Firethorn White Paper

James Cao
Ironwood Cyber
james.cao@ironwoodcyber.com

Donal Lowsley-Williams
Ironwood Cyber
donal.lowsleywilliams@ironwoodcyber.com

Aaron Estes D.Eng
Ironwood Cyber
aaron.estes@ironwoodcyber.com

Ethan Puchaty
Ironwood Cyber
ethan.puchaty@ironwoodcyber.com

Abstract

A proprietary method of detecting ransomware based on initial research at the Darwin Deason Institute for Cyber Security at Southern Methodist University (SMU) has been implemented. The method utilizes an end-to-end big data machine learning pipeline to generate binary classification models that detect the presence of ransomware infections based on side-channel hardware sensor data.

1 Introduction

Ransomware, a form of malicious software known for its data encryption and extortion tactics, has surfaced as a formidable challenge within the cybersecurity landscape. Notably, this cyber threat has evolved into a highly lucrative criminal enterprise, causing unprecedented financial ramifications and operational disruptions. In the face of escalating ransomware campaigns, both the public and private sectors are grappling with the profound implications of these attacks, compelling a renewed research focus on the dynamic intersection of ransomware and ransomware detection. Recent estimates indicate that the global cost of ransomware attacks over five years in the financial sector alone exceeded 35 billion USD, affecting organizations of varying sizes and industries. The resulting financial backlash not only necessitates remediation costs but also translates into lasting reputation damage, litigation expenses, and potential regulatory fines. Beyond these financial repercussions, ransomware invasions present a fundamental challenge to data security, privacy, and the integrity of digital systems. Such attacks undermine the very essence of cybersecurity principles, infringing upon the confidentiality and availability of information. Conventional, signature-based detection approaches, which rely on predefined malware patterns, have inherent architectural faults that leave them weak to rapidly evolving ransomware strains. As such, the cyber community has assumed a pioneering role in the development of innovative methodologies and technologies. These advancements, grounded in behavioral analysis, artificial intelligence, and machine learning, have the structural foundation necessary to redefine ransomware detection. Firethorn is Ironwood Cyber's proprietary technology that utilizes third-party services along with custom code to gather data for model training, generate ML models, deploy them for inference, and alert on attacks. Additional capabilities are available that provide reporting based on the models used on Firethorn clients. This white paper delves into the technicality behind the Firethorn solution, its novel detection strategies, development status, and identified future developments. The findings and analyses of this white paper aim to equip organizations, stakeholders, and researchers with the technical deep dive behind Firethorn, fortifying their capacity to navigate the ransomware labyrinth with financial prudence and technologically informed resilience.

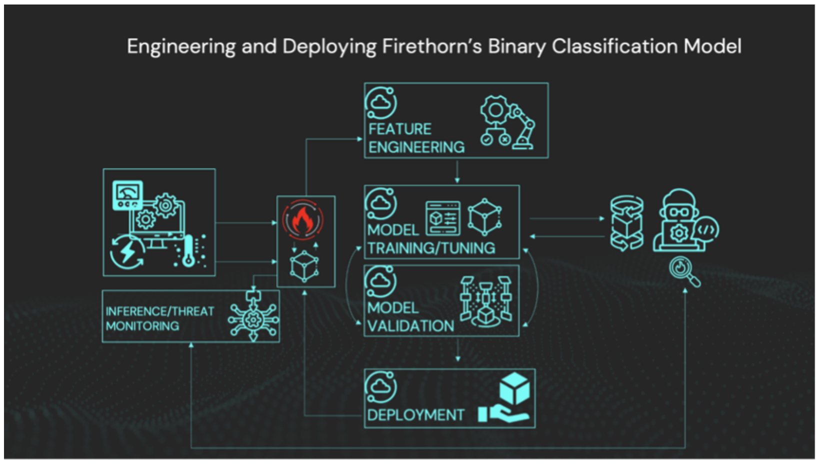

Engineering and Deploying Firethorn's Binary Classification Model
FEATURE ENGINEERING
MODEL TRAINING/TUNING
MODEL VALIDATION
DEPLOYMENT
INFERENCE/THREAT MONITORING

Figure 1: Model Training Overview

2 Methodology

In this section, the research methodology is outlined, highlighting the framework employed for developing the Firethorn model training pipeline, the acquisition of side-channel features, the intricacies of feature engineering, and the deployment of the training model for ransomware detection. The methodological exposition encompasses the overall approach taken, methodologies for data collection, analytical techniques employed, as well as the tools and resources instrumental in fulfilling the research objectives. The comprehensive coverage extends to all parts of the Firethorn model pipeline and the infrastructure designed to cater to a production-ready environment.

