These issues of scale are not only limited to systems that focus on performing malware and file analytics. Current methods used for capturing and sharing IOCs are also significantly limited when scaling across large internal and external user bases. Ultimately, issues of scale and resiliency of current systems limit the scope of analysis and collaboration that can be performed.

Once analysis systems have collected data and IOCs, working with the collected body of knowledge is another area in need of improvement. To be truly effective, we must merge existing systems together to leverage their strengths in collecting data while also providing the infrastructure needed to make sense of the data collected. Currently, multiple systems are required to complete operational and strategic threat intelligence tasks, each system storing a subset of the data that must then be consumed by a human analyst. As shown by Rieck *et al.* [17] and Chau *et al.* [4], the computational merger of this data can prove to be invaluable when performing advanced analytics using machine learning, as well as during human guided analysis. Unfortunately, no such system or framework exists that can tackle this problem. Furthermore, systems designed to make sense of cyber threat intelligence data are not able to easily digest data from the highly different sources.

III. SYSTEM DESIGN

A. Skald

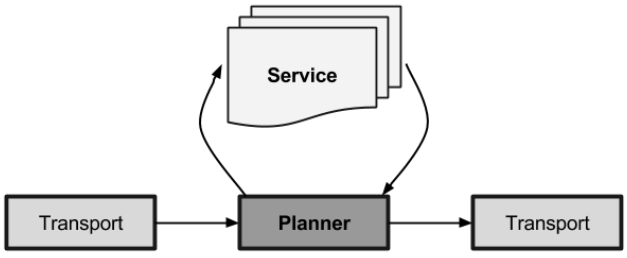

Fig. 1: Organization of the Framework

The Skald framework provides a foundation for developing analytic platforms to support teams of analysts working to thwart cyber-crime. It provides a foundation that is centered around the three core pillars of resilience, scalability, and flexibility. The Skald framework is designed to support many types of object input by making the pipeline file and data-type agnostic. Skald then provides the structure needed to perform user-defined advanced analytics against a set of the data for one to many object types. This means that Skald based systems can asynchronously perform appropriate work (for example on PE32 files, PDF documents, and lists of domain names), with individual objects subjected to differing workflows based on user and datatype requirements. These actions are possible even while presenting an interface to empower the analysts as to allow them to retrieve and share their results using their preferred tools and scripting interface. Skald is then capable of scaling these actions across millions of objects while remaining resilient to failures and providing the flexibility needed to change analytic methods and core features.

The Skald framework is designed around the eXtended Service-oriented Architecture (xSOA) with loosely coupled Services [12], [13] and the microservices design pattern. This was chosen as the foundation as xSOA naturally provides flexibility and loosely coupling Services allows the system to be resilient to faults. Additionally, leveraging the principals of loosely coupled architecture affords a distributed system that can scale to large workloads. Skald has been slightly modified from the traditional xSOA architecture by incorporating elements from the Borg system [19] to ensure that the system would satisfy the three pillars - resilience, scalability, and flexibility - when conducting analysis.

Three main elements, *Services*, *Planners*, and *Transports*, compose the Skald framework. As illustrated in Figure 1 the *Transport* functions as the main orchestrator and moves data and tasking around to the *Planners*. The *Planners* then allocate infrastructure, enforce security, task the *Services*, and monitor the *Services* health. The *Services* perform the requested work and provide the resultant response, along with pertinent meta-information such as error messages, back to the *Planner*.

The Planner plays a major role in enabling the scalability of the system. Like the Borg model, the Planner is able to see all available tasking and the requirements of Services. Because of this, it is able to package Services together when this makes sense. This allows a Skald system to operate efficiently by substantially reducing the infrastructure and network overhead for transmitting data to and from Services. In our evaluation we found that this was a major boon in particular when scaling to a cloud-based infrastructure.

The independence of elements and abstraction provided by the Planner creates a system that can be flexible. Because a Planner only needs to know how to communicate with a Service and what infrastructure it needs to execute the task, Services can be exchanged, added, or subtracted to the system easily. As long as the Services can communicate with the Planner, and vice versa, the Planner requires no knowledge of how the Services accomplish their tasks. Should a newer, better, or simply different method be discovered to extract

