TFNr: An Approach to Galvanized Malware Analysis

Kevin Kusiak
Department of Computer and Information Technology
Purdue University
West Lafayette, IN, U.S.A.
kkusiak@purdue.edu

Tahir Khan
Department of Computer and Information Technology
Purdue University
West Lafayette, IN, U.S.A.
tmkhan@purdue.edu

***Abstract*—Malware analysis has been a staple of the defensive security industry for multiple decades, and common static and dynamic analysis techniques have been devised and deployed in sandboxed environments. However as malware has evolved, previously stable analysis environments have become unstable in the sense that malware has grown to detect its virtualized execution environment and shuts down either partially or entirely, ruining the possibility of any signature capture. In this paper, a novel blackhole routing protocol, TFNr, is introduced and implemented in a mediocrely large malware analysis sampling environment. A custom structured control flow algorithm is also devised to aid TFNr in directing malignant and benign traffic and effectively storing and parsing obfuscated malware binaries. The proposed implementation of TFNr was implemented in a closed, compartmentalized, system existing on QubesOS to properly partition network segments and provide a mutation engine for malware matching various architecture signatures.**

***Index Terms*—network partitioning; malware analysis; sandboxing; compartmentalization; generative model; temporal model; TFNr; blackhole routing; structured clustering**

I. Introduction

In recent years, containerization has made a prominent use case over hardware virtualization. Containers provided a stripped-down and easy-access way to sandboxing services and applications for easy deployment and shipment. In a malware analysis perspective, both containers and virtual machines (VMs) are being heavily used to sandbox, execute, and analyze the effects of malware samples. However, this poses a major flaw due to the security defects of both containers and VMs due to their inherent and obvious constriction. These defects deal with anything concerning privilege escalation and container/VM escapes, as well as killswitch tendencies that are present in many homomorphic malware binaries as they detected execution in a virtual environment and discard themselves from the runtime environment.

Physical separation from a network is also in many cases ruled out as modern malware has grown to detect and exfiltrate data from air-gapped systems. Nation-state threat actors have grown to target USBs and peripherals on non-networked computers to escape these physical barriers [1].

This is where the core concept and functionality of compartmentalization adeptly comes into play. QubesOS is one of the pioneers of the usage of compartmentalization of the XEN virtualization architecture to truly sandbox applications and services. Compartments are in essence virtual machines connected by one singular backend, thus abstracting the ”virtual” aspect of a virtual machine [2]. Each compartment is only connected by segmented XEN net backends, therefore constricting external and internal connections from malware.

Based on these breakthroughs and unique benefits of compartmentalization, we have devised a method in which malware can be executed without avail and analyzed without any threat on the host operating system or any neighboring systems. Based on current states of the malware analysis compartment (henceforth known as a qube), a zero-day resistant environment can be coerced with custom generative networks that can emulate and invoke expectable user behavior.

This proposed approach includes a novel protocol labeled as the triadically functional network redistribution protocol (TFNr) to simulate a granular network environment in which a malware sample has every expectation and viewpoint on a potentially external network, all the while being present in a separately compartmentalized environment. This algorithm, alongside an implementation of a Xen-based compartmentalized system where abstractions in hardware are effective enough to efficiently sandbox and classify malware through means of signature clustering.

II. Landscape Analysis

A. Batch Signature Collection

Software compartmentalization is a relatively new technology, however the benefits it poses prove to be extremely useful in intelligent analysis of self-executed dynamic malware. This specialized form of malware analysis can be used to cluster families and groups of malware samples. Kirk and Rima [3] propose a novel method for automatic signature clustering, namely one generated by Hyperion binary static analysis that results in a structured control flow graph (SCFG) of malware variants. A malware binary is accepted as an input argument and is stepped through, having Hyperion work as a debunker for control flow obfuscation. At the end, the SIFT3 string edit distance algorithm is then employed alongside a quality threshold (QT) measurement, which then outputs a reasonably sufficient classification of the malware sample. This method boasts a time complexity of: Omin(n,m)maxOffset

As reasonable as this implementation is, obfuscation has been seen to alter signature comparisons in this de-generative model. Subroutines are reorganized, null values are inserted, and a variety of methods are undergone to deter automatic malware analyzers. Jin-Young and Sung-Bae [4] propose a solution based on variational autoencoding (VAE) that has two aspects: an encoder and a decoder. The encoder learns data in a defined latent space while the decoder reconstructs these data points in the latent space. After this initial generalization, subsequent malware samples can be detected to a much simpler extent and stored in a central-image repository.

The problem that this approach faces is the actual execution of the malware at hand. Although a detection and mutation engine can be manipulated and imaged in VAE, the malware still has a chance to escape an air-gapped or hardened malware analysis system, thus compromising the internal network as well as invalidating the malware image capture. From this comes one of two potential solutions, the first being the CloudIntell malware detection system [6].

This approach uses Amazon EC2 instances with so-called ”detection queues”. Requests by a client (a malware analyst in this case) are forwarded into the detection queues which then uses features extracted from the submitted portable executables (PEs) to make training samples for future analysis. In doing so, malware can be run in a potentially sandboxed environment that not only will raise detection rate but will also lower resource consumption by the host.

