Future Technologies Conference (FTC) 2017
29-30 November 2017| Vancouver, Canada

Sensor-based Ransomware Detection

Michael A. Taylor, Kaitlin N. Smith, and Mitchell A. Thornton
Darwin Deason Institute for Cybersecurity
Southern Methodist University
Dallas, TX 75275-0122
{taylorma, knsmith,mitch}@smu.edu

***Abstract*—A new method for detection of ransomware that is present in an infected host during its payload execution is proposed and evaluated. Data streams from on-board sensors present in modern computing systems are monitored and appropriate criteria are used that enable the sensor data to effectively detect the presence of ransomware infections. Encryption detection depends upon the use of small yet distinguishable changes in the physical state of a system as reported through on-board sensor readings. A feature vector is formulated consisting of various sensor output that is coupled with a detection criteria for the binary states of "ransomware present" versus "normal operation". Preliminary experimental results indicate that ransomware is detected with an overall accuracy in excess of 95% and with a corresponding false positive rates of less than 6% for four different types of encryption methods over two candidate systems with different operating systems. An advantage of this approach is that previously unknown or "zero-day" versions of ransomware are vulnerable to our detection method since no prior knowledge of the malware, such as a data signature, is required for our method to be deployed and used.**

***Keywords*—Ransomware detection; physical sensor side channel; feature vector; encryption**

I. INTRODUCTION

Malware is a term that we use here to refer to malicious software and is used to refer to all forms of software that can be used to compromise computer functions. This compromise causes harm to the victim computer and ultimately to the user or owner of the host computer. There are a large variety of types of malware including, viruses, worms, adware, bots, rootkits, spyware, trojans, and the primary subject of this investigation, ransomware. Ransomware is a form of malware that holds a victim computer system's files hostage while demanding a ransom to release access to those files back to their legitimate owner.

A typical ransomware attack scenario involves infection of victim computer through penetration of an attack vector whereby the malware resulting from the attack contains a payload that, unbeknownst to the victim, engages in rendering important files as unusable, through their encryption with a key that is unknown to the victim. Upon completion of the initial silent encryption phase, the original unencrypted files are deleted and the victim is alerted that their files are now inaccessible and will remain so until a ransom is paid. It is also often the case, that the attacker will demand ransom within some time period or otherwise the encryption key will be destroyed resulting in permanent loss of the victim's data. Fig. 1 contains a high-level diagram of the chain of events characterizing a typical ransomware attack from the point of view of the adversary.

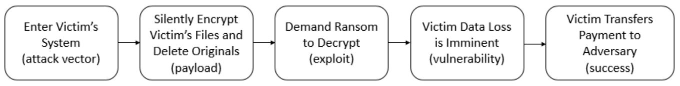
Enter Victim's System (attack vector)
Silently Encrypt Victim's Files and Delete Originals (payload)
Demand Ransom to Decrypt (exploit)
Victim Data Loss is Imminent (vulnerability)
Victim Transfers Payment to Adversary (success)

Fig. 1. Typical ransomware attack.

Ransomware attacks can occur through a variety of means. As a more specific example, a common attack vector is the use of email spear phishing where a victim receives an email message that somehow causes the victim to click on an embedded link to a webpage that, in turn, causes the victim's browser to display the adversaries' webpage and to lure the victim into downloading the malware. This malware could be presented as a macro contained within an Office® document or some other executable. Once the executable is run, it proceeds to encrypt the victim's local files in a silent mode and upon completion of the encryption, it then notifies the user that their files are now inaccessible due to the encryption. Next, the exploitation of the victim occurs through a demand of payment. The typical form of ransom payment is through an anonymous transfer of non-traceable funds through the darkweb using electronic currency such as bitcoin. The victim is promised that, if the ransom is received within a prescribed timeframe, the key will be delivered allowing their encrypted files to be decrypted. While this example scenario is based upon the premise of email phishing as the attack vector penetration method, other means for delivering the malware payload are also possible as well as other varieties of payload activity, victim exploitation, and vulnerability [10]. More detailed information about well-known versions of ransomware, including how they infect computers can be found in [1], [2] and [5].

Effective defense against a ransomware attack is generally considered to comprise a multi-tiered or layered approach [6]. Detection of the malware during the time it is being downloaded to the victim computer is the outer defense, and if possible, can prevent the ransomware from ever entering the system. This defense approach targets prevention of the attack vector from ever penetrating a victim's host computer. Packet signature monitoring via an intrusion detection system (IDS) or file signature monitoring via a local antivirus software program can provide this capability, but only if these methods are capable of recognizing the malware through knowledge of the data signatures. While this is a desirable defense, it is notoriously difficult to prevent infection with previously unknown ransomware versions, or so-called “zero-day” attacks. In the case of zero-day ransomware, data signatures and other corresponding characteristics are unknown by definition. Furthermore, the increasing presence of polymorphic malware is causing signature-based approaches to become less effective than they once were [7].