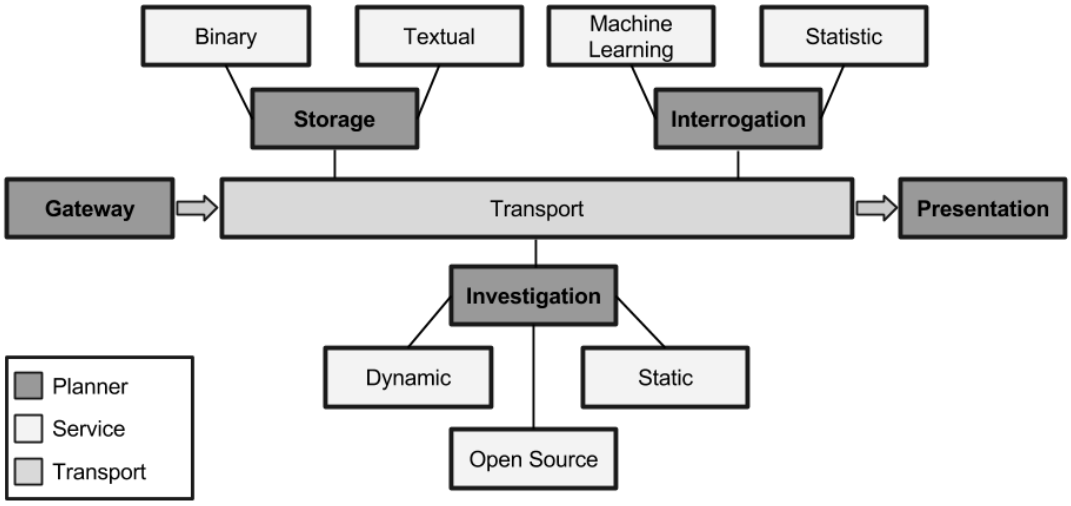

Fig. 2: Organization of the Framework Themes

PE header information, for example, it can be swapped out with or added alongside the old one very easily. Additionally, this model allows 3rd parties to provide the Planners and Services components for a Skald based system. Thus, an implementation based on Skald can outsource Planner or Service creation and development to a company specializing in that work.

Finally, Planners are logically broken into themes as illustrated in Figure 2. This breakout allows Skald to provide a standard API/language that allows analysts to plug in different viewing agents, ingest routines, and custom data query scripts. This allows the analysis of Skald's resultant data to be the final goal and helps tackle the data overload problem discussed by Woods *et al.* [26]. Specifically, Skald's data output format remains flexible and allows presentation of that data to vary based on the needs and will of the analyst. With the Skald design, it is feasible to have a Presenter Service that allows one analysts to view the data through a tool such as Maltego while a RE analysts can view a set of the data through a plugin in IDAPro, while not requiring a change in how the data is gathered, or affecting the data formats of other analytic processes.

1) Investigation: The Investigation Planner is responsible for performing analysis against one object. It does this through providing four primary functions. The first function is to schedule the execution of its analysis Services against a supplied object. These Services then perform static analysis, dynamic analysis, and gathering of data from other sources that are either publicly or privately available. This satisfies our goal for resiliency by having the analysis performed outside of the Planner component. As such, a failed Service will not hinder other Services and the Planner itself. The second function is to perform QoS operations through monitoring the health of Services and reporting status. Additionally, the Planner should execute remediation operations such as adjusting its scheduler and issuing additional Services. Thus, allowing the system to scale to meet operational requirements. The third function is the enforcement of the ACL by adhering to configuring restrictions listed in the ACL-meta tags. Finally, the fourth function is to optimize the tasking of Services such as reducing data transfer over network boundaries.

To help illustrate the goal of the Investigation Planner, in an ideal world the Planner would function as follows. The Transport layer will submit an object for Investigation along with a set of taskings and ACL tags. The Planner will then identify which Services are available for tasking and configure them according to the tasking request and ACL meta tags. Next, the Planner will identify how best to execute the Services through either packaging an object together with a set of Services for execution on one node or through sending the object to an preexisting node dedicated to a Service. The Planner will then monitor the health of the Service and perform any remediation action as needed. For example, if a Service is unable to gather data from a source such as VirusTotal due to a query cap, the Planner will reschedule the Service's execution once the cap has expired. Once the results of a Service is received, the Planner will then pass the results to the Transport layer to have the Storage Planner store the results. Additionally, if a Service returns new objects, the Planner will package a new object with a set of taskings and ACL tags. It will then submit the object to the Transport layer to begin analysis.

2) Interrogation: The Interrogation Planner focuses on what to do with the data and metadata after it extracted from the raw objects of analysis. This takes on two forms. The first focuses on how best to organize data in the system for retrieval. The second form is how to performs analysis across a set of elements in storage. Additionally, the Interrogation Planner should be capable of performing caching to reduce database queries and Service tasking.

In the first form, Interrogation Services will gather the necessary data and then present them in a digestible form. For example, a Service for displaying results from VirusTotal

(gathered by an Investigation Service) will receive the needed stored results and then provide the mechanism to present what is desirable. This separation provides flexibility by allowing the display to change based on user needs. For example, if at a later date IDAPro integration is desired, an Interrogation Service can be created that provides this interaction. Additionally, the separation allows a Skald system to scale with the number of users querying a system for data.