As beneficial and effective as this may seem combined with some variable SCFG implementation of clustering, malware on its own has grown to be highly dynamic and intelligent and can worsen false positives in future detections. As stated later in the study on CloudIntell by Younas et al., ”To make decisions whether a file is clean or malicious and that too accurately and instantly, the detection engine requires constant and rigorous training” [5]. Scalability and the approach towards generative malware in this approach is extremely weak and prone to failure. Although being fed through multithreaded and asynchronous detection engines, malware can behave differently under load especially with the growth of artificially intelligent cases. This is why an environment needs to be adapted where malware can be ran without any internal interference.

This is where Capsicum [6] is proposed as a middle-man for cloud sandboxing and bare-metal containerized options. Capsicum is more or less an application programming interface (API) rather than a full-blown OS. Capsicum employs privilege separation (henceforth known as compartmentalization) with the addition of capabilities. Since capabilities extend UNIX objects they can be delegated dynamically with minimal overhead, allowing for a wide array of kernel-level primitives and user space support modules to be utilized to balance client services over the same hardware stack.

Although this does indeed sandbox both kernel and user-mode services, they still both have access to the same hardware stack. Capabilities only go up logically, they don’t affect parent processes of the master node. This means that one single-level kernel-level exploit allows for pivoting between capabilities and compromising both the integrity of the malware as the malware analysis workstation(s). A visualization of the capabilities paradigm that Capsicum offers can be seen below in Figure 1.

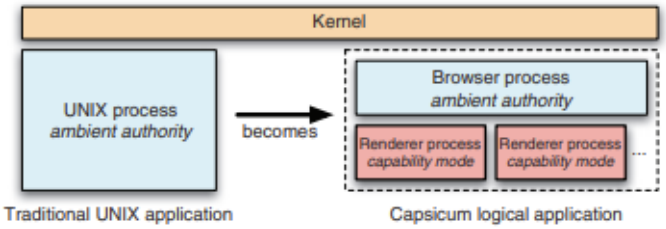

Fig. 1. Capability visualization for Capsicum [6]

B. Network Partitioning Schemas

This is where network partitioning comes into play, namely the incorporation of hyper-heuristics to find a low-cost number of network partitions in order to ”generate and optimize a population of executable programs” [7]. Pope and Tauritz recommend a multi-level graph partitioning method for networks consisting of three subsects: ”coarsening, partitioning, and refinement”. The coarsening phase of network segmentation uses a set of relatively smaller graphs to visualize a network graph via continuous sets of vertices. Partitioning then takes this set of decomposed graphs to locate the smallest approximation graph which is then used as the input to a looping set of calculations until the partition matching the original input graph is located [7].

This creates a potentially feasible alternative to physically air-gapped systems and networks for the execution of malware. The malware (assuming to be of an intelligent state) will be stuck in a logical calculation loop as every consequent calculation of the host operating system network appears to be a legitimate end-point, when in fact it is only a derivative and non-networked node.

Regarding the use of blackhole routing to make a generatively large ad hoc network, specifically a mobile ad-hoc network henceforth known as a MANET, the K-nearest neighbor algorithm comes into play [8]. This algorithm allows for fuzzy inferences to be intermingled into a randomly-generated node array in which neighborhoods of routing nodes are calculated. Each node in this array is appropriately placed in a sporadic cluster, in which each node calculates a circular level of trust and exchanges information with any and all information nodes. Stemming from this, a node is nominated for a cluster head via process of neighbor reliability lookup (K-nearest neighbor algorithm) [8].

This method of routing also introduces the concept of the AODV (ad-hoc on demand distance vector) routing protocol, which in turn is a demand-based routing protocol that transmits messages internally in a network based on unicast, multicast, and broadcast message paths [8]. This routing protocol is forwarded through four main message notations [8]:

1) An overview route request (RREQ) that creates route from a specified source to a destination
2) A route reply (RREP) that sends notifications in a reverse order from a destination node to the source node
3) A router error (RERR) which generates path error messages on transmitted data
4) A route reply acknowledgement (RREP-ACK) which confirms the existence of a route.

The visualization of this protocol at a packet implementation level can be seen below in Figure 2.

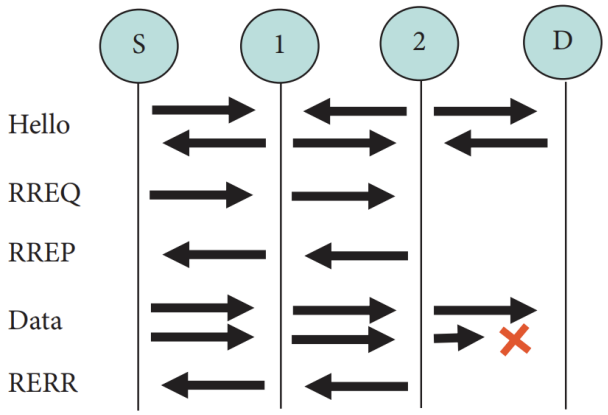

Fig. 2. Path discovery process in AODV [8]

Although this implementation is strong for data in transit, it is not strong in the sense of malicious forged transmits, with the concept of malformed data at rest. TFNr aims to fix these implementations and provide for a complex internal hierarchy of malware detection.