2.1 Data Collection

The initiation of the model pipeline commences at dedicated laboratory machines configured for conducting ransomware simulations. These machines systematically collect data from hardware sensors, encompassing parameters such as CPU temperatures, disk speeds, RAM power usage, and more, by leveraging Firethorn's driver. Simultaneously, machines equipped with Firethorn perpetually engage a machine learning model for real-time inference, making predictive assessments concerning the presence of an ongoing cyberattack. This multifaceted data is seamlessly transmitted to a real-time data streaming platform, facilitated by an Apache Kafka cluster, where it awaits subsequent processing to render it amenable for training purposes. The ransomware simulations are orchestrated on these machines utilizing Firestorm, an attack simulation software engineered to emulate real-world threat scenarios, including instances inspired by notorious ransomware strains such as WannaCry and Ryuk.

2.2 Side-channel Hardware Samples

Side-channel analysis is oriented toward the exploitation of information leakage arising from the physical instantiation of a computer system, in contrast to its logical operation. Firethorn, a proprietary driver designed for Windows Intel Machines is executed, facilitating direct interaction with the kernel for the acquisition of sensor metrics. Subsequently, each hardware sample is acquired as a floating-point value and is promptly allocated to the specific resource from which it was extracted, such as CPU or RAM metrics, within a frequency of 10 Hz. These hardware samples are subsequently transmitted to cloud-based storage. Concurrently, they are incorporated into a secondary one-second window and passed to the Firethorn model for the computation of inference-related metrics.

2.3 Firethorn Attack Simulator and Duststorm User-Behavior Simulator

Simulation execution is handled by Firestorm, an internally developed tool at Ironwood Cyber. The application is instrumental in conducting ransomware simulations on target machines. It operates through a configurable set of parameters, affording the flexibility to emulate real-world ransomware attacks more effectively. Duststorm is a secondary application companion that serves as a complementary application, contributing to the introduction of noise and benign activities on the machines. The primary objective of Duststorm is to train the models in a manner that mitigates false positives. There are a variety of different load-generating programs included within Duststorm, each with its own parameters aimed at enabling the collection of a good distribution of data that mimics real computer usage metrics by end-users at the hardware level.

2.3.1 Firestorm Design

Firestorm exhibits a versatile architecture akin to a "ransomware-as-a-service" framework, offering users the capability to meticulously configure various ransomware types for deployment in targeted attack campaigns. The platform allows for the fine-grained customization of encryption methodologies, victim file sizes subject to encryption, and the activation of behavior patterns mirroring those observed in real-world malware. Once the parameters are meticulously defined, they are primed for deployment, effectively launching an "attack" on the designated client machine. It is worth noting that Firestorm resides on the client machine but remains dormant until it receives activation parameters dispatched from a centralized orchestration service. An intuitive user interface has been developed below to facilitate the transmission of these parameters through the simulation API. In addition, the simulation API is called in an autonomous script to randomize attacks on client machines.

Firestorm leverages a multitude of patterns found in reverse-engineered malware strains, as well as insights documented in technical deep-dives, to replicate the characteristics of authentic ransomware attacks. For example, the simulator possesses the capability to engage in multithreaded encryption, as well as partial encryption for expedited compromise—aligning with methodologies reminiscent of Lockbit. Another example is the making sensitive files unrecoverable via the allocation of zero bits before deletion, like the Dharma strain. The overarching objective of Firestorm resides in the emulation of the strategic maneuvers employed by ransomware attacks, effectively subjecting computer systems to duress in a manner consistent with both established and novel ransomware variants. It is noteworthy that, despite the diversity among ransomware strains, the end goal of ransomware is the same and the resources utilized during encryption of files would not change.

2.3.2 Duststorm Design

Duststorm was developed with the primary aim of introducing controlled noise to the hardware infrastructure, specifically targeting metrics associated with RAM, CPU, and disk operations. During the earlier phases of investigation, the Firethorn system exhibited an impressive capacity to discern deviations in system behavior on test machines, yielding consistently accurate outcomes. Nevertheless, scaling out showed that Firethorn demonstrated a pronounced proficiency in distinguishing the steady or "safe" operational state of a computer system from any subsequent deviations. In practice, this attribute translated to an elevated risk of generating false positive alerts in scenarios where end-users imposed substantial computational burdens on their systems, potentially leading to unwarranted notifications. The design of Duststorm was built to generate noise and strain from an end-user to provide the training data with more information and in turn reduce the number of false positives. Things such as crunching together mathematical equations or printing the sequences of DNA are two "loads" that are available to run to simulate CPU usage. Duststorm simulations are also developed based on real world research. For example, background "loads" that can emulate user disk behavior, such as reading and writing to a file, was designed to write a specified number of bytes per write speed defined in the parameter of the experiment. Average bytes per second written was calculated as:

$$\alpha = (200char \cdot min)/60s \cdot 2bytesperchar$$

With a 50% type speed, that would evaluate to around 3.33 bytes per second. Duststorm creates the size of the bytes to be written to any file by taking that average and multiplying it by a randomized write time in between the minimum and maximum write time set within the experiment parameters. The simulation will also be inactive during a read time to emulate a user reading a file.