If malware penetration is not prevented and the malware manages to be downloaded to the victim machine, the next line of defense is to detect its presence and halt its operation before or at least during the initial stages of victim host file encryption. Recently, an approach has been developed that performs payload detection through monitoring the integrity of victim host file system [3]. This method provides several metrics and indicators that are used to detect the presence of data files that are in the process of being encrypted. One of these metrics is the use of information entropy calculations. Information entropy is a single-valued metric that indicates when a data set has less structure, determinism, and redundancy. The idea underlying the use of an entropy metric is that an encrypted file is one that closely resembles a file of random data. This is due to the side effect that encryption generally produces data that appear to be random in order to prevent unauthorized decryption through the exploitation of determinism or redundancy in the encrypted file. Therefore an increase in entropy for a given file indicates the high likelihood that the file is being encrypted [8].

We propose an alternative method for ransomware detection on an infected host system. Instead of monitoring file system attributes, we monitor victim host system behavior by taking advantage of the increasingly large number of onboard sensors. In this sense, our method uses a physical side channel approach where the victim’s files are not directly monitored, rather the behavior of the victim machine is monitored and onboard sensor provided data are used as side channel information that indicate when an encryption operation is occurring. This monitoring can be accomplished through a background process that is loaded at boot time and thus continuously monitors the system for suspicious behavior. Once this suspicious behavior is detected, the user can be alerted and the suspicious processes can be suspended. The central difference between our approach and other previous approaches is that we use secondary effects to detect the presence of malware rather than a direct effect, such as measuring increases in file entropy.

Another recent approach for malware detection involves using embedded hardware performance counters that are present in most modern CPU architectures [14], [15]. This approach uses machine learning to create detection models that monitor minor variations in malware execution characteristics. Our approach differs from the use of hardware performance counters in that we use data being supplied from the suite of embedded sensors that are also present in modern computing platforms rather than performance counter data. Furthermore, our approach is designed to specifically detect ransomware since ransomware uses encryption to enable the victim’s data files to be held hostage, and hence, allows them to be recoverable when a ransom is supplied in exchange for the decryption key. Our approach uses data sources that are secondary to malware execution patterns and it does not rely upon the presence of performance counters. By targeting a specific class of malware, namely ransomware using encryption in the payload, we can achieve high detection accuracy rates.

We propose that our sensor-based detection methodology be used to complement more traditional signature-based approaches that are intended to prevent attack vector penetration. In contrast to prevention of attack vector penetration, the technique described here is designed to detect the presence of ransomware when penetration has been achieved. Our side channel-based or sensor-based approach has an advantage in comparison to antivirus or IDS systems in that zero-day versions of ransomware can be detected since previously captured malware signatures are not required. Furthermore, it is not necessary to monitor individual files and calculate entropy or other metrics that must be continually re-computed and compared with one another as is the case in the solution provided in [3].

We have implemented an experimental prototype system based on sensor monitoring and have tested it through use of a variety of scenarios where simulated ransomware is undergoing the silent phase of encrypting victim files. To evaluate our method, we used four different encryption methods from the Python Cryptography Toolkit that have been reported to be commonly used by adversaries during the development of ransomware [9]. Our experiments have yielded high accuracy rates in excess of 95% percent with false positive and false negative rates of approximately 6% or less. We anticipate that with proper tuning of our method, these accuracy and error rates can be further improved.

II. Background Concepts Used in the Approach

A. *Physical Sensors*

Most modern computer systems comprise sensors that monitor the state of internal hardware components. These sensors continuously gather and supply information that is communicated with other devices and subsystems within the system for the intended purpose of ensuring that the system stays within specific operating specifications. If sensor data reveals that a system component is approaching a boundary for a recommended value of an operational specification, safety mechanisms will typically be engaged in order to correct the internal environment so that system malfunctions can be prevented. For example, when the data from a temperature sensor of a computer’s central processing unit, CPU, begins to increase, a signal is sent to the CPU cooling fan. This signal causes the fan to either become active or to increase the fan speed in order to cool the CPU. Additionally, there are sensors that provide input to other subsystems such as internal power management units, PMUs, to conserve power usage.

Typically, computer system components are designed to be compact in size through the use of transistors with feature sizing in the nanometer scale. As a direct result, whenever computations become more complex, more stress in placed on a computer’s hardware components. This increased stress occurs because a large number of transistors are simultaneously switching in a circuit that correspondingly cause an increase in dynamic power consumption resulting in more heat dissipation during heavy computational activity. Thus, monitoring the side channels of a system with embedded sensors that measure temperature, power consumption, and battery voltage levels can give insight into the type of processing that is underway on a computer at a given time. With this thought in mind, we hypothesize that monitoring a computer's side channels through periodic observations of sensor output data could also indicate when a resource-heavy task, such as encryption, is occurring. Since ransomware utilizes encryption in its payload to deny its victims access to their files, analyzing data from a computer's side channel sensor data could allow trends to emerge in regard to how a computer behaves while under ransomware attack.