In the second form, an Interrogation Service will perform advanced analytics using techniques such as machine learning. Supporting Services will perform the computational work load while the Planner will distribute the work load across available Services to complete the task at hand. This load balancing can be integrated with mini-batch training techniques to achieve large scale model training and generation, or could serve as a distribution layer for pre-trained models so that querying Services can use such models locally.

B. TOTEM

Implemented in Scala, TOTEM was developed concurrently to the definition of the Skald framework. Scala was selected here for its functional programming capabilities, ability to use already existing JVM libraries, and widespread support. Compared to other major languages such as Python, Scala was selected due to its speed and ease of deployment. One of our goals with Totem was to ensure that it could run across a diverse set of VPS providers, and the ease of deploying JVM programs to Google Compute Cluster, AWS, and others, were major considerations. Because of the fact that we needed TOTEM to be easy to add new Servers, while allowing developers to be able to easily develop work flows across their unique needs, TOTEM has been implemented as a series of libraries, allowing users to modify their installations to their needs. Totem was designed to be approachable to both small users and large ones, a departure from previous mass malware processing systems. As a result, there are relatively few external requirements - both in the hardware and software realm. The TOTEM stack is easily extended, but for discussion's sake, we will refer to the stack used during the early production stages at Novetta.

C. TOTEM Terms

TOTEM is somewhat different from other stream processing systems, so it is important to clearly state the language that will be using to describe the system going forward.

1) Worker: A specific instance of a TOTEM worker system, separate from the Message Queue. Each Worker is a self-contained system of processes, which coordinate with each other on the local system, and are capable of executing a predefined set of analytic tasks. Workers have no knowledge of other members of their swarm, and only communicate with the transport layer.
2) Job: A structure which details the location of a file to analyze, the individual analysis tasks to be performed, any associated information or context needed to perform those tasks, and the number of times this overall job has been attempted.
3) Work: An individual analytic process to be executed. Given the example of having a Job where the file in question is a PDF document, a possible task would be the extraction of textual data from that document.
4) Work Results: The results returned from a single analytic process. A Job which describes three Work units - the hashing of a file, application of Yara rulesets against that file, and the submission of that file to VirusTotal - would generate three Work Results. These results can analytic successes, failures, or errors.
5) Eviction: The removal of a Job from a worker's local store. This occurs when the job has resolved as a success, or failure, with regards to operations performed, and not necessarily the analytic result. TOTEM has a concept of partial completion - Work which needs to be requeued are re-queued, whereas Work which is completed is forwarded on, even within Jobs that result with a combination of these results.

D. TOTEM Components

The TOTEM system relies on relatively few discrete components, which can be directly associated with the component primitives laid out in Skald. In this section we will discuss architectural components of TOTEM, and how they relate to their corresponding Skald counterparts. TOTEM's architecture is optimized towards preventing unneeded network transfer of files. This makes relying on other stream processing systems such as Apache Storm or Spark difficult, as both systems have built in shuffle and enrichment distribution mechanisms which heavily favor distributed services and network shuffle operations. Additionally, it is assumed that users will wish to process large amounts of work at inconsistent times, forcing the need to have components that can dynamically scale the members of the worker and analytic swarm without lengthy data re-balancing operations. Additionally, many tools which are currently used by analysts assume that analytic targets exist on a local disk, a concern which encourages workers to have as many analytic Services co-resident as is feasible. These design constraints heavily factored into our selection of technologies, as detailed below.

1) Transport: The backbone of Skald-inspired services, TOTEM relies on RabbitMQ to transport data into and out of the system. Our initial work was with the RabbitMQ AMQP platform because of its robustness to traditional failure scenarios, its support for complex routing topologies and persistence/retry strategies, and ability to horizontally scale to cope with large workloads well. In selecting our Transport layer message queue, we prioritized reliability and routing expressiveness over raw speed. This is primarily a product of the environment that we originally deployed TOTEM into (AWS), more than a hard requirement or commentary on other queuing platforms.

2) Binary Storage: This component serves as an interface to other backing file systems. In our smaller-scale deployments, we have opted to have an HTTPS layer proxy con-

nection requests to our short-term, or long-term datastores. This has the advantage of making our connectors extremely generic, but depending on the deployment, users may opt to have their installation directly access their binary store of record.

3) Planner: Within the TOTEM system, the user effectively assembles the Planner component during the creation of the Worker.

4) Analytic Services: TOTEM is designed to allow analysts to write new analytic processes in languages that they are familiar with. As a result, we needed a communication protocol that was common across a wide variety of languages, was relatively simple to understand and debug, and could move variable quantities of arbitrary data. We selected HTTP as our method and protocol of communication. HTTP allows arbitrary data to be passed, is very well understood, and operates at reasonable speeds. There is nothing that prevents users from implementing their own communication protocols, but the initial release of TOTEM provides assets for communicating with HTTP services. The services offered by the TOTEM worker are typically shipped concurrently with the worker process itself at deployment time.