Below is an example image that shows another basic simulation of opening and closing a file. Duststorm will randomly call a number of these methods with defined parameters to generate a unique and randomized set of noise for the hardware samples.

2.4 Data Processing

Upon reaching the data processing phase, hardware and simulation data are funneled into Spark, a distributed platform tailored for extensive big data processing. This initial step involves transforming the data into a coherent, tabular format, accompanied by pertinent timestamps and machine identification. Leveraging the simulation data, entries are categorized as either "under attack" or "not under attack." Feature engineering is subsequently employed, facilitated by a proprietary C# library. This library is responsible for deriving time-dependent features and generating comprehensive statistical summaries based on input data. Notably, this library serves a dual purpose, as it is also engaged during live inference to ensure consistent feature calculations across both training and real-time scenarios. Feature generation represents the final stage of data processing, after which the data is securely archived in Azure blob storage, primed for the subsequent training phase. This whole phase is automated so that training can be done more easily and efficiently.

2.4.1 Flatten Data

Flattening the data not only streamlines preprocessing but also enhances interpretability and reduces the risk of information loss, allowing researchers and practitioners to harness the full potential of real-time, event-driven data from Kafka. Custom code is used to massage the hardware samples from Kafka into a flattened tabular format that can be used further down the pipeline.

2.4.2 Tag Data

Tagging and labeling data in machine learning involves assigning categories or attributes to individual data points, facilitating the algorithm's understanding and prediction of patterns within the data. This process is essential for training and supervising machine learning models, as it helps establish ground truth and enables the model to learn from labeled examples. Proper labeling ensures that the model can make accurate predictions and generalizations, and it underpins various applications like image recognition, natural language processing, and classification solutions. The quality and accuracy of the labels directly impact the model's performance, making meticulous data labeling crucial for the success and reliability of machine learning applications. In the context of Firestorm, the labeling process entails creating a time window corresponding to the duration of attack simulations. During this predefined window, hardware samples streamed in within this window are marked as "True." All Firestorm run contain RunInfo data which serves to timestamp the start and conclusion of the attack, defined as finishing full file system encryption.

2.4.3 Feature Engineering

Derivative features refer to new variables or characteristics derived from existing data. These features are created by applying mathematical operations or transformations to the original data, which can include differentiation, integration, scaling, or other functions. Derivative features often capture additional information or patterns that may not be evident in the raw data, enhancing the model's ability to recognize complex relationships and make more accurate predictions. Ironwood Cyber has created a Feature Engineering library used in training to turn the raw quantitative hardware sensor metrics read from the computer into generalized statistics that help with building a more general architecture model. Included in the library are the calculations of mean and deviations of values given a vector, or window, of hardware samples. These engineered features play a crucial role in improving the performance of machine learning models by providing them with a richer set of information to work with, enabling more robust and insightful analyses.

2.5 Model Training and Tuning

During the training stage, the data is also resampled to achieve a desired class distribution of positive and negative entries and runs through hyperparameter tuning on a variety of model architectures. The most effective architecture used are gradient boosted decision trees, where additional trees are trained to correct the mistakes of the exiting ensemble. Models that perform well on the validation set are then exported and released to Firethorn for client inference. Model training and tuning is a manual process done by the engineers, as different parameters and techniques used can impact the model.

2.5.1 Data Aggregation

Data is aggregated by reading in flattened parquet files within Azure's blob storage. Models can be trained on a day's worth of data or months' worth of data. All hardware samples are labeled with the ID of the client machine that it came from to help understand the variations in different hardware and architectures. The types of attacks that are run on the training data are randomized. Using the Firestorm parameters, an automation engine randomizes the values for different encryption settings, file sizes, behaviors, within a given range specified by the engineer. The same is done for Duststorm.

2.5.2 Addressing Class Imbalance

Handling class imbalance in machine learning is a vital preprocessing step aimed at addressing situations where the number of instances in different classes is significantly uneven. This imbalance can lead to biased model predictions, as the model may favor the majority class due to its prevalence in the dataset. To mitigate this issue, we have employed techniques such as oversampling the minority class, under sampling the majority class, generating synthetic data, or utilizing specialized algorithms like SMOTE (Synthetic Minority Over-sampling Technique). These methods help rebalance the class distribution, enabling the model to make more equitable and accurate predictions while preventing it from being dominated by the majority class, which is especially critical in applications like ransomware prediction.

2.5.3 Hyperparameter Tuning

In this work, hyperparameter tuning is conducted on the SparkML Gradient Boosted Decision Trees (GBDT) classifier via the Hyperopt library. Essential hyperparameters such as tree depth, bin count, iteration number, step size, and information gain are optimally configured to enhance the model's generalization and predictive performance. The hyperparameters are sourced from well-defined search spaces. Numerical hyperparameters such as tree depth, bin count, and iteration number are optimized within specific ranges using quantized uniform distributions. For capturing the essence of learning rate variability effectively, a log-uniform distribution is employed for the step size. Lastly, a continuous uniform distribution is utilized for tuning the minimal information gain required for a split. For more information on the parameters, please check Appendix [x]. For Firethorn, the identified hyperparameters that were best for classification included 10 evaluations or more utilizing a best param option on each iteration.