A significant advantage of this approach as compared to other side channel methods is that the sensors and a means for querying them are natively provided. Thus there are fewer concerns in deploying and accessing sensors for the purpose of side channel exploitation. Furthermore, the trend has been that an increasingly diverse number of sensors are provided as integral components in modern computing devices. A typical smart phone has many embedded sensors that could be used to support security applications including power monitors, accelerometers, ambient light sensors, antennas (including GPS receivers), fingerprint scanners, barometers, cameras, touchpad pressure sensors, and others. Even rack-mounted industrial servers contain a significant number of sensors that measure subsystem power consumption, temperature, and other environmental factors. All of these deployed sensors in modern computing devices provide a rich set of data sources that may be used to provide internal side-channel information for the environment in which a computing device is operating. Use of these sensors has been used in other security-related applications in the past. As an example, in [13], sensors present in mobile computing devices have been used to provide a user demographic classification capability for mobile devices with embedded touchscreens.

Conventional computers are comprised of the same set of basic internal devices to enable their operation. However, manufacturers may choose to use different and unique sets of components for their various computer models. Due to this variation among different product models, corresponding differences among the readings of the internal onboard sensors can occur when they are queried. Instantaneous values of sensor readings can be accessed via the command line or through calls to the operating system using an application that queries and interprets the onboard sensor data. During our experimentation, the Hardware Monitor and the Open Hardware Monitor applications were used to provide information from systems running Apple's OSX® and Microsoft's Windows® operating systems. As an example of large number of available on-board sensors, a list of the 59 sensors and their readings from an Apple Macbook® is provided in Table 1.

The on-board sensors in Table 1 for the Apple Macbook are provided as an example. Windows®-based machines also have a similar complement of accessible on-board sensors. In this investigation, Python polling scripts are used to continuously access data from the hardware sensors and to record the information in the form of `.csv` file for further analysis. Operational deployment of the method would likely use a more sophisticated technique for obtaining sensor data such as an interrupt-driven background process or an event-driven polling technique.

B. *Machine Learning Concepts*

In the investigation reported here, prediction models were created using Machine Learning (ML) techniques. Models are trained using a large amount of data gathered and processed from an experimental environment. We hypothesized that the sensor data, such as that provided in Table 1, can be used to form a feature vector that differentiates between the binary machine states of "normal operation" versus "ransomware payload execution" (*i.e.*, unauthorized encryption activity). The model is trained to weigh the feature vector components with a goal of predicting the machine state with high accuracy. For the implementation described here, we used a simple logistic regression approach as the ML classification algorithm wherein our goal is to discriminate between the binary states of "normal operation" versus "ransomware payload execution." We note that many alternative classification algorithms are available and subsequent research efforts will focus upon choosing the best form of classification methodology.

Logistic regression is a statistical method that is used to create models that determine if an instance belongs to a certain category. More detailed information about logistic regression can be found in [4]. When this method is used as a binary classifier for ML, a probability estimate produced by logistic regression is compared to a threshold value. This comparison results in a predicted state. When the prediction is false, a system state of "normal operation" is declared, otherwise the predicted state is "ransomware payload execution." In order to create a logistic regression model that can provide accurate predictions, the model must be formulated with a training dataset that is as accurate and complete as possible. The example below shows how a simple logistic regression algorithm can be implemented within Python.

```
From sklearn.linear_model import LogisticRegressionCV

lr_sk = LogisticRegressionCV()
lr_sk.fit(X_train,y_train)
yhat = lr_sk.predict(X_test)
```

In this example, `lr_sk` is an object of class `LogisticRegression`. This object will eventually be used as the classifier model. However, before the model is effective, it must be trained using the fit method. `lr_sk` is trained using `X_train` and `y_train`. `X_train` utilizes a matrix containing the training data while `y_train` holds a vector of values that correspond to `X_train` for the categorical feature that is to be predicted. Once the model has been trained, the prediction method is used to classify new data present in `X_test`.

C. *Ransomware Payload Model*

The encryption algorithm used within an instance of a ransomware implementation encodes files in such a way that it is virtually impossible to recover the data without knowledge of the decryption key. In many cases, such as in the malware implementations known as **CryptoLocker** and **CryptoWall**, the encryption algorithms adhere to the Advanced Encryption Standard (AES) [1], [5]. AES is currently one of the encryption methods of choice for the United States government due to the algorithm's well-known ability to protect sensitive data [11].

To evaluate the robustness of our methodology, four different methods of encryption are used in our experiments and are encapsulated within scripts written in Python, a *C*-based programming language, in order to simulate the execution of a ransomware payload. Specifically, we used Electronic Code Book, Cipher-Block Chaining, Cipher FeedBack, and XOR encryption. The Python Cryptography Toolkit implementations are used to implement these four encryption algorithms within our simulated ransomware scripts. Electronic Code Book, Cipher-Block Chaining, and Cipher Feedback are symmetric-key algorithms that follow the AES standard. XOR encryption is included in our study due to the method's ability to encode files quickly with a corresponding relatively low complexity. Due to the unavailability of a cyber range during the test of our ransomware payload detection prototype, we refrained from using actual samples of malware, such as those described in [5], that are currently a threat to all Internet-connected devices.