E. TOTEM Concurrency

Unlike other systems, such as CRITS and VIPER, TOTEM is designed to achieve throughput through concurrent sample processing. TOTEM's concurrency model is based on Actors, implemented through the Akka project. It is important to note that Actors are only to be used for stateful concurrency, hence our actual calls to analytic services are done via Futures. Our services are connected over HTTP/S REST interfaces, and are language agnostic.

Unlike other distributed systems, each TOTEM instance is effectively stand alone. TOTEM's communication is message-centric, and form the sole means of sharing information between multiple worker nodes. Messages both from exterior source, and interior sources, contain all the context appropriate for that message. This means that a new work element details to each worker what work needs to be done, where the resource of operation is located, and what, if any, optional commands should be performed. These messages are persistent throughout the message queue - without explicit acknowledgment of completion or failure, no messages are removed from the message queue.

Individual workers have no knowledge of whether or not there are any other workers running. This makes each TOTEM worker effectively a stand-alone deployment. As all communication between nodes is performed via messages passed asynchronously, there is no need or support for direct internode communication or conflict resolution. In a departure from Skald, currently we do not use an external service such as Zookeeper to register enrichment services - this is currently being considered as future work. Currently, we take advantage of our transport layer's wildcard routing key matching to selectively route messages to applicable workers.

This decentralization allows nodes to fail individually, without the risk of taking down other elements of the system with them. While TOTEM cannot guarantee zero work loss during catastrophic failure, there is no portion of the system which can directly cause the failure of any other, and work lost during failures is limited to only work locally stored during processing. After such failures, such work tasks are retried due to their presence on the queue system. This results in an at-least-once processing system. This was selected due to the concern of losing work, as opposed to the concern of doing some work tasks more than once. Our planned worst case scenarios in standard operation revolve around the partitioning or failure of the message queuing backbone. Should this happen, work will obviously stop flowing through the system, but will resume when connections have been reestablished. Work in progress will be discarded.

F. TOTEM Work Resolution

To prevent work loss we ensure that messages are fully processed before removing the original Job from the Transport layer. To do so, each worker, within its' local state, ensures that Jobs have had their component Work elements submitted to all applicable analytic services on the system, that responses from those services have been collected, that those responses have been transmitted to the Transport layer and that any failed or incomplete work has been submitted for re-processing. While we cannot make perfect data loss guarantees, this multi-stage confirmation prevents tasking from being lost.

IV. TOTEM Example

This example will demonstrate the core elements of a TOTEM enrichment service. Implementing this function as a solitary worker requires the development of a configuration file, the definition of the work and result class types, the overall system driver, and the addition of this work to the parser work routers. The example driver below will generate two messages, and submit them for processing by the Yara, PEInfo, and FileMetadata services. Defined services will return results to the results queue, and any undefined services will re-queue their work.

A. Driver

Each worker is controlled by a driver, a class which serves as the primary entry point to the program, and which represents the Planner component as defined by Skald. This driver handles the parsing of configuration settings, defines the structure of the component queuing system communications, and serves as a central point of control for internal worker services. The majority of this functionality is defined by the TOTEM libraries and only require instantiation and configuration.

B. WorkEncoding

The following three worker functions along with the enclosing class which inherits the WorkEncoding trait are

needed to parse, route, and manage the various enrichment services that this worker supports.

```
class TotemEncoding(conf: Config) extends WorkEncoding {
  ...
  def GeneratePartial(work: String): String = { ... }
  def enumerateWork(key: Long,
    filename: String,
    workToDo: Map[String, List[String]]
  ): List[TaskedWork] = { ... }
  def workRoutingKey(work: WorkResult): String = { ... }
}
```

C. MetadataREST

This is the TOTEM representation of an enrichment service - the class that represents how work is to be done, how to interact with the service itself, and the internal classes that represent the work and its respective results. Each work element, and their result, can be as simple (a string) or as complex (a fully native POSO), as desired, allowing developers to further enrich analytic results, or begin secondary processing as a part of a longer chain of operations.

```
case class MetadataWork(
  key: Long,
  filename: String,
  TimeoutMillis: Int,
  WorkType: String,
  Worker: String,
  Arguments: List[String]
) extends TaskedWork {
  def doWork(): Future[WorkResult] = { ... }
}
case class MetadataSuccess(
  status: Boolean,
  data: String,
  Arguments: List[String],
  routingKey: String = "metadata.result.static.zoo",
  WorkType: String = "FILE_METADATA"
) extends WorkSuccess
case class MetadataFailure(
  status: Boolean,
  data: String,
  Arguments: List[String],
  routingKey: String = "",
  WorkType: String = "FILE_METADATA"
) extends WorkFailure

object MetadataREST {
  def constructURL(
    root: String,
    filename: String,
    arguments: List[String]
  ): String = { ... }
}
```

D. DRAKVUF

Since the proliferation of poli- and meta-morphic malware, dynamic malware analysis has been an effective approach to understand and categorize malware by observing the execution of malware samples in a quarantined environment [8], [25]. The interaction between the executing malware sample and the host OS allows dynamic malware analysis systems to collect behavioral characteristics that aid in formulating defensive steps.

