

# CS 5472 - Advanced Topics in Computer Security

## Topic 7: Ransomware (2)

Spring 2021 Semester

Instructor: Bo Chen

[bchen@mtu.edu](mailto:bchen@mtu.edu)

<https://cs.mtu.edu/~bchen>

<https://snp.cs.mtu.edu>

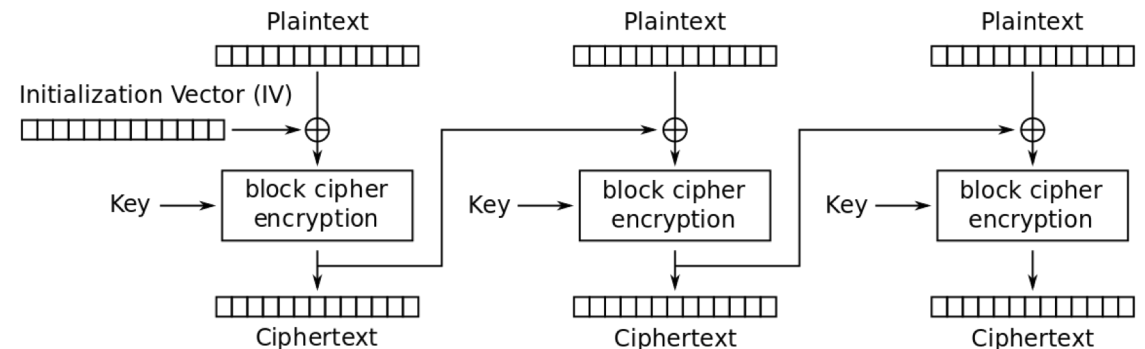
# Crypto Ransomware

- Encrypt the data, and ask for ransom
- Defenses:
  - Detection: need to detect ransomware as soon as possible to prevent ransomware from corrupting more data (**UNVEIL introduced on Tuesday**)
    - What if before the ransomware is detected, some of the data have been encrypted by the ransomware
  - A better defense: detection + recovery (**today**)



# A Little More on Detecting Ransomware

- File system access activities
  - File system activities without ransomware are different from that with ransomware
- Cryptographic primitives
  - Block ciphers for encryption



Cipher Block Chaining (CBC) mode encryption

# What about Data Recovery?

- Back up data **online**, like using public cloud services (iCloud, Dropbox, Google Drive, etc.)
- A few limitations for online backup solution
  - What if I don't have Internet connection?
  - What if my Internet connection is low-bandwidth (2G/3G)?
  - Even if I have high-bandwidth Internet connection (4G/LTE), I don't want to pay for the network usage. I will wait until I have free Wi-Fi to back up data
- Even if I have free Wi-Fi, I cannot do the backup continuously and hence the data in the computer/mobile device are not synchronized with the data of the online backup. Why?

# What about Data Recovery?

- Manually/ Automatically back up data in the **local storage** periodically
- Pros: does not rely on network
- Cons:
  - The backup of the entire data will occupy a lot of local storage space
  - The local backup may be also corrupted by the ransomware



A better data recovery strategy?

# Copy-On-Write (COW)

- When the file system reads/ writes files, the data are actually read (by the process) into the memory and written (by the process) back to memory
- Copy-On-Write in the file system:
  - **N** processes read the same file, and only one copy of the file needs to be maintained in the memory (**rather than N copies**)
  - Only after one process modifies the file, a modified copy of the file will be created in the memory in a new location, **but the original copy of the file is still there**
- How can I take advantage of COW for ransomware data recovery (**you will find more details in today's paper presentation**)?

# File System Review

Applications' view:

Files (.doc, .pdf, .txt, ...)



file systems (FAT, NTFS, EXT4, ...),  
implement system calls like open,  
read, write, etc

Manage the mappings between  
the applications' view and the  
block device's view

block device  
interface: allow to  
read/write a block  
of any size and any  
alignment.

Physical storage medium  
(hard disk drive, flash  
storage, etc.)