Electronic Code Book encryption is based on the concept of a block cipher in which blocks of information are encrypted rather than each bit being encrypted individually as in a stream cipher. Additionally, each encoded block of information is independent of any other block [12]. Because of this independence, Electronic Code Book encryption is characterized by the fact that matching blocks of information will always be identical, even in their encrypted form. Cipher-Block Chaining is another type of block cipher, but it's encryption method differs as compared to the Electronic Code Book method. With Cipher-Block Chaining, each encoded block of data depends on the previously encoded information [12]. Due to this dependence, matching blocks of plaintext information will result in different encryption values. Cipher Feedback encryption is a type of stream cipher in which small groups of bits are encrypted individually rather than large blocks of plaintext being processed as individual symbols [12]. Cipher Feedback encryption is based upon each new stream of data being encoded with a dependence on the previously encrypted data thus allowing matching plaintext data to have unique values in their plaintext form. XOR encryption is a simpler and far less secure method of encryption where information in plaintext form is simply XORed with an encryption key, thus, it is a basic substitution cipher. Although we recognize that XOR encryption could be easily defeated through cryptanalysis, it is included in our experiments since we wanted to evaluate our detection method with an extremely lightweight method and we felt that the XOR method was at least representative of the behavior of many lightweight methods. We anticipate that future versions of ransomware, particularly those that target limited resource systems such as IoT devices, may likely employ such lightweight encryption methods due to the practical constraint that a limited amount of processing power is available within this class of devices.

Our script that emulates the ransomware encryption payload includes the four encryption methods mentioned previously as well as options to set delays and random intervals for activity. To ensure that no data was actually lost during experimentation, the keys for encryption were hardcoded in the encryption script and were written as plaintext metadata in each encrypted file. The reach of the script was limited to a set of dummy test directories. Three dummy test sets were used: a small directory measuring 16 MB in size, a medium directory with 334 MB of files, and a large directory that was 3 GB in size. To collect training data, the encryption script randomized the encryption processes to execute at varying intervals of time so that a total of 40 encryptions would take place within the small directory. 20 encryptions would take place within the medium directory, and four of these would take place within the large directory. All of the encryption iterations taking place in each directory were evenly divided among the four different encryption techniques. During the duration of the script's run, hardware sensor data was gathered on average at every half second in order to obtain examples of the status of the hardware components during normal computer activity and during a covert encryption process.

Once the test data was populated within the machine learning algorithms to train the ransomware detection model, the encryption script was randomly activated in the background of the test computer in order to test the effectiveness of the model's ability to detect hidden encryption activity based on hardware sensor readings. In the experiments reported here, the amount of time required for training was proportional to the speed of the processor being used and ranged between 1.8 and 3.5 hours. The operational phase of the ransomware detection algorithm ran as a background process thus allowing normal usage of the victim computing system.

III. Method for Detecting Ransomware

Encryption detection through hardware sensor monitoring depends upon the detection of small yet distinguishable changes in the physical state of a system as reported through on-board sensor readings. Rather than focus on any one aspect of victim host machine performance, a feature vector of system sensor data is used to classify the system states. The relationship between increasing and decreasing sensor readings gives a strong indication of what type of task a system is currently performing given that enough data has been collected to form a clear indication of what a task looks like. In the case of encryption, the following sequence of events occurs: 1) data is read/accessed from the filesystem into main memory; 2) the accessed data is encrypted in main memory; and 3) the encrypted data is then written back to secondary storage. The system sensors, when individually monitored, would likely only allow very broad conclusions to be drawn such as indications of when the system has increased CPU usage or disk I/O operations are occuring. However, combining all the system sensors into a feature vector allows much more focused conclusions to be drawn that are indicative of sequences of events such as those outlined for an executing encryption process. Given that the sensor-data feature vector indicates a state of encryption, appropriate alerts can be issued and the corresponding encryption processes can be suspended or halted.

While a large number of sensors are available as evident in Table 1, we anticipated and subsequently observed that some of the sensors dominated in terms of provision of useful classification data. In particular the sensors that indicate main memory power usage emerged as being particularly sensitive to the presence of encryption activity. This is likely due to the fact that encryption algorithms are very memory-intensive operations.

While the dominant sensors were identified empirically in this study, in the future we intend to employ a more rigorous assessment and augment our ML approach based upon preprocessing analyses to refine our feature vector definition. Techniques such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and others will be employed to further refine the feature vector. We anticipate that such refinements will increase accuracy through the provision of enhanced discrimination in the classification algorithm thus reducing the false negative and false positive rates as well as reducing feature vector dimensionality. Another desired outcome of reducing the feature vector dimensionality is enhancement of performance of the detection algorithm. We intend to focus upon enhancements that results in decrease of false negative errors rather than false positive errors since the former error type results in more serious consequences than those of the latter type.