As dynamic malware analysis systems have become widely deployed, malware has evolved to detect and evade such systems by either refusing to execute in a sandboxed environment, or by modifying its run-time behavior to lead the analysis system astray [2]. Consequently, it is critical for dynamic malware analysis systems to provide a *stealthy* environment to hide the presence of the data collection from the executing sample [7]. However, even recent analysis systems that meet the stealth requirements [6] have considerable resource requirements and require manual interaction with the malware samples, which constitutes a barrier to scalable, automated dynamic malware analysis.

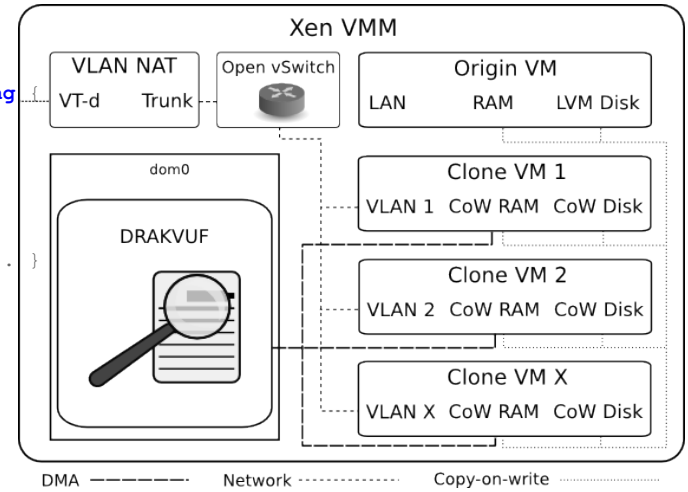

Fig. 3: System overview of DRAKVUF

DRAKVUF is a prototype system designed to address these concerns directly. An overview of DRAKVUF can be seen in Figure 3. In this prototype we implemented a set of data collection mechanisms for 32-bit and 64-bit versions of Windows 7 SP1 using only hardware virtualization extensions as the source of information. DRAKVUF builds directly on the Xen Project Hypervisor, and through the use of Virtual Machine Introspection becomes capable of monitoring the execution of the sandbox purely from an external perspective.

As dynamic malware analysis relies on observing the live execution of malware, the *fidelity* of the collected data is essential. As rootkits employ a variety of techniques to hide their presence on a system, the broader the scope of data collection, the more likely the analysis will reveal useful features. In our current prototype system we focus on Windows but our system could easily be extended to monitor Linux and other operating systems as well, as the underlying monitoring techniques are OS agnostic.

A key feature of existing dynamic malware analysis systems is the ability to trace the execution of processes by monitoring system calls. However, monitoring *only* system calls limits the execution trace to the interaction between user-space programs and the kernel, which does not include the execution of kernel-mode rootkits. To overcome this issue, in *DRAKVUF* we took an alternative approach by directly trapping *internal* kernel functions via #BP injection. With direct trapping *DRAKVUF* is able to monitor malicious drivers as well as rootkits, which was previously not possible

with just system call interception.

The location of the kernel functions are determined by extracting information from the debug data provided for the kernel. The use of debugging information has been an established method in the forensics community and it is the most convenient avenue to gain insight into the state of the operating system. In *DRAKVUF* we make use of the Rekall forensics tool [16] to parse the debug data provided by Microsoft to establish a map of internal kernel functions.

At runtime, *DRAKVUF* locates the kernel automatically in memory without having to perform signature based scans, which improves resiliency as compared to existing forensics tools [11], [24]. To automatically locate the kernel in memory we make the observation that Windows 7 uses the FS and GS registers to store a kernel virtual address pointing to the `_KPCR` structure, which is always loaded into a fixed relative virtual address (RVA) within the kernel, identified by the `KiInitialPCR` symbol. As we have obtained the RVA for all kernel symbols, including `KiInitialPCR`, we only have to subtract the known RVA of the symbol from the address found in the vCPU register to obtain the kernel base address.

Once the kernel base address is established, *DRAKVUF* can trap all kernel functions via #BP injection. With internal kernel functions being trapped, the logs thus generated provide a full trace of the execution of the OS from the moment the malware sample is executed.