2.5.4 Deploying Model

Upon the successful development of an efficient model, the model is exported into ONNX format and subsequently archived within a GitLab model repository. The models are then subject to distribution onto individual machines for the purpose of conducting simulation testing. Following this phase, they are deployed to both staging and production environments. The subsequent diagram below visually represents the ONNX model, offering insight into the structure of the gradient-boosted decision trees. Each tree regressor amalgamates its outcomes, culminating in the calculation of probabilities and the assignment of labels within the context of binary classification.

3 Performance Results

The model's overall performance highlights the practicality and effectiveness of utilizing the side-channel method for anomaly detection. During the training and inference phase, the team achieved exceptional accuracy rates with minimal false positives when tailoring models to individual machines. From a scalability perspective, the potential to create dedicated models for individual machine promises superior accuracy, though it comes with increased costs. In contrast, the prowess of the generalized model lies in its adaptability and cost-effectiveness in deployment. These promising results underscore the innovative potential in the team's approach to quickly and accurately detect new and evasive malware. Ironwood's algorithms are constantly improving Firethorn's model to craft a resilient, architecture-agnostic endpoint model.

Once a model is deployed to clients, inference is run and analyzed on new attacks. Attack simulations that are run on client machines return a report that generates a confusion matrix among other metrics based on the current Firethorn model. This is used to further validate the model using live inference with Ironwood's Firestorm simulator.

A Windows machine with a model trained to its architecture can detect anomalous behavior in less than one second against multiple cipher strains and parameters. A detection time in the milliseconds highlights the effectiveness of the model's side-channel analysis in identifying threats independent of the standard techniques seen in the industry, such as observing signatures, known malware, etc. Five different ciphers were tested against a single architecture with insignificant variability in results. Because this technique can infer the technique at such a fast rate, Firethorn also allows mechanisms for inference by calculating alerts within a 1 second window. Though the latency will be lower, this can reduce the number of false positives by calculating a certain percentage of positive or negative alerts within a set window.

This section will present the outcomes and findings of the machine learning models generated, highlighting their performance on general architectures and models trained to fit on one specific architecture. These results are produced from simulations run via Firestorm and Duststorm, allowing for precise and high-resolution metrics on how well the model is performing within the specific experiment parameters.

3.1 Model Performance Evaluation/Results

This section dives into the evaluation of the machine learning models on both general and specific architecture datasets, emphasizing their strengths and weaknesses in each context. The performance of the models is analyzed on training and test datasets, with a focus on aspects such as overfitting, generalization, and any observed bias or variance. A comparison is made between the performance of models tailored to a specific architecture and those designed for widespread use, emphasizing distinctions and implications. While this section will be more focused on the objective results of the model and technical evaluation, the next section will highlight the successes identified.

In this study, the performance of Firethorn is evaluated across various performance metrics to assess its effectiveness in classifying ransomware. Different hyperparameters were used for the training of Gradient Boosted Decision Trees (GBDT) which performed the best out of the evaluated algorithms.

Models were validated using both the F1-Score metric and false positive rate. F1-Score is a comprehensive metric that balances precision and recall, calculated as the harmonic mean of the two:

$$\text{F1-Score} = \frac{2(P \cdot R)}{P + R}$$

Data patterns that exhibit diversity necessitate additional model engineering and training. Varied architectures introduce unique hardware, sensor types, and resource utilization during security incidents. Firethorn's solution involves constructing a sophisticated, all-encompassing model by harnessing insights from a wide array of machines, focusing on shared patterns, particularly within statistical feature deviations.

As anticipated, the model tailored for a specific architecture outperforms the generalized model when assessed on test data. The results will dive into the predictive performance of the endpoints. Additional information on the hardware is provided in Appendix A, affirming the confidence in the versatility of this approach.

3.2 General Architecture

Data from eight different machines within the Ironwood Cyber laboratory was aggregated to create a singular model for general architectures. The model, designed to encompass various machines, was tested on General Architecture A. Tuning followed a consistent approach, leveraging SparkML GBDT/SynapseML GBDT for binary classification, while multiple algorithms and libraries were explored for optimization.

Several generalized architecture models were developed and deployed on client machines for performance evaluation. Accuracy on the validation set ranged from 90% to 99%, with varying recall and precision values. These variations may be attributed to potential overfitting, discrepancies in hyperparameter tuning, variations in data quality during training, and the inherent diversity of machine learning algorithms used in model construction. However, there is a stable 99% accuracy that highlights the effectiveness of this approach, and when tested across different machines showcased very impressive results.