TABLE I. APPLE MACBOOK INTERNAL SENSORS AND READINGS

Apple Macbook Sensor	**Value**
SMART Disk APPLE SSD SD0128F (135251405113) [TEMPERATURE]:	136.4 F
SMC AIR INLET [TEMPERATURE]:	102.2 F
SMC BATTERY [TEMPERATURE]:	87.8 F
SMC BATTERY CHARGER PROXIMITY [TEMPERATURE]:	111.2 F
SMC BATTERY POSITION 2 [TEMPERATURE]:	87.8 F
SMC BATTERY POSITION 3 [TEMPERATURE]:	87.8 F
SMC CAMERA PROXIMITY [TEMPERATURE]:	113 F
SMC CHARGER PROXIMITY TEMPERATURE [TEMPERATURE]:	100.4 F
SMC CPU A PROXIMITY [TEMPERATURE]:	120.2 F
SMC LEFT PALM REST [TEMPERATURE]:	87.8 F
SMC MAIN HEAT SINK 2 [TEMPERATURE]:	93.2 F
SMC MAIN LOGIC BOARD [TEMPERATURE]:	96.8 F
SMC PLATFORM CONTROLLER HUB CHIP TEMPERATURE [TEMPERATURE]:	129.2 F
SMC SSD BAY [TEMPERATURE]:	98.6 F
SMC SSD TEMPERATURE A [TEMPERATURE]:	138.2 F
SMC SSD TEMPERATURE B [TEMPERATURE]:	120.2 F
SMC WLAN CARD [TEMPERATURE]:	98.6 F
Smart Battery bq20z451 (1) [TEMPERATURE]:	82.4 F
Battery 1 Cell 1 [VOLTAGE]:	3.69299 V
Battery 1 Cell 2 [VOLTAGE]:	3.69398 V
Battery 1 Voltage [VOLTAGE]:	7.38699 V
SMC CPU CORE [VOLTAGE]:	1.66211 V
SMC CPU SUPPLY 1 [VOLTAGE]:	1.05176 V
SMC DC INPUT [VOLTAGE]:	0 V
SMC POWER SUPPLY/BATTERY [VOLTAGE]:	7.16016 V
SMC SSD SUPPLY [VOLTAGE]:	3.29883 V
SMC WLAN CARD [VOLTAGE]:	3.29883 V
Battery 1 Current [CURRENT]:	1.45599 A
SMC 5V S0 LINE [CURRENT]:	0.0498047 A
SMC BACKLIGHT [CURRENT]:	0.00292969 A
SMC MAIN HEAT SINK 2 [TEMPERATURE]:	93.2 F
SMC MAIN LOGIC BOARD [TEMPERATURE]:	96.8 F
SMC PLATFORM CONTROLLER HUB CHIP TEMPERATURE [TEMPERATURE]:	129.2 F
SMC SSD BAY [TEMPERATURE]:	98.6 F
SMC SSD TEMPERATURE A [TEMPERATURE]:	138.2 F
SMC SSD TEMPERATURE B [TEMPERATURE]:	120.2 F
SMC WLAN CARD [TEMPERATURE]:	98.6 F
Smart Battery bq20z451 (1) [TEMPERATURE]:	82.4 F
Battery 1 Cell 1 [VOLTAGE]:	3.69299 V
Battery 1 Cell 2 [VOLTAGE]:	3.69398 V
Battery 1 Voltage [VOLTAGE]:	7.38699 V
SMC CPU CORE [VOLTAGE]:	1.66211 V
SMC CPU SUPPLY 1 [VOLTAGE]:	1.05176 V
SMC DC INPUT [VOLTAGE]:	0 V
SMC POWER SUPPLY/BATTERY [VOLTAGE]:	7.16016 V
SMC SSD SUPPLY [VOLTAGE]:	3.29883 V
SMC WLAN CARD [VOLTAGE]:	3.29883 V
Battery 1 Current [CURRENT]:	1.45599 A
SMC 5V S0 LINE [CURRENT]:	0.0498047 A
SMC BACKLIGHT [CURRENT]:	0.00292969 A
SMC BATTERY CURRENT [CURRENT]:	0.78125 A
SMC CPU CORE [CURRENT]:	0.566406 A
SMC CPU HIGH SIDE [CURRENT]:	0.241211 A
SMC CPU SUPPLY 1 [CURRENT]:	0.0107422 A
SMC CPU/VRM SUPPLY 2 [CURRENT]:	0 A
SMC DC INPUT [CURRENT]:	0.00195312 A
SMC DDR3 MEMORY 1.35V LINE [CURRENT]:	0.881836 A
SMC DDR3 MEMORY S3 LINE [CURRENT]:	0.0771484 A
SMC DISCRETE BATTERY [CURRENT]:	0.738281 A
SMC LCD PANEL [CURRENT]:	0.000976562 A
SMC POWER SUPPLY/BATTERY [CURRENT]:	0.770508 A
SMC SSD SUPPLY [CURRENT]:	0.0771484 A
SMC WLAN CARD [CURRENT]:	0.0107422 A
SMC 5V S0 LINE [POWER]:	0.164062 W
SMC BACKLIGHT [POWER]:	0.015625 W
SMC CPU CORE [POWER]:	0.964844 W
SMC CPU HIGH SIDE [POWER]:	1.72266 W
SMC CPU SUPPLY 1 [POWER]:	0.0078125 W
SMC CPU/VRM SUPPLY 2 [POWER]:	0 W
SMC DDR3 MEMORY 1.35V LINE [POWER]:	1.05469 W
SMC DDR3 MEMORY S3 LINE [POWER]:	0.0898438 W
SMC LCD PANEL [POWER]:	0 W
SMC POWER SUPPLY/BATTERY [POWER]:	5.51172 W
SMC SSD SUPPLY [POWER]:	0.25 W
SMC WLAN CARD [POWER]:	0.0351562 W
Battery 1 Current Capacity [CAPACITY]:	503 mAh
Battery 1 Total Capacity [CAPACITY]:	6559 mAh
SMC FAN Exhaust [RPMS]:	1192 RPM
SMC AMBIENT LIGHT 1 [LIGHT]:	70