In the current implementation *DRAKVUF* tracks all system calls and heap allocations of the kernel and generates logs based on the associated tag of the structure. If the tag of the structure is one of the already known 2,254 tags, the log contains further details about the type of the object to aid the analyst in identifying allocations that may be of further interest. To detect for example a hidden process, an analyst can now apply a cross-view check to determine if the allocated structures are also accessible via standard lists [10]. *DRAKVUF* further traps the routines responsible for freeing these structures. Thus, providing a full-view into the life-cycle of the structures. In the next section we further illustrate how this approach enables us to track the active usage of `_FILE_OBJECT`s.

Monitoring filesystem accesses is one of the core feature of any dynamic malware analysis system. However, prior agentless VMI approaches have attempted to monitor file-system accesses by modifying the disk emulator to intercept events [15]. While such an approach is effective, reconstructing high-level file-system accesses (like path and permissions) from the low-level disk-emulation perspective is in itself a form of the semantic gap problem and requires extensive knowledge of file-system internals. However, the internal kernel structures that *DRAKVUF* tracks reveal highly valuable information about the execution state of the system, such as the complete set of running processes, kernel modules, threads, and even objects allocated for filesystem accesses by the OS.

The process by which we catch filesystem accesses is shown in Figure 4. When a file is accessed, either by

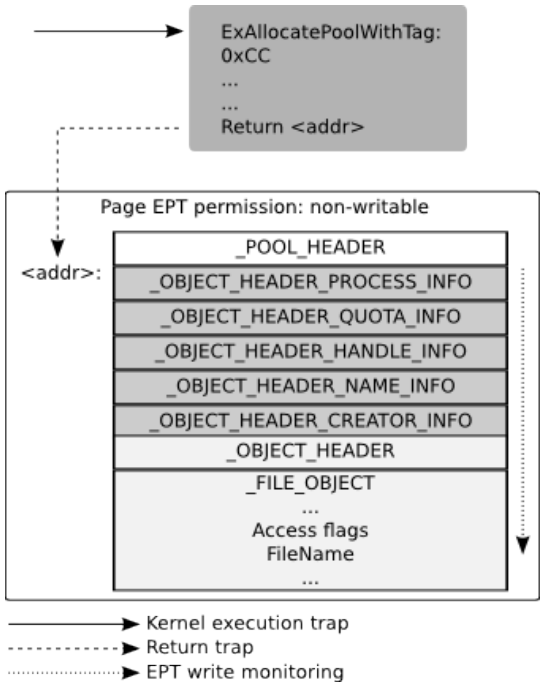

Fig. 4: Tracking file accesses by monitoring the allocation of `_FILE_OBJECT`s in the Windows kernel heap.

the OS or by a user-land process, a `_FILE_OBJECT` is allocated within the kernel heap with the accompanying tag, `"Fil\xe5"`. When the allocation address is caught, we mark the page on which the structure is allocated as non-writable in the EPT. As the `_FILE_OBJECT` is preceded by a set of optional object headers (shown with a gray background), we derive the exact location of the access flags and file name by subtracting the known size of the `_FILE_OBJECT` from the end of the heap allocation. This allows us to determine the full path of the file as well as the access privilege with which the file is accessed, such as read, write and/or delete permission, without the need to have any deeper understanding of the filesystem itself.

In *DRAKVUF*, we also have turned our attention to a critical step so far overlooked in automated dynamic malware analysis: we start the execution of the malware sample without leaving an identifiable trace of the monitoring environment. With systems where an in-guest component is used, the execution can be initiated by the monitoring agent itself, but the same in-guest component could be potentially used to detect monitoring, even if it is only an autostart script. On the other hand, when no in-guest agent is present, the sample has to be started manually. Therefore, in order to avoid creating any artifacts within the analysis VMs but to allow automated execution, we implemented an injection mechanism that hijacks an actively running but arbitrary process within the VM to initiate the start of the sample on our behalf.

The hijack mechanism takes over the execution at the first instruction executed in `CPL3`, as shown in Figure 5, and locates the CreateProcessA routine in kernel32.dll's export

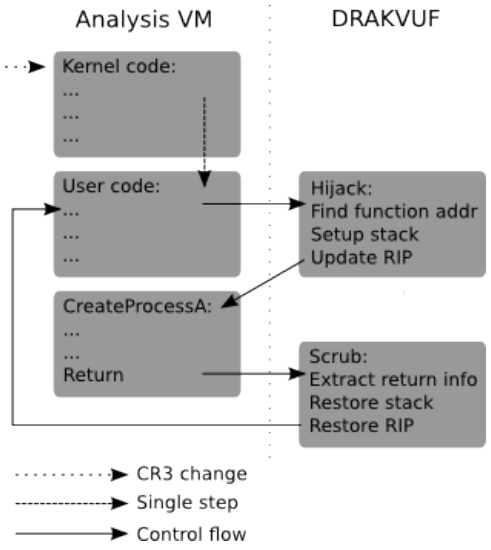

Fig. 5: Process hijacking employed in *DRAKVUF* to externally start the execution of the malware sample.

table. The stack of the process is updated to contain the input arguments required for calling the function, as well as the `RCX`, `RDX`, `R8` and `R9` registers on x86_64, while the original content of the `RIP` register is pushed as the return address on the stack. The return address is further injected with a breakpoint to notify us when the routine finished. The execution of the analysis VM is resumed after placing the address of CreateProcessA in the `RIP` register.