# Main Operations of A File System – Write System Call

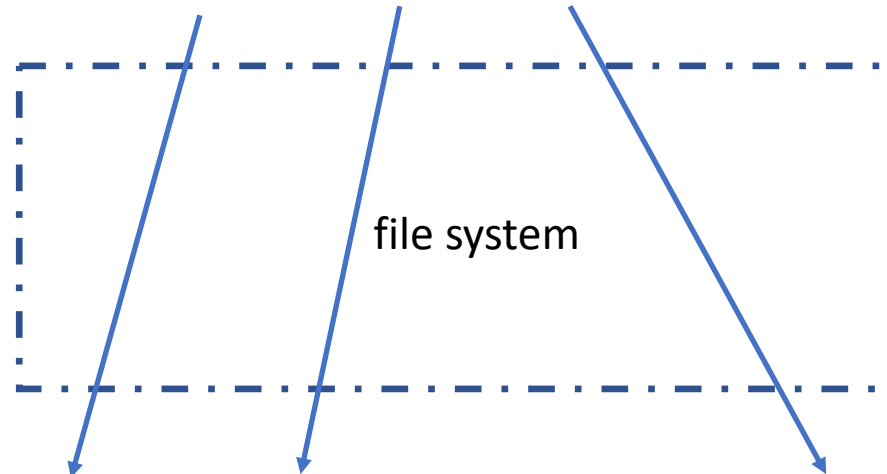
```
#include <unistd.h>
```

```
ssize_t write(int fd, const void *buf, size_t count);
```

Applications' view:



=  +  +   
chunk 1   chunk 2   chunk 3



File system meta-data are used  
to keep track of each block

block device



# Main Operations of A File System – Read System Call

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t count);
```

Applications' view:



=



+



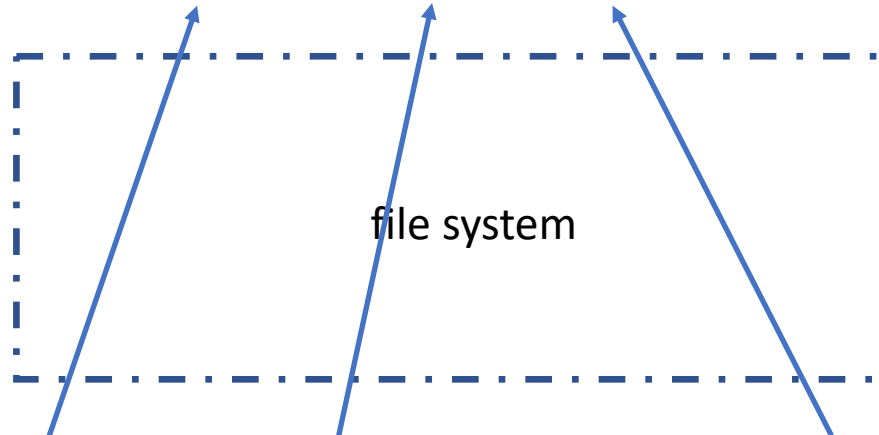
+



chunk 1

chunk 2

chunk 3



file system

Read the meta-data to find out location of each block



block device

# The Efforts of My Research Group on Ransomware/Malware Defenses

- Wen Xie, Niusen Chen, and **Bo Chen**. Incorporating Malware Detection into The Flash Translation Layer. 2020 IEEE Symposium on Security and Privacy (S&P '20), San Francisco, CA, May 2020 (extended abstract).
- Peiyong Wang, Shijie Jia, **Bo Chen**, Luning Xia and Peng Liu. MimosaFTL: Adding Secure and Practical Ransomware Defense Strategy to Flash Translation Layer. The Ninth ACM Conference on Data and Application Security and Privacy (CODASPY '19), Dallas, TX, USA, March 2019.
- Le Guan, Shijie Jia, **Bo Chen**, Fengwei Zhang, Bo Luo, Jingqiang Lin, Peng Liu, Xinyu Xing, and Luning Xia. Supporting Transparent Snapshot for Bare-metal Malware Analysis on Mobile Devices. 2017 Annual Computer Security Applications Conference (ACSAC '17), Orlando, Florida, USA, December 2017 (**Distinguished Paper Award**)
- Kul Prasad Subedi, Daya Ram Budhathoki, **Bo Chen**, and Dipankar Dasgupta. RDS3: Ransomware Defense Strategy by Using Stealthily Spare Space. The 2017 IEEE Symposium Series on Computational Intelligence (SSCI '17), Hawaii, USA, Nov. 27 - Dec. 1, 2017.

# Paper Presentation

- ShieldFS: A Self-healing, Ransomware-aware Filesystem
- Presented by Parker

# ShieldFS: A Self-healing, Ransomware-aware Filesystem

Parker Young | CS5472 Spring 2021 | 3/18/21

Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barengi, A., Zanero, S., & Maggi, F. (2016, December). ShieldFS: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (pp. 336-347).

# Outline

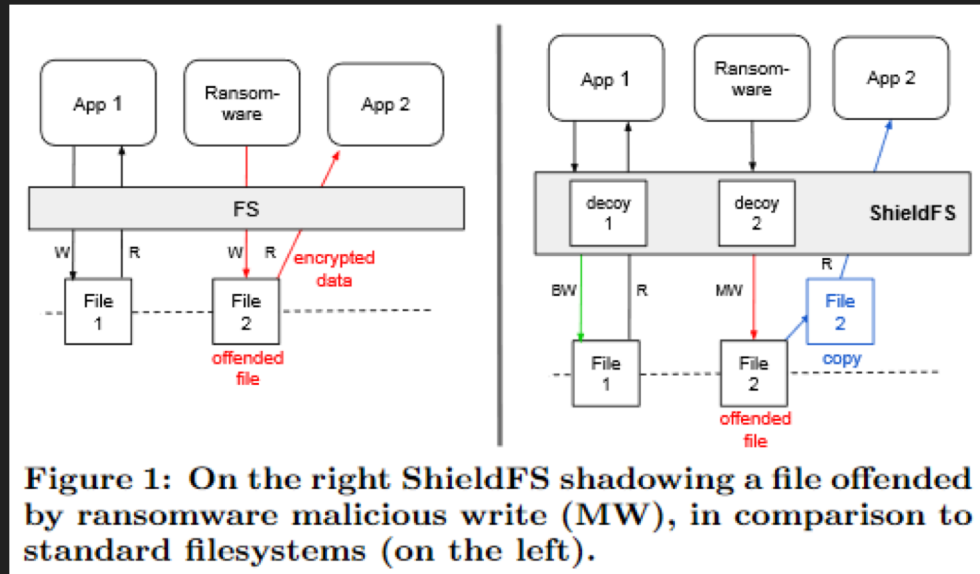
- Ransomware
  - Working at the Filesystem Level
  - Modeling Filesystem Usage
- ShieldFS Specification
  - Detecting Ransomware Filesystem Activity
  - Detecting Cryptographic Primitives in Process Memory
  - Automatic File Recovery
- ShieldFS Implementation
- Testing ShieldFS
  - Cross Validation
  - Testing against Unseen Samples
  - Assessing Overhead
- Potential Limitations
- Conclusions

# Ransomware

- **Malware that encrypts data on a victim's machine**
  - Then asks victim to pay a fee to decrypt the data
- **A Nov 2015 survey found ~50% of victims paid the ransom**
  - From Jan-Mar 2016, **>\$209M worth of payments were made** in the US alone
- **Current ransomware is cryptographically strong**
  - Chances of successful recovery without paying the ransom have **drastically decreased**
- **Current reactive and preventative approaches for ransomware attacks aren't completely effective**

# Working at the Filesystem Level

- Based on previous work, the authors look at monitoring ransomware activity **at the filesystem level**
  - Attempt to **detect and revert changes** made by ransomware

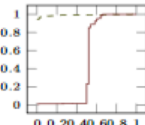
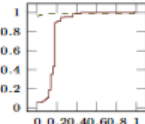
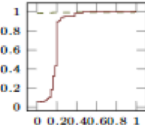
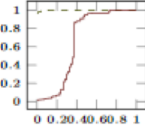
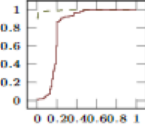