All algorithms in the ransomware detection experiments were written in Python. To develop a model that detects ransomware, a training set of hardware sensor data is first be gathered. This data must include examples of how the sensors behave on the host computer under normal operating conditions as well as whenever a covert encryption process is taking place. After the sensor training data has been gathered, logistic regression is used to fit the model to the training data. Due to the slight variation between the components of each computer, the resulting ransomware detection model is unique for each machine. This model is then used to classify the state of the computer whenever the hardware sensors are routinely polled. If the model predicts that a suspicious encryption process is taking place on a computer, the user is alerted and the suspicious process is either suspended or terminated.

The detection algorithm is employed after model training has completed and it runs as a background process to allow normal usage of the system. A pseudocode version of the detection algorithm is provided below:

```
// load model from binary file
model = load('./model.pkl')
attack_count = 0
previous_prediction = 0
under_attack = False
// check sensor data and make prediction
while True
     data = monitor.read_sensors()
     prediction = model.predict(data)
     //  determine  action  based  on  current  and
     //  previous data
     if prediction:
             attack_count += 1
     else:
             if previous_prediction == 0:
                     attack_count = 0
                     under_attack = False
     previous_prediction = data
     // set condition to under attack if positive
     predictions
     // increase above threshold
     if attack_count > threshold:
             under_attack = True
```

IV. Exerimental Results

Testing was conducted on two machines, one running Apple OSX® and the other running Microsoft Windows®. Specifically, the Apple OSX machine is a Macbook Air with a 1.3GHz Intel® i5 processor and 4GB of main memory and the Windows® machine comprised an Intel® i7 processor with 32GB of main memory.

Training data was collected on both machines and the data was used to generate a prediction model. The new encryption detection method was tested utilizing a ransomware simulation testing script written in Python. The size of the directory and the method of encryption were selected by randomly picking a number between 1 and 100. All values of 60 and below caused encryption of the small directory, all values from 61 to 90 encrypted the medium directory, and all values from 91 to 100 encrypted the large directory. The particular encryption method used is randomly selected among the four types we implemented in our experiments.

After a particular directory has been encrypted, the script waits a random amount of time before performing additional encryption. The amount of time it waits is proportional to the size of the directory it previously encrypted. After encrypting a small directory, a random amount of time between 1 and 60 seconds is selected, a time between 5 and 10 minutes is selected for the medium directory, and a time between 15 and 30 minutes is selected for the large directory. The script also randomly selects a value between 5 and 15 and waits for an hour and a half after encrypting that many gigabytes of data. Randomness and wait times are utilized in order to simulate the attempts made by an adversary to avoid detection of ransomware payload execution. During the encryption process, the script searches for files by recursively starting from a given path. Files that have extensions matching a list of common user file types are read and their data is encrypted. After encryption the data is copied over the existing data in the original file.

After testing the Windows® machine for 5 hours 94.2% of sensor polls were accurately predicted as either "under attack" or "no attack". The confusion matrix in Fig. 2 shows the relationship between the predictions made by the model and the actual state of the machine. During the periods the script was performing encryption 98.1% of polling predictions correctly identified a state of under attack. During the periods the script was not performing encryption 92.5% of polling predictions correctly identified a state of no attack.

1.9% of the checks that occurred during periods of encryption incorrectly predicted that there was no attack (*i.e.*, a false negative error) while 7.5% of periods with no encryption incorrectly predicted that there was an attack (*i.e.*, a false positive error). Our classification method was tuned in a conservative fashion to focus more upon the reduction of false negative errors than the case of false positives as the former error type is assumed to be more critical than the latter.

The overall accuracy of the encryption detection method is illustrated in Fig. 3. The uppermost graph, (a), of the figure represents the actual periods of encryption or "truth data" while the plot on the bottom, (b), represents the actual predicted periods of encryption. These graphs depict the machine state on the vertical axis with zero indicating normal operation and one indicating under attack. The horizontal axes depict time.