Our focus in *DRAKVUF* has further been to maximize the *throughput* of the physical machine used for the analysis. While the performance overhead imposed on the execution of individual malware samples is important and should be kept to a minimum, in our opinion such overhead is only of concern if it actively interferes with the analysis. In our system we take advantage of Xen's copy-on-write (CoW) memory feature to limit the memory requirements, and the LVM CoW disk capability to limit the harddrive space requirements on the host machine. With the combined use of these CoW techniques we are able to greatly increase the number of analysis sessions that can be performed concurrently on a given physical machine.

The CoW techniques employed by *DRAKVUF* require the presence of a static domain whose disk and memory can be used as a reference point, referred to as the *origin domain*. The origin domain can be configured as a regular domain before it is cloned, then once a clone is created it remains statically frozen. When cloning is initiated, first the LVM CoW disk is setup, followed by Xen creating the domain for the clone VM. The content of the origin domain's memory is piped into the newly created clone domain, immediately followed by memory de-duplication via memory sharing. After memory sharing, the only memory overhead is the memory allocated for QEMU to provide disk and network I/O to the clone, and the pages where breakpoints are injected. In our tests, the creation of a full VM clone with 2GB of CoW memory and CoW disk was performed in less than 10 seconds.

V. FUTURE WORK

A. Intel Virtualization Exceptions

As virtualization extensions continue to evolve, alternative security models become more viable for defense security research that were previously unable to guarantee secure isolation. In recent years Intel has introduced a new extension, dubbed #VE (short for Virtualization Exceptions). This has been in direct response to the defensive research community's finding that two-stage paging (such as Intel EPT) based tracing present significant overhead that can prove prohibitive for certain applications. The first - and thus far only - feature known that will use the #VE mechanisms is the newly introduced `VMFUNC` instruction's `EPTP switching` option.

The EPT extension on Intel has been from the beginning capable of maintaining up to 512 distinct EPTs in the `VMCS` for each vCPU. Nevertheless, all modern open-source hypervisor currently use only one EPT shared across all `VMCS` of the VM. However, in case the hypervisor used more pages, it would be possible to maintain one restrictive table used for trapping and one table for allowing the execution to flow normally. The `VMFUNC` instruction is designed to allow switching between such "views" from within the guest. Further allowing EPT violations to be selectively delivered to the guest in the form of interrupts without performing a `VMEXIT`, the performance overhead of the EPT based tracing can be significantly reduced.

As the hypervisor can still selectively configured which EPT entries are injected via #VE into the guest and which ones are still trapped to the hypervisor, the hybrid model of using both in-band and out-of-band delivery model first proposed by Payne et al. [14] is becoming a lot more viable in the near future.

The first open-source implementation by Intel for the upcoming #VE extension has already been proposed for the Xen Project Hypervisor 4.6 release, and has been dubbed the *altp2m* subsystem. In a joint effort, we have worked with Intel to enable this feature to be used in a purely out-of-band monitor scenario as well. The core feature added to Xen that has immediate use-case for out-of-band scenarios is the support added for multiple second-stage pagetables. With the possibility of maintaining multiple sets of tables, it is now possible to selective switch the view only on the violation-causing vCPU. While this method still doesn't obtain the performance benefits of the hybrid in-band/out-of-band model, it will enable our prototypes to perform analysis on multi-vCPU guests as well.

B. Mobile malware

With the rapid growth of the use of smart-phone devices there has already been a rapid expansion of malware targeting this new platform [**?**], thus malware analysis has to be able to also observe these types of malware. In our evaluation we have performed an initial exploration of the ARM CPU's virtualization extensions to support the type of malware

analysis discussed in this thesis. As part of this effort, we have implemented an initial set of tracing features based on the ARM two-stage paging mechanisms that will be available as part of the Xen Project Hypervisor 4.6 release. For execution tracing the Secure Monitor Call (SMC) instruction has already been proposed as a viable method to further improve the monitoring capability [1], [9], but there is no hypervisor support present for it yet. Furthermore, at this time there is no singlestepping support available for ARM platform which forms another obstacle to porting our prototypes to the ARM platform. As such, further research is required to properly identify all the capabilities of the ARM platform.