Another valuable and novel black hole routing protocol is GPSR, which is the greedy perimeter stateless routing protocol [9]. This protocol aims to implement a greedy forwarding schema as well as a perimeter forwarding topology. GPSR packets are specifically marked by a destination addressing schema, thus allowing for locally optimized and greedy choices to be made on the behalf of next hops in a routing table [9]. GPSR allows for seamless interface queue traversal, promiscuous sniffing capabilities off of a network interface, and support for Layer 2 level feedback [9].

This protocol combined with a very simply implemented beaconing algorithm allows for a routing-like table in which all relative neighbors are listed and encoded as two four-byte floating point quantities that are sequentially jittered by 50% to uniformly distribute the inter-beacon transmissions [9].

This approach allows for a planarized graph to be made in which the vast majority of routes in an intranet are empirically located. From this originate a so-called relative neighborhood graph (RNG) and a gabriel graph (GG) which equally allow for an algorithm to be implemented in which a network would be devised and appropriated where there are no crossing links between routes [9].'

The implementation of the beaconing algorithm along with greedy forwarding can be seen in the top half of Figure 3, while the implementation of perimeter forwarding via sequential positioning can be seen in the polygonal figure in the bottom half of Figure 3. There is a significant downfall here in the implementation of GSPR, as it is not able to properly gauge costs in a mixed environment, leading to potentially malformed and maliciously forged transmits propagating throughout a forwarding schema.

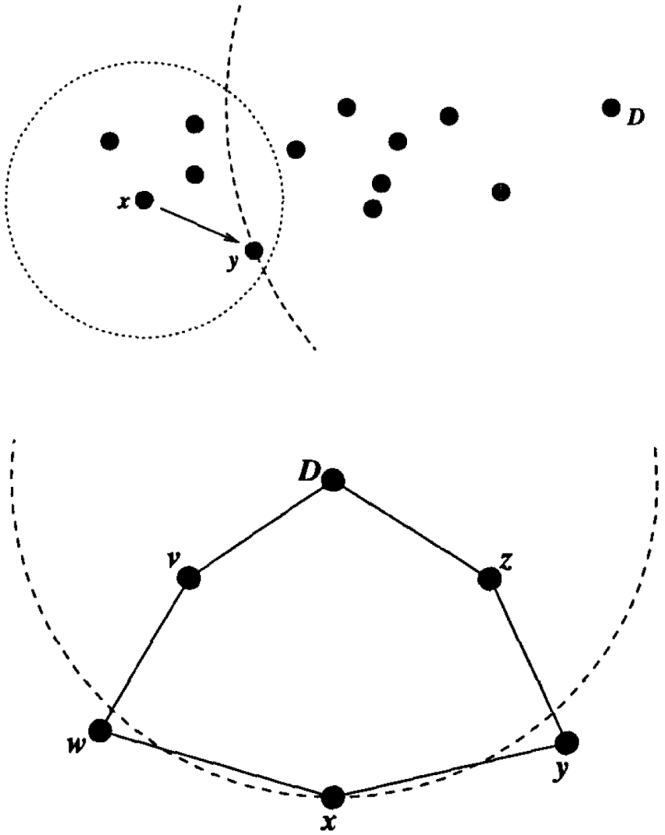

Fig. 3. Dynamic forwarding for GPSR [9]

An algorithm that can be implemented in situations where network optimization is needed when dealing with maliciously logged connections is the cuckoo search algorithm [10].

This algorithm allows for local search and global search capabilities, which is inline with the general concept of discovery probability. This concept references the fact that this cuckoo algorithm incorporates a method of particle swarm optimization, which satisfies globally network-wide optimization requirements in a large ISP-area network as well as supports multipathed and multimodal optimization.

The algorithm, as seen below in Figure 4, references both a local random walk as well as a global explorative random walk being implemented by Levy flights, where a dynamic Markov chain is being input into a status quo where the next location depends on the current location and vice versa [10].

C. Network Partitioning Schemas

Moving past the logical implementation of networking, the next possible step to overcome when implementing a

safely air-gapped system for malware analysis is a moderately efficient method to analyze effects of malware and logging the results for future detection. This consists of the Shapley Ensemble Boosting and Bagging approach, proposed by Kumar and Subbiah [11].

Their approach capitalizes in the point that malware authors have various inconsistencies in their crafted binaries which can effectively help ”distinguish malware and benign software”. The various methods of packing incorporated in the crafted binaries increase bytewise entropy, allowing for dynamic analysis and identification of ”polymorphic and metamorphic engines that attempt to obfuscate identically performing malware” [11].

Kumar and Subbiah propose a bagging model built around the gradient boosting decision tree (GBDT) machine learning (ML) model. This allows for practically infinite visualization of capabilities and equally so infinite visualization of interactive and intuitive callbacks by potentially malicious binaries. A Shaplay value, defined as ”the true representation of the amount of contribution of features” [11] allows for a theoretical minimum of false negatives when it comes to benign software. As a result of their study, it was discovered that their bagging and analysis ML models approximately resulted in 97.87 and 97.50 percent accuracy when it came to predicting future zero-day malware. The outliers and potholes in this data prove a substantial amount of worth to unpacking and executing obfuscating binaries. Being able to execute the programs in a time-sensitive and resource-sensitive manner would have allowed for a higher detection rate as well as a larger (and safer) retention rate when it comes to training future models. This bagging model can be seen visualized below in Figure 4.