The Apple machine was tested by only encrypting the large directory after a random wait period between 30 and 60 minutes over a 6-hour period. This method gives a clear indication of how well the new detection method can detect periods of high volume encryption. The confusion matrix in Fig. 4 shows the relationship between the predictions made by the model and the actual state of the machine.

After testing the Apple machine, 98.2% of the sensor polls resulted in accurate predictions. During the periods the script was actually performing an encryption operation, 99.7% of the polling predictions correctly identified a state of "under attack." During the periods the script was not performing encryption, 97.7% of polling predictions correctly identified a state of "no attack." A false negative rate of 0.27% of the checks that occurred during periods of encryption incorrectly predicted that there was no attack while a false positive error rate of 2.3% of observations with no encryption incorrectly predicted that there was an attack. Fig. 5 shows the periods of actual encryption in the uppermost portion, (a), and periods of predicted encryption in the lower portion, (b). As in Fig. 3, the vertical axes depict machine state and the horizontal axes depict time.

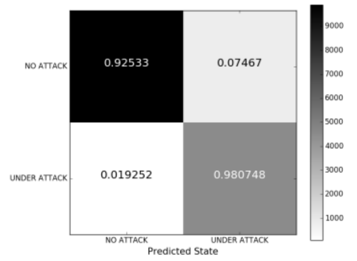

Fig. 2. Confusion matrix representing actual machine state vs. ransomware detection model prediction for a Windows machine.

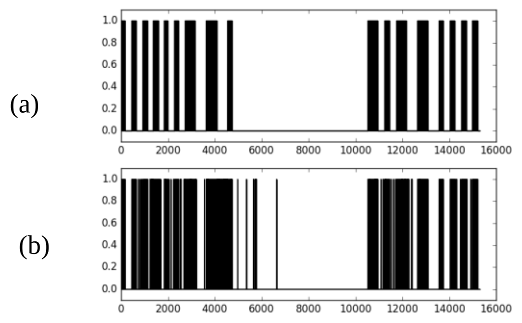

Fig. 3. (a) Plot of encryption activity vs. time and (b) Plot of ransomware detection model prediction vs. time for a Windows machine.

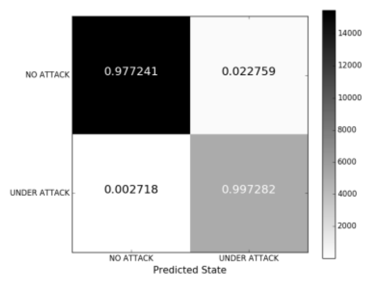

Fig. 4. Confusion matrix representing actual machine state vs. ransomware detection model prediction for an Apple machine.

Upon further analysis of the results we found that most periods of false positive predictions occured directly after a correct attack prediction. This can be observed in Fig. 5 which contains false positive periods after the second and fourth encryption periods. We believe that implementing additional testing and filtering techniques that more closely scrutinize predictions being made for a short period directly following a positive prediction period will result in increased overall accuracy. Currently our method does not make use of any temporal or history data regarding past recent predictions. We also believe that significant improvements can be made to the method when such temporal data is included in our prediction.

It is noted that there are cases when encryption is being legitimately conducted on a host system. It was indeed the case that our preliminary experimental results also detected this situation. In one case, we computed a false positive detection when the system performed a routine incremental backup operation. In operational deployment of our method, these cases will be accounted for through use of white-listing or other methods that notify the detection process that legitimate encryption operations are in process. Such notification methods will have to be carefully implemented in a manner where they cannot be exploited by malware to prevent exploitation. System registry data could be used to label processes that employ legitimate encryption and the ransomware detection process can be augmented to verify if a detection is the result of a legitimate process or not before a state of “ransomware payload execution” is declared.

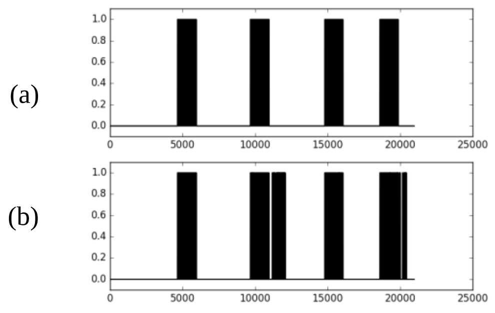

Fig. 5. (a) Plot of encryption activity vs. time and (b) Plot of ransomware detection model prediction vs. time for an Apple machine.

Our experiemental ransomware detection algorithm used a simple polling or sampling method wherein the operational phase of the detection method would periodically query the sensors to obtain readings. This approach suffers from potential aliasing problems, particularly if the malware payload were to be implemented in short bursts or use some other form of intelligence about the state of the victim system before encryption is executed. In the future, we intend to investigate the use of an alternative means to schedule sensor queries such as an event-based technique. This enhancement should decrease error rates while also reducing the avergae computational overhead since it is assumed that ransomware payload execution is a relatively rare event.