C. Data-only malware

As new and thus far unknown forms of malware appear, malware analysis systems will also need to adapt. In recent research Vogl *et al.* [23] demonstrated the feasibility of persistent data-only malware, capable of performing arbitrary computations without inserting any new code into the system. The detection and analysis of such malware will pose a particular challenge in the future, albeit some proposals have already been made to potentially detect such malware using performance monitoring counters [27]. These counters have been shown in prior research to be trappable to the hypervisor [22], thus malware analysis systems will need to adopt its monitoring strategy accordingly.

VI. CONCLUSION

Cyber crime is a problem that will continue to plague us for the foreseeable future. The criminals have evolved into sophisticated organizations with vast infrastructures, complex software development operations, and sophisticated supply chains. The sheer complexity and volume of information currently required for analysts to attempt understanding and perform mitigation against this activity is simply overwhelming. Unfortunately, this problem is steadily growing and shows no signs of abating anytime soon.

We need to learn from our peers in the financial and organized crime sectors if we wish to thwart cyber crime. Simply put, our analysts need to be empowered. We need to provide them with the tools that allow them to effectively do the jobs and identifying the 5Ws (Who, What, Where, When, Why, and How) of cyber criminal activity. Only then will we be better equipped to disrupt the infrastructure used and hinder the actors behind the development, distribution, and financial/informational gains associated with cyber crime. The previous path of only trying to protect our systems from an arguably infinite number of possible ways they could be exploited and hijacked has only provided little if any observable results. If we do not change our ways and look towards our analysts, will we be stuck in an arms race between malware and defensive software, without an end anywhere near in sight.

To this end we developed the Skald model and two implementations, Totem and Drakvuf. We have shown how these tools can overcome many of our current challenges and provide a solid platform to enable our analytic teams. Our implementations can scale easily in the millions, easily incorporate new and old techniques, and gracefully handle failures. Yet our systems are only the beginning and we hope others adopt our techniques to create competitors and assist us with developed of our open products.

REFERENCES

[1] Ahmed M Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. Hypervision across worlds: real-time kernel protection from the arm trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 90–102. ACM, 2014.

[2] Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Efficient detection of split personalities in malware. In *NDSS*, 2010.

[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] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *SIAM International Conference on Data Mining*, volume 2, 2011.

[5] Kim-Kwang Raymond Choo. The cyber threat landscape: Challenges and future research directions. *Computers & Security*, 30(8):719–731, 2011.

[6] Zhui Deng, Xiangyu Zhang, and Dongyan Xu. Spider: Stealthy binary program instrumentation and debugging via hardware virtualization. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, New York, NY, USA, 2013. ACM.

[7] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008.

[8] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44, 2012.

[9] Xinyang Ge, Hayawardh Vijayakumar, and Trent Jaeger. Sprobes: Enforcing kernel code integrity on the trustzone architecture. *arXiv preprint arXiv:1410.7747*, 2014.

[10] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007.

[11] James S Okolica and Gilbert L Peterson. Extracting forensic artifacts from windows o/s memory. Technical report, DTIC Document, 2011.

[12] Mike P Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE, 2003.

[13] Mike P Papazoglou and Willem-Jan Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16(3):389–415, 2007.

[14] Bryan D Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008.

[15] Bryan D Payne, MDP de Carbone, and Wenke Lee. Secure and flexible monitoring of virtual machines. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007.

[16] Rekall. `https://github.com/google/rekall`.

[17] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[18] National Threat Assessment Ctr US Secret Service, United States of America, CERT® Division of the Software Engineering Institute, United States of America, CSO Magazine, and United States of America. Us cybercrime: Rising risks, reduced readiness key findings from the 2014 us state of cybercrime survey. 2014.

[19] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[20] VirusTotal. Statistics. `https://www.virustotal.com/en/statistics/`, May 8 2015.

[21] Paul Vixie. Internet security marketing: Buyer beware. http://www.circleid.com/posts/20150420_internet_security_marketing_buyer_beware/, April 20 2015.

[22] Sebastian Vogl and Claudia Eckert. Using hardware performance events for instruction-level monitoring on the x86 architecture. In *Proceedings of EuroSec'12, 5th European Workshop on System Security*. ACM Press, April 2012.

[23] Sebastian Vogl, Jonas Pfoh, Thomas Kittel, and Claudia Eckert. Persistent data-only malware: Function hooks without code. In *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[24] Volatility. https://github.com/volatilityfoundation/volatility.

[25] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE*, 5(2), 2007.

[26] David D Woods, Emily S Patterson, and Emilie M Roth. Can we ever escape from data overload? a cognitive systems diagnosis. *Cognition, Technology & Work*, 4(1):22–36, 2002.

[27] HongWei Zhou, Xin Wu, WenChang Shi, JinHui Yuan, and Bin Liang. Hdrop: Detecting rop attacks using performance monitoring counters. In Xinyi Huang and Jianying Zhou, editors, *Information Security Practice and Experience*, volume 8434 of *Lecture Notes in Computer Science*, pages 172–186. Springer International Publishing, 2014.