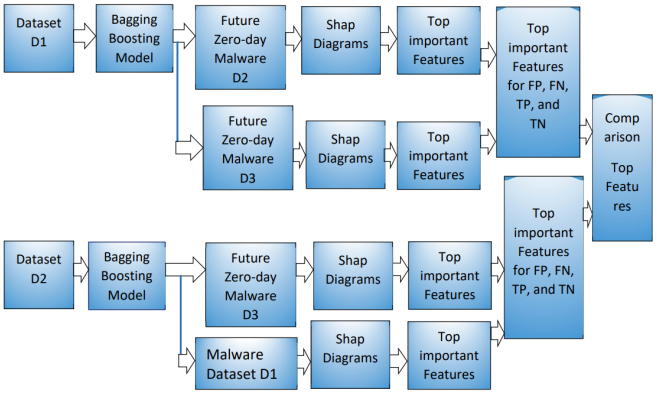

Fig. 4. Shapley Ensemble bagging and boosting method [11]

The large problem proposed by the previously mentioned bagging approach is the concept of polymorphic malware as it is multi-threaded and multi-layered in nature, proving to be difficult for ML models to detect and approach. As discovered by Wanswett and Kalita [12], polymorphic malware can ”reorder code, reassign registers, and obfuscate control flow”. This is where it becomes important to apply a hierarchy of malware detection techniques to properly weed out false positives and false negatives in the dataset.

These 5 tenants are as follows: ”principal component analysis (PCA) as a statistical method, multilayer perceptron model to generate signatures for malware clusters, machine learning to extract features and mine data, hamsa to act as a network based signature generated system, and finally an efficient hybrid technique to act as a suspicious traffic filter (STF), zero day attack evaluation (ZAE), and a signature generator (SG)” [12]. These definitions and instantiations of network-based zero-day polymorphic malware detection techniques provide a strong basis for not only signature-based detection,but also heuristic and behavioral based [12]. As robust as these techniques may be for various types of polymorphic malware, the line between malicious and benign queries can easily become skewed and lead-to long term network intrusion from intelligent species of malware. This enhanced mutation engine-based approach to malware analysis can be seen logically enumerated below in Figure 5.

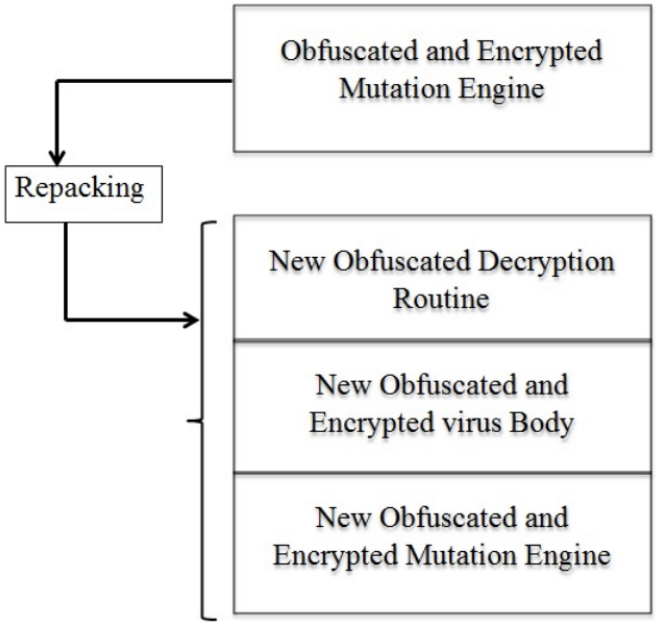

Fig. 5. Mutation Engine Depacking Routine [11]

To move past the data collection and alignment for polymorphic malware as well as deter any network-based escapades, a robust classification method needs to be implemented to properly classify propagation of malware in various (such as Windows, Linux, MacOS, and BSD) systems. This is where a support vector machine (SVM) as proposed by Kumar and Singh [13] comes into play.

The SVM is a supervised learning algorithm that ”classifies an executable as malicious or benign or whatever class that we want to retrieve” [13]. Combined with a sandbox, the SVM can monitor all metadata and metanetwork access that a potentially malicious executable exhibits. The SVM ”builds linear functions from the set of the given labeled training

dataset and then tries to cleave the samples into two different classes of negative and positive samples" [13]. The output of these functions in a multi-SVM design could easily be used in a granular and network-segmented compartmentalized sandbox in order to potentially classify various types of malware and store these classifications in an external database for further analysis or public release.

Being that the biggest downside of SVM and the combined method of sandboxing could be potential disallowments of malware in a perceived sandboxed environment or attacks on hardware or hardware-abstracted software, a novel solution is proposed by Rushdan et al. [14] for using the Cuckoo sandbox to abstract hardware in software-defined networks (SDN).

As stated by Rushdan et al. [14], the proposed solution implements a "mininet simulation tool , in order to implement SDN, forwarding switches, and the client PCs". This approach followed with a torus-like architecture for a finitely infinite network architecture based on some sort of content-centric networking (CCN) could prove to be a useful network-based sandbox. Agents could heftily be deployed at network end-points to enumerate and isolate potential threats and outbound requests initiated by malicious binaries.