Confusion Matrix	TP: 2047678, TN: 735144, FP: 6928, FN: 4988
Accuracy	0.9971
Recall	0.9976
Precision	0.9966

Table 1: Generalized Architecture Training On Validation Data

Figure 3. below showcases the results from inference after running a simulation on General Architecture A (Appendix A).

TP:7995	FP:123
FN:510	TN:920

Figure 2: Confusion Matrix

The below graph showcase the performance on Architecture A running a generalized model.

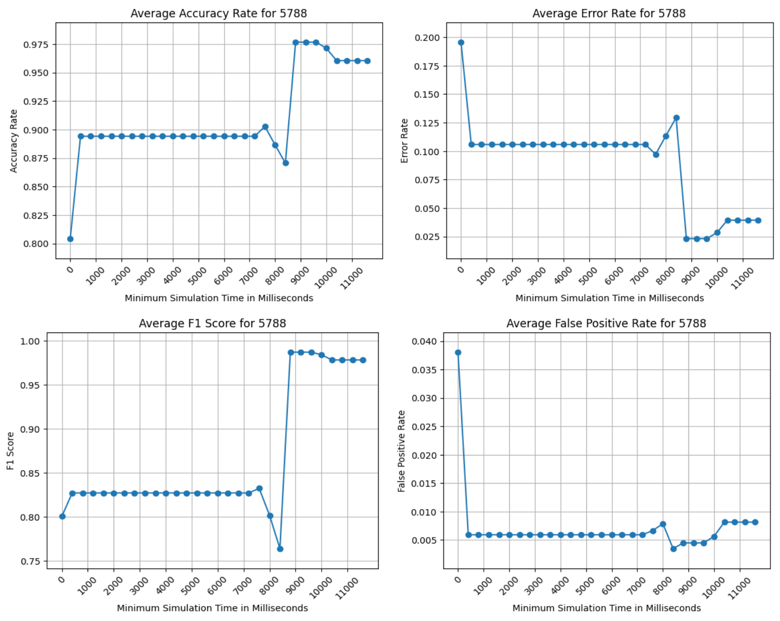


Figure 3: Inference results for General-Fit Model Architecture

For one simulation run, Architecture A exhibited a <1% false positive rate and an impressive accuracy rate of 98%. This performance can likely attributed to the model's emphasis on the unique patterns associated with one architecture over another, potentially leading to a bias favoring that specific architecture and/or performance of sims on that architecture. False positives have been observed to get down to less than 1%. For Architecture A there was still high accuracy even with a low false positive rate. For other architectures there were also extrememly low false positive rates (<1%) at the cost of model precision (and the increase of false negatives). Moreover, the variations in hardware specifications and the varying efficiency of different machines can result in some systems being inherently more capable of running ransomware simulations at an accelerated pace. Notably, simulation parameters also encompass different encryption modes such as Cipher Block Chaining (CBC) and Galois/Counter Mode (GCM) stream encryption as just a few used, which impact the resources of the architecture differently. Some architectures have shown to have a low false positive rate but also a substantial amounts of false negatives that show the need to build a generalized model across a wider range of architectures.

However, it's essential to recognize that these findings reveal promising opportunities for further enhancement. By refining the generalized model to bridge the performance gap across multiple architectures, a more balanced and accurate anomaly detection solution can be achieved. The technique is already proven to work for classification and this pursuit of optimization reinforces the commitment to providing robust security solutions that adapt to diverse architectural configurations.

The generalized model is an especially exciting opportunity as it will enable model deployment on machines without any kind of specialized training on the machine beforehand. With such promising early results for this technique at the current stage, it is likely with increased data collection measures a generalized model can perform at minimal false positive rates and high recall rate. With additional machines we can scale up data collection and quality as well as gain stronger insights into model performance with more testing opportunities available.

In addition, the models had an average detection time of less than a second. It is also noted that random noise was also generated during this time for training, so the model itself is not classifying on any known active or non-active/idle state. Within the parameters of running simulated attacks on these machines, the results show high confidence that side-channel analysis can be used to classify a machines current state.

3.2.1 General Architecture Performance on Enterprise Client Machines

When running Firethorn on client machines, a vast majority of machines had less than 1% false positive rates on alerts. It is noted that these machines also are not included in the training set for the model that was deployed. Some machines did have a high FP rate which is attributed to different architectures and potentially running Firethorn on virtual machines (from which there are no HW sensors). Below is a graph of the false positive rate from 55 machines running in production environments.

There are notable FP rates of above 2% on these machines which are attributed to different types of architectures running the model. A good amount of production machines however had a very low FP rate. However, there were some outlier machines that had a high >90% FP rate which were omitted from the graph. It is speculated that these machines are virtual machines that don't have a hardware sensor and therefore a limited amount of features used for inference. Further planned research is to incorporate the data from production machines (demonstrating actual real-world Enterprise usage by employees) into new training data used for training models.