Because these preliminary experiments were conducted under the framework of a supervised machine learning model, the choice of the training data is a crucial aspect of the method. It will be important to craft a learning phase that is capable of charaterizing ransomware payload behavior even when the actual ransomware may be in the form of a zero-day exploit. Fortunately, most known ransomware uses well-known encryption methods, thus the training phase can focus on detection of these types of encryption.

V. Conclusion

A new method for the detection of ransomware is devised, implemented, and experimentally verified for accuracy and error. The method is applicable to both previously known as well as zero-day instances of ransomware that employ encryption in the payload. The detection method results in very low, if any, data loss since encryption detection can occur very early in the timespan of the malicious encryption activity. The technique is based upon monitoring on-board, hardware sensor data streams rather than characteristics of the targeted data. The new technique requires a minimal amount of modification to hosting computer systems since it uses pre-existing physical sensors, supporting circuitry, and system software assets that provide access to the sensor readings. Since the approach does not require direct interaction with the filesystem, it does not require extensive changes to the filesystem nor frequent disk accesses. The ransomware detection technique has been experimentally shown to be effective for both Apple OSX® and Microsoft Windows® operating systems.

Future research includes devising new methods for increasing the sensitivity of the classifier while also reducing error rates. One such method is the implementation of additional system features into the feature vector. We believe that gathering system performance data in conjunction with system sensor data could greatly improve the new method's ability to recognize periods of encryption. Additionally, we plan to analyze and reduce the dimensionality of the feature vector such that only the most discriminating data sources are included as features. We also plan to implement and evaluate a multiple model majority vote ensemble method for prediction based on the feature vector. Our future plans also include the evaluation of alternative machine learning methods such as the use of classifiers other than a simple linear regression binary classifier perhaps including a multi-state classifier that could include a state of "legitimate encryption." Finally, we plan to evaluate the sensor-based ransomware detection approach in an environment such as a cyber range where actual samples of ransomware are used.

References

[1] L. Abrams. (2013 Oct. 14). *CryptoLocker Ransomware Information Guide and FAQ* [Online]. Available: http://www.bleepingcomputer.com/virus-removal/cryptolocker-ransomware-information

[2] G. O'Gorman, G. McDonald. "Ransomware: A growing menace." Technical Report, Symantec Corporation, 2012.

[3] N. Scaife, H. Carter, P. Traynor, and K.B. Butler, "Cryptolock (and Drop It): Stopping Ransomware Attacks on User Data," in proc. *IEEE Int. Conf. on Dist. Computing Systems*, pp. 303-312, June 2016

[4] D.W. Hosmer, S. Lemeshow, R. X. Sturdivant. "Introduction to the Logistic Regression Model," in Wiley Series in Probability and Statistics: Applied Logistic Regression, 3rd ed. Wiley, 2013.

[5] J. Wyke, A. Ajjan. "The Current State of Ransomware." Technical Report, SophosLabs, 2015.

[6] S. Mehmood. "Enterprise Survival Guide for Ransomware Attacks." Technical Report, The SANS Institute. 2015.

[7] P. Mell, K. Kent, and J. Nusbaum, "Guide of Malware Incident Prevention and Handling," Recommendations of the National Institute of Standards and Technology (NIST), Special Publication 800-83, 2005.

[8] C.S. Oejman, P.J. Bruillard, B.D. Matzke, A.R. Phillips, K.T. Star, J.L. Jenson, D. Nordwall, S. Thompson, and E.S. Peterson. "LINEBACKER: LINE-speed Bio-inspired Analysis and Characterization for Event Recognition," in proc. *IEEE Security and Privacy Workshops*, pp. 88-95, 2016.

[9] D. Bonderud. (2016 Oct. 17). *CryPy Ransomware Slithers Onto PCs With Unique, Python-Based Encryption* [Online]. Available: https://securityintelligence.com/news/crypy-ransomware-slithers-onto-pcs-unique-python-based-encryption/

[10] J. Seitz. *Black Hat Python.* No Starch Press, Inc. 2015.

[11] "Announcing the Advanced Encryption Standard (AES)," *Federal Information Processing Standards Publication 197.* United States National Institute of Standards and Technology (NIST). 2001

[12] D. Litzenberger. *Package Crypto Python Cryptography Toolkit* [Online]. Available: https://www.dlitz.net/software/pycrypto/api/current/.

[13] A. Alharbi and M.A. Thornton, "Demographic Group Classification of Smart Device Users," in proc. *IEEE Int. Conf. on Machine Learning and Applications*, pp. 481-486, Dec. 2015.

[14] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, S. Stolfo. "On the Feasibility of Online Malware Detection with Performance Counters," in proc. *40th Annual Int. Symposium on Comp. Arch,* pp. 559-570, June 2013.

[15] A. Tang, S. Sethumadhavan, S. Stolfo. "Unsupervised Anomaly-based Malware Detection using Hardware Features," in proc. *Int. Symposium on Research in Attacks, Intrusions, and Defenses,* pp. 109-129. Sept. 2014.