D. Emulating Prediction Datagrams

As malware depends on predictable and real-time user input and output, a reputable, generative, and repeatable behavior manipulation model needs to be implemented. Kun et al. proposed a non-degenerative model, named the User Behavior Emulator (UBER) [15], that captures long-term realistic user data from a set of users on a system. This data is then manipulated by automation control modules which, as again proven by Kun et al., defeats the anti-sandbox detection mechanisms put forth by malware.

Similar to the concept of emulated user behavior comes the concept of unstructured input sources, namely through means of user information taxonomies. As identified by Olukoya et al., in basic user-level systems (that is, systems in which user input is a necessity) malicious apps can be visualized as a generic commodity [16]. This study goes further on to draw a connection between malicious user behaviors and the user of unstructured inputs, thus resulting in an insufficient and permission-based privilege hierarchy.

Once the user element is either compromised or proved to be benign, the next focus area for malware is usually the process spectrum, more specifically being executable operations. According to a study by Tobiyama et al., a convolutional neural network was created and trained onto different endpoints and feature images that were respectively denoted as either malignant or benign [17]. Following the classification of this training data, it became apparent that identifiable clusters of malicious data follow a very uniform distribution control set, and equally so resemble each other in some sense even if the process binary itself is unique [17].

III. Experimental Design and Methodology

A. Environment Design

For the initial environment setup for the implementation of TFNr, a multi-faceted approach was taken to first filter out malicious queries and segment proper signature classification clustering so that the malware can properly execute in an enclosed and simulation-free environment.

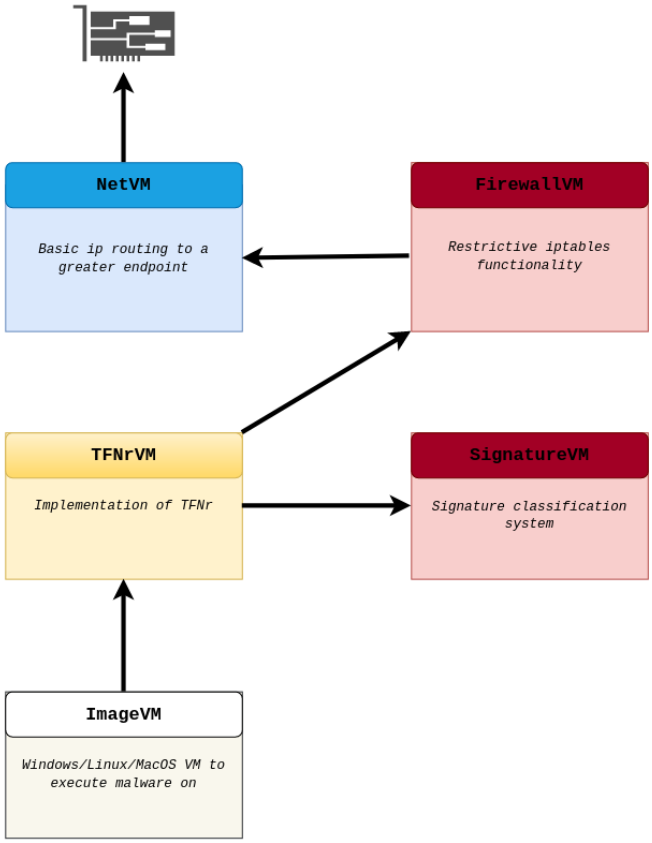

Fig. 6. Environment paradigm

The Qubes operating system, operating on top of the Xen hypervisor was implemented running five major qubes (individual virtual machines) as seen above in Figure 6:

1) A network virtual machine to provide a basic routing interface
2) A firewall virtual machine to implement heuristic methods and a very robust way of filtering out negative gateway connections and inbound and outbound packets
3) A signature virtual machine to implement a custom signature clustering approach
4) A TFNr virtual machine to implement an instance of the TFNr protocol
5) An image virtual machine to act as a logical placeholder for either a Windows, Linux, or MacOS platform on which to execute the malware on

The network virtual machine was a simple Layer 3 implementation of a forwarding daemon, basically acting as the virtual network interface card (NIC) of the system and probing

traffic through an intermediary out to the internet. The firewall virtual machine was also a simple implementation containing simple *iptables* rules to ensure a minimally viable and working system.

B. Signature Algorithm

The signature clustering algorithm on the signature virtual machine was a custom SCFG in which a fork of the original Hyperion SCFG was combined with a generative deep learning model containing a variational auto-encoder. This algorithm, part of the Kullback-Leibler divergence hierarchy, used a simple distributional model through a CNN (convolutional neural network) to initialize and segment malware clusters, ultimately sorting them through a quality threshold clustering algorithm (QTC). By clustering these signature samples in a centralized location through methods of control flow malware analysis, signatures are more likely to generate truthful results and will generate conditionally less false negatives.