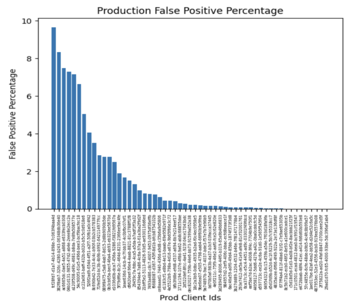


Figure 4: Predictions On Prod Machines

3.3 Single Fit Architecture

When trained only on samples from a specific architecture, a model was built that exhibited an impressive accuracy range of 96% to 97% when tested on a validation dataset. This architecture, trained with a similar hyperparameter tuning approach, used exclusively utilized data samples from a single machine, enabling a more tailored model fit to the unique architectural characteristics. To achieve a balanced class distribution, data resampling was performed, leading to a more focused dataset with approximately 8 million rows, reflecting the singular source of data generation. Subsequently, the model's performance on the same validation dataset excelled, yielding precision rates of up to 99.7% and recall rates as high as 96%.

A single fit architecture was trained to see performance for a single machine and may not be feasible for scalability across multiple machines as it requires individual training on specific machine data. However, single fit models are tailored for the architecture that it is trained on and can provide better results by reducing variability in architecture. Like the general model, models can be trained in under two hours dependent on the amount of data and hyperparameters used. Data regarding single fits are available upon request.

3.4 Feature Importance and Contribution

In the analysis of feature importance, the objective was to discern the variables and attributes exerting substantial influence on Firethorn's predictions. The evaluation unveiled that the metrics related to the disk exhibited significant importance in the model's decision-making process, as evident from the noteworthy importance score of 0.58 in the context of a single fit endpoint, as depicted in Figure 5. This underscores the pivotal role of disk-related metrics in the prediction of system encryption, aligning with the notion that the most influential features are those directly impacted by ransomware encryption.

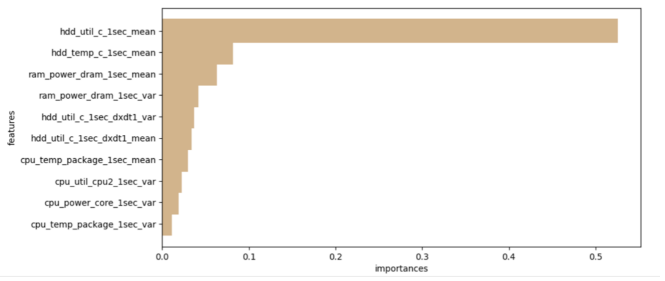


Figure 5: Larger model size with 68 features of importance on single fit endpoint

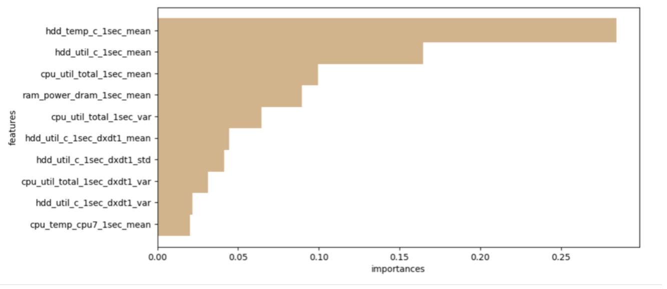


Figure 6: Generalized architecture fit with features trained from multiple endpoints

4 Performance Highlights

4.1 False Positive Rate Precision

In our extensive analysis of the results, it is evident that the model exhibited exceptional performance in classifying ransomware, particularly in terms of its FPR. The FPR, which measures the rate of false positives or erroneous detections, had runs that returned a less than 1% rate, showcasing the model's ability to make highly accurate predictions while maintaining a remarkably low rate of false alarms. This level of precision is of paramount importance, as it significantly reduces the chances of false alerts and unnecessary responses in practical applications. The model's consistent placement in the 90th percentile for accuracy further reinforces its efficacy and reliability. These outcomes serve as a testament to the model's capability and the promise it holds for enhancing security and anomaly detection systems.

4.2 Low Latency Data Monitoring

Hardware samples are collected at a rate of 100 milliseconds (ms), ensuring precision and leaving no room for data loss or latency issues. This data is then rapidly processed by the model, operating at a remarkable temporal efficiency of 0.1 seconds to provide probabilities on the state of the machine. Models continuously generate predictions based on a 1 second window of this data at a 10 Hz rate.

Low latency is crucial for maintaining data integrity and accelerating anomaly detection. Higher latency can lead to data loss, undermining the accuracy and timeliness of the anomaly detection system. The inconspicuous latency of this system enhances its agility in responding to evolving hardware states, making it exceptionally efficient for real-time monitoring and detection.