This signature clustering method depends on the mathematical analysis seen just below. Given the superset of variables alpha and beta, in relation to the respective probability of a process (string) being benign or malicious, an argumentatively-based evaluation is done on the restricted internal set of benign packets as compared to the external set of malicious packets.

$$\supset \alpha, \beta \int E(b|\alpha, \beta)$$

This evaluation can be summarized via the Gaussian kernel to the expression below, depicting the idea that a true homomorphic evaluation of malware can be conduced with the equation calculating the relative frequency of benign packets in relation to a set of benign and malicious packet frequencies.

$$(\alpha)/(\alpha + \beta) = (\rho + 1)/(\rho + \alpha + 2)$$

This expression turns out to be within the acceptable normal distribution as produced by the Gaussian distribution, and equally so the Gaussian kernel, as distinctly shown using the transform equation as depicted below.

$$X \sim \mathcal{N}(\mu, \sigma^2)\,.$$

This algorithmic implementation of a homomorphically existent mutation engine was tied with a quantitative set of Markov chains [19] to construct a valid trace analysis and graph generation. An Ether system was commutatively defined to track a certain subset of processes (EXE and ELF processes) and dynamically compare each to a stock kernel process list(where it be Windows, Linux, or a BSD-based kernel) and thus flag traffic based on those measurements.

C. Multipathing Routing

To properly implement the official blackholing portion of this utopian algorithm, a properly constructed synergetic algorithm needed to be created. Very similarly to the AODV protocol, TFNr was constructed in the sense that broken links are fixed on a convergence-based theory through route maintenance, ultimately providing a software-level illusion of a multiprotocol switching mechanism (such as something similar in composure to MPLS).

Given this rough algorithm, a proper visualization can be made corresponding to nodes in a network paradigm as seen below in Figure 7. The RERR and FERR values are polars and respectively correspond to the values of the reverse endpoint registry request, and the forward endpoint registry request. The RTI represents an expected traffic endpoint, and respectively stands for the ”regular traffic inquiry”.

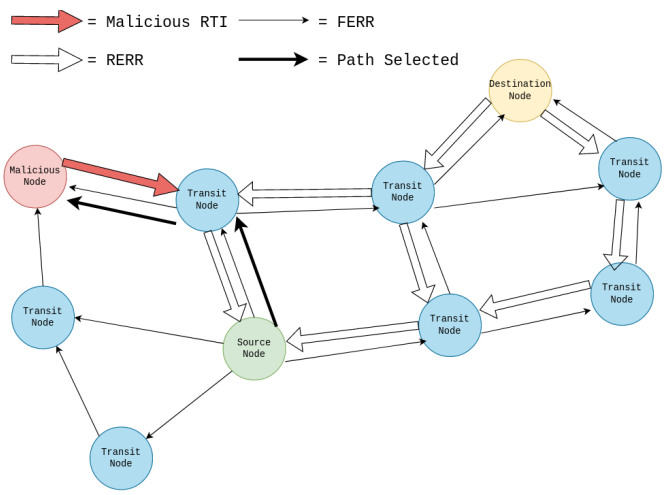

Fig. 7. Custom TFNr Network Structure

Traffic in TFNr will always originate from the entry node and go to the transit node with the least advertised cost. The first route from the entry node will always be to a transit node as the transit nodes will be inherently trusted as they contain the root trust for an application, which at its top-level attribute will never be compromised. Following this root trust node, the packet flow might end up traveling to a malicious node because of the least-cost principle, as seen above in Figure 7.

Upon arrival at a malicious node (or any node at that fact), the structured control flow algorithm is consulted to determine if the node is benign or malicious. If identified to be in a malicious state, the malicious node will attempt to connect back to the source node via some stateful protocol, but will be immediately black-holed and the probing process will restart again originating at the source node. This allows for malware to initiate an outbound extension to some outbound resource (direct host or a command and control server), but will never reach its destination due to means of corruption and blackholing, ultimately proving the malware useless.

At the same time, the destination node, if it is proven to be a stateful and established connection, will be similarly routed through TFNr and undergo a similar probing algorithm to make its way back to the source node, thus making this protocol turing complete.

The pseudocode for the generic TFNr algorithm can be seen below in Algorithm 1, where k represents the number of transient network connections, i represents the minimum

number of feasible indicators of malicious activity, and x represents the set of hashed indicators of compromise of preexisting malware samples. i and x are derived as parameters from the structured control flow algorithm described earlier.

Algorithm 1 Generic TFNr Algorithm

Require: $k \geq 1024$
Ensure: $i + 1 = x + 2$
  $forward \leftarrow True$
  $nexthop \leftarrow 0$
  while $true$ **do**
    $controlflow \leftarrow Signature(NODEattribute)$
    $nexthop \leftarrow controlflow \bigoplus NODEattribute$
    if $nexthop == 0$ **then**
      $forward \leftarrow True$
    else$nexthop == 1$
      $forward \leftarrow False$
      $break$
    end if
  end while
  if $nexthop = 1$ **then**
    $blackhole()$
  end if

IV. Results

A basic and reasonably conclusive structured control flow signature collection algorithm was successfully developed, alongside an implementation of the bare network partitioning capabilities of TFNr excluding any search optimization techniques as stated earlier. The entirety of this proof-of-concept was written in C++ and cython to optimize speed over reliability and memory safety. To test the enhanceability and effectiveness of the protocol and environment, a simulation was run on ns3 [20], which is an open source discrete-event network simulator for wide area networks. In the following tests, a control factor in the form of the signature clustering algorithm was defined. The custom signature clustering algorithm was used to identify malware acolades and properly direct traffic either through a blackhole or through a NIC out to the Internet. A sample size of 88,467 malicious nodes was generated through ns3 and used as a data set to ensure how efficient TFNr is over blackholing traffic as compared to its closet competitor: GPSR.

Three main attributes were captured when analyzing how effective the new blackhole routing protocol was:

1) totalTx: total transmit size
2) totalRx: total receive size
3) PDR: packet delivery ratio

The reason these three attributes were used for measurements was mainly because of the fact that they were standardized among GPSR. TFNr is a polymorphic routing protocol by nature, meaning with its interoperability capabilities, practically any routing protocol and its subsidiaries can be emulated.

Upon running both protocols (GPSR and TFNr) through the 88,467 sample size in ns3 and evaluating As seen in the visualizations below in Figure 8, an appealing results paradigm was created. The former of the two figures shown in Figure 8 represents a graph of the correlation for PDR versus total transmit size, while the latter shows a graph showing the correlation for PDF versus total receive size.

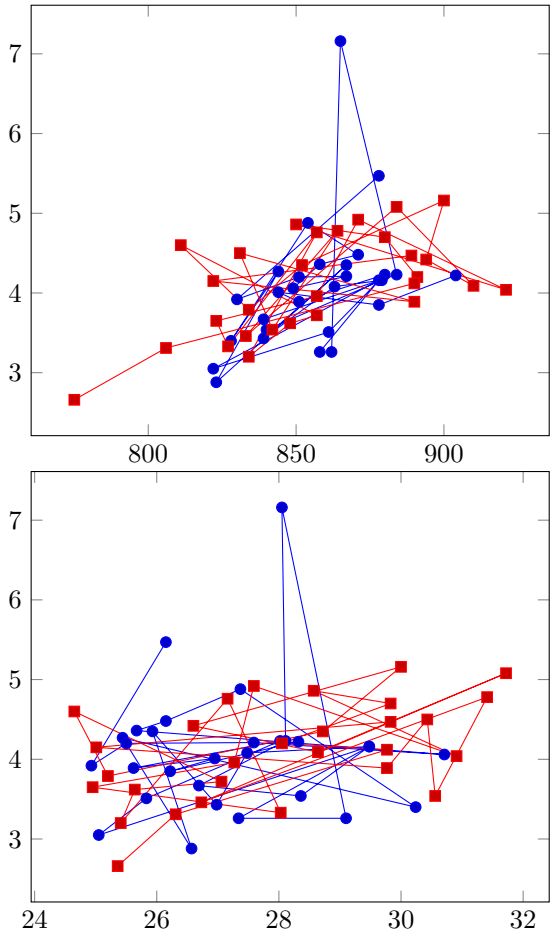

Fig. 8. TFNr Results

Based on coercive data manipulated from both these graphs and the test data, it is unfortunate to state that pre-optimization, the TFNr protocol does indeed not out perform GPSR and by an extension any other moderately advanced black hole routing protocol. In terms of this assessment, a lower PDR is ideal, and where the TFNr protocol reached a peak of 2.88% PDR, GPSR reached a peak of 2.66% PDR. Again, like stated earlier this implementation of TFNr is not currently optimized through the Cuckoo search algorithm and no methods of fitness have yet to been applied on the protocol, so this is a relatively expectable number.

V. Limitations and Future Work

What truly hinders the TFNr protocol from operating at anywhere near one hundred percent efficiency is the inexperience and immaturity the codebase contains. This codebase was written entirely by Kevin Kusiak, a 20-year old undergraduate

student, in the course of barely 7 weeks of work. On the other hand, the motivation and point-of-reference for TFNr was GSPR, which has nearly 25 years to acclaim to its maturity alongside a widespread team of researchers and security engineers that dedicated parts of their lives to improving and growing GSPR.

Aside from a pure implementation perspective, it was extremely difficult from a development perspective to figure out algorithmically how to parse through obfuscated binaries and direct traffic to or from black holes without inducing any false negatives. This was done through quick work-arounds by cross-checking process identifiers with other known malware samples or data sets at runtime, but will need to be something to be worked out in the future.

A final downfall in the implementation and research of network partitioning can be seen in our applied concept of end-to-end registry of network connection tracking and for expected traffic endpoints. A serious limitation in this scope is that network connection tracking only works as long as a packet transmits unaltered and uncorrupted encapsulated traffic and does not commit any further encapsulation or decapsulation between nodes. This network protocol capitalizes on the concept of plausible deniability, where only common traffic from layer three to seven will ever be forwarded through any node, not containing any layer two level traffic or those required complex application layer gateways. Additionally, a surprisingly large amount of false negative traffic gets let through between gaps in the network partitioning segments and gets injected into routing tables on a virtualized host, which is a point of discussion for the future.

VI. Conclusion

In this paper, we have presented the Triadically Functional Network Redistribution protocol, TFNr, to act as a novel blackhole routing algorithm that uses application states and a custom structured control flow signature algorithm to dynamically either route traffic through a NIC to the Internet or rather blacklist it and drop the packet(s). Our simulations on 802.3 exclusive networks can seemingly stimulate over 1,000,000 concurrent malware signatures and clusters with the network partitioning only being able to handle exactly one entry node, one exit node, and one malicious node. All the inherent benefits provided by TFNr stream from modular mobile ad-hoc networks, as they are principally cohesive and likeminded in the same goals of sporadic network blackholing. The proposed implementation of TFNr was implemented in a closed, compartmentalized system originating on QubesOS, and provided a multi-faceted approach and highly successful approach to the concept of safe and effective malware analysis. Although not being wholly optimized via fitness and search algorithms, the TFNr ended up reaffirming beliefs in the algorithm as it statistically made its way to the top of blackhole routing algorithms in terms of packet delivery ratios in conjunction with malware detection and mitigation percentages.

References

[1] I. Arghire, "17 Malware Frameworks Target Air-Gapped Systems for Espionage," SecurityWeek, Dec. 03, 2021. https://www.securityweek.com/17-malware-frameworks-target-air-gapped-systems-espionage/ (accessed Feb. 21, 2023).

[2] J. Rutkowska, "Software compartmentalization vs. physical separation (Or why Qubes OS is more than just a random collection of VMs)," 2014. Available: https://invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_separation.pdf

[3] R. A. Awad and K. D. Sayre, "Automatic clustering of malware variants," 2016 IEEE Conference on Intelligence and Security Informatics (ISI), Tucson, AZ, USA, 2016, pp. 298-303, doi: 10.1109/ISI.2016.7745494.

[4] J.Y. Kim and S.B. Cho, "Obfuscated Malware Detection Using Deep Generative Model based on Global/Local Features," Computers & Security, vol. 112, p. 102501, Jan. 2022, doi: https://doi.org/10.1016/j.cose.2021.102501.

[5] Q. K. Ali Mirza, I. Awan, and M. Younas, "CloudIntell: An intelligent malware detection system," Future Generation Computer Systems, vol. 86, pp. 1042–1053, Sep. 2018, doi: https://doi.org/10.1016/j.future.2017.07.016.

[6] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway, "A taste of Capsicum," Communications of the ACM, vol. 55, no. 3, pp. 97–104, Mar. 2012, doi: https://doi.org/10.1145/2093548.2093572.

[7] A. S. Pope and D. R. Tauritz, "Automated design of multi-level network partitioning heuristics employing self-adaptive primitive granularity control," Proceedings of the 2020 Genetic and Evolutionary Computation Conference, Jun. 2020, doi: https://doi.org/10.1145/3377930.3389819.

[8] G. Farahani, "Black Hole Attack Detection Using K-Nearest Neighbor Algorithm and Reputation Calculation in Mobile Ad Hoc Networks," Security and Communication Networks, vol. 2021, pp. 1–15, Aug. 2021, doi: https://doi.org/10.1155/2021/8814141.

[9] B. Karp and H.-T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," Jan. 2005, doi: https://doi.org/10.21236/ada440078.

[10] X.-S. Yang and S. Deb, "Cuckoo search: recent advances and applications," Neural Computing and Applications, vol. 24, no. 1, pp. 169–174, Mar. 2013, doi: https://doi.org/10.1007/s00521-013-1367-1.

[11] R. Kumar and G. Subbiah, "Zero-Day Malware Detection and Effective Malware Analysis Using Shapley Ensemble Boosting and Bagging Approach," Sensors, vol. 22, no. 7, p. 2798, Apr. 2022, doi: https://doi.org/10.3390/s22072798.

[12] B. Wanswett and H. K. Kalita, "The Threat of Obfuscated Zero Day Polymorphic Malwares: An Analysis," 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Dec. 2015, doi: https://doi.org/10.1109/cicn.2015.230.

[13] S. Kumar and C. Bhim Bhan Singh, "A Zero-Day Resistant Malware Detection Method for Securing Cloud Using SVM and Sandboxing Techniques," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2018, pp. 1397-1402, doi: 10.1109/ICICCT.2018.8473321.

[14] H. Al-Rushdan, M. Shurman, and S. Alnabelsi, "On Detection and Prevention of Zero-Day Attack Using Cuckoo Sandbox in Software-Defined Networks," The International Arab Journal of Information Technology, vol. 17, no. 4A, pp. 662–670, Jul. 2020, doi: https://doi.org/10.34028/iajit/17/4a/11.

[15] S. Liu, P. Feng, S. Wang, K. Sun, and J. Cao, "Enhancing malware analysis sandboxes with emulated user behavior," Computers & Security, vol. 115, p. 102613, Apr. 2022, doi: https://doi.org/10.1016/j.cose.2022.102613.

[16] O. Olukoya, L. Mackenzie, and I. Omoronyia, "Towards using unstructured user input request for malware detection," Computers & Security, vol. 93, p. 101783, Jun. 2020, doi: https://doi.org/10.1016/j.cose.2020.101783.

[17] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware Detection with Deep Neural Network Using Process Behavior," 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Jun. 2016, doi: https://doi.org/10.1109/compsac.2016.151.

[18] R. Linger, K. Sayre, T. Daly, and M. Pleszkoch.

[19] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," Journal in Com-

puter Virology, vol. 7, no. 4, pp. 247–258, Jun. 2011, doi: https://doi.org/10.1007/s11416-011-0152-x.

[20] nsnam, “ns-3,” ns-3, 2019. https://www.nsnam.org/ Function extraction technology: computing the behavior of malware. In System Sciences (HICSS), 2011 44th Hawaii International Conference on, pages 19. IEEE, 2011.