4.3 Mean Time to Detection

The model has provided an exceptional time to detection in detecting when under attack. Detection time is figured out by determining the first true positive prediction from the start of the simulation run. In one model that is on a machine that has not been included in the training data, an attack was detected within 300ms, or less than one third of a second, from when the attack first started. The performance on that specific attack was a 93% accuracy with a 98% precision.

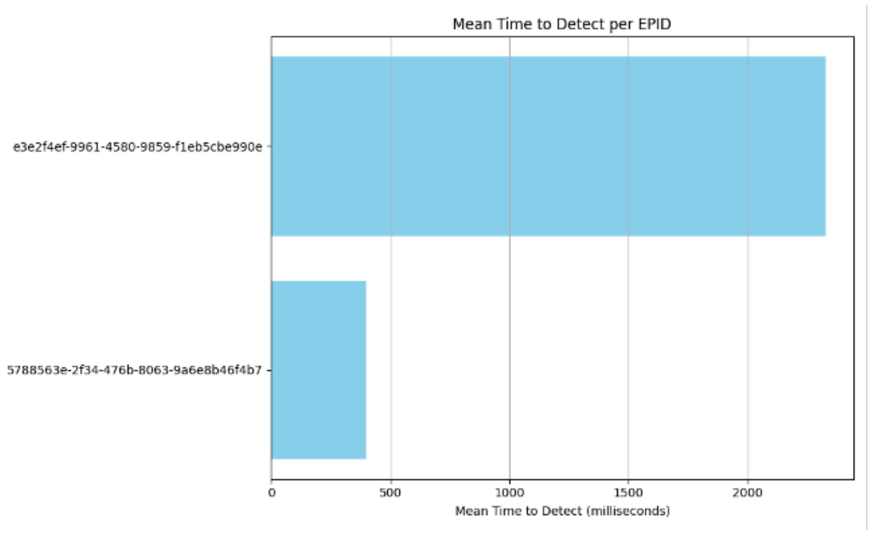


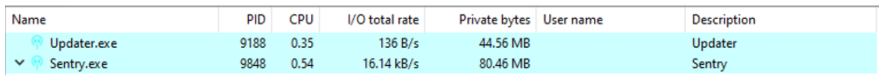
Figure 7: Mean time to detect from simulations

Due to the granularity of detection, a hierarchy of alerts can be built to minimize the number of false positives and increase accuracy. Inference is being done by taking a 1second window of samples, polling it continuously, and performing calculations to achieve features that are fed into the model. A current method that is being done is counting the number of model positives that appear in a certain set, and if a certain number is reached, alert on an attack. This inference can be built out further for techniques such as identifying the probability of attack in a window or increasing latency for accuracy.

4.4 CPU utilization, power consumption, memory utilization

The CPU utilization of Firethorn is exceptionally low. Embedded within the Ironwood Sentry agent, the Firethorn inference model operates with extraordinary finesse. When examining the spectrum of software executed on a client machine, it becomes evident that Firethorn exerts minimal demand, characterized by an average utilization rate of less than 1%.

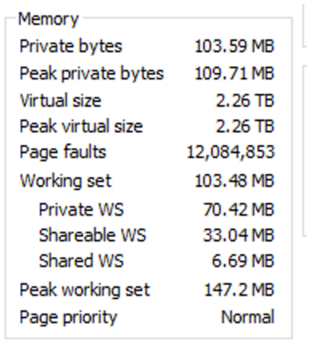
This seemingly modest CPU utilization carries profound implications for power consumption within a standard desktop environment. Firethorn's lightweight computational footprint and its discerning anomaly detection capabilities results in an efficient detection method. By expending minimal CPU resources, Firethorn leaves out the need for extensive power allocation, thereby ensuring that the system is not put under computational stress and can converse energy for user activity.



Name	PID	CPU	I/O total rate	Private bytes	User name	Description
Updater.exe	9188	0.35	136 B/s	44.56 MB		Updater
Sentry.exe	9848	0.54	16.14 kB/s	80.46 MB		Sentry

Figure 8: Process Hacker Screenshot showing Sentry CPU rate, (running Firethorn) Architecture A

Memory utilization emerges as another facet of Firethorn's exceptional resource efficiency. The Firethorn inference model exhibit an acceptable allocation of memory resources, ensuring that it operates with a remarkably low memory footprint. The lean memory utilization minimizes the system's overall memory overhead, preserving the broader memory pool for other critical applications and processes.



Memory	
Private bytes	103.59 MB
Peak private bytes	109.71 MB
Virtual size	2.26 TB
Peak virtual size	2.26 TB
Page faults	12,084,853
Working set	103.48 MB
Private WS	70.42 MB
Shareable WS	33.04 MB
Shared WS	6.69 MB
Peak working set	147.2 MB
Page priority	Normal

Figure 9: Memory usage of Firethorn for Architecture A

5 Opportunities and Future Research

In concluding this section, the team confidently leverage the insights gained from the Firethorn development to propel the models towards greater accuracy. Ironwood's journey toward achieving a consistent 98-99% assurance and precision rate under varying parameters revolves around optimizing data quality, setting the stage for future advancements. To this end, the following strategies are embraced to improve performance:

- Run attack simulations on customer machines running the client to add a larger variety of both authentic noise and hardware specifications.
- Reverse actual ransomware strains to incorporate into Firestorm + continue to add more malicious capabilities.
- Increase the amount of data used for training by adding more machines to the construct or extending simulation hours.
- Increase the number of features generated by the hardware sensors
- Increase and improve the infrastructure to handle substantial amounts of data for training.
- Building out reporting capabilities to get better visibility of real-time performance across all Firethorn clients.
- Building out reporting capabilities to get better UI visibility into model predictability for certain simulation runs.
- Building out reporting capabilities to get easier visibility into the feature derivatives that alert on malicious behavior.
- Build out different inference techniques to reduce false positives due to the high latency of data being inferred.

The proposed future research, areas for improvement, further exploration, and refinement of the models and methodologies employed, will help improve model performance considering the diverse architecture requirements of real-world applications.

6 Conclusion

The Firethorn model has made significant strides over the past year in ransomware detection through the innovative use of side-channel analysis. We have successfully established an infrastructure that enables us to capture and categorize metrics associated with ransomware and various malicious attacks effectively. By focusing on the consequential effects of malware, Firethorn has demonstrated a remarkable capability to uncover obfuscated and new malware swiftly, improving detection rates and system resilience. Utilizing intrinsic hardware sensors for analyzing malicious behaviors, our approach negates the need for direct filesystem access, promoting a non-intrusive yet highly effective detection strategy.

Our successes pave the way for further enhancements. We've identified opportunities to improve data collection and quality, aiming to tailor models to meet individual client needs, focusing on protecting critical assets vulnerable to ransomware attacks. By continuously refining our models and exploring new avenues, we are committed to increasing the accuracy and reducing false positives, ensuring that Firethorn remains at the forefront of technological advancements in ransomware detection.

7 Author Information

James Cao

James Cao is a Cyber Software Engineer Sr Staff at Ironwood Cyber and one of the architects for the Firethorn product line. Before Ironwood Cyber, he was the Senior Security Engineer for Lockheed Martin's Space Internal Cloud, and the Technical Lead for the Space Cyber Test Lab. James has extensive experience in multiple DoD programs including Next-Gen OPIR, FBM, and CPS where he performed threat modeling, software development, and Supply Chain cyber security. James has an undergraduate degree in computer science from Colorado State University and a master's in information management from the Harvard Extension School.

Donal Lowsley-Williams

Donal Lowsley-Williams is a Machine Learning Software Engineer at Ironwood Cyber co-architecting the Firethorn product line. Prior to joining Ironwood Cyber, Donal worked at a healthcare startup building medical billing error detection software. Donal graduated from Cornell Tech with a master's degree in computer science where he concentrated on Machine Learning and Cryptography. He completed his undergraduate degree at the Cornell University Hotel School, where he minored in Computer Science and taught Python and Statistics.

Ethan Puchaty

Ethan is the Chief Technology Officer for Ironwood Cyber, leading evolution and development of the Firethorn™ and Enlight™ product line. Prior to co-founding Ironwood Cyber with CEO Aaron Estes, Ethan was a Fellow at Lockheed Martin, most recently leading platform cyber architecture development and solutions across the Aeronautics and Space Systems business areas. He has led engineering efforts and developed novel cybersecurity technology solutions across air, space, ground, and weapon systems platforms during his 11-year tenure with Lockheed Martin. His experience includes the conception, research and development of cross-platform intrusion detection modules for embedded space systems including GPS III, SBIRS and Next-Gen OPIR.

Aaron Estes D.Eng

Dr. Estes is an expert software security analyst, security solution architect, engineer and professor who has worked with the nations top defense contractors, financial institutions, and electronics and entertainment conglomerates to assess information security risk and solve some of the most critical security concerns of todays' changing cyber world. Aaron has set his professional career apart by focusing on academic discipline as well as creative passion to create a level of skill based not only on technical skill but also in-depth knowledge of engineering discipline and computer science theory. Aaron has achieved a depth of emersion into the world of computer security rarely seen in this relatively young field. Aaron is a Lockheed Martin Fellow, the highest honor an engineer can receive, which he has earned three times with Lockheed Martin IS&GS, Lockheed Martin Space, and Lockheed Martin Aeronautics.

8 Appendix

Appendix A

Present Architectures

Environment	CPU	RAM	Storage	Graphics	Network
Architecture A - 5788	Intel(R) Core(TM) i7-8559U CPU @ 2.70GHz 2.71 GHz	16 GB	Samsung SSD 970 Evo Plus - 250 GB	Intel Iris Plus Graphics 655	Intel Ethernet Connection I219-V

Figure 10: System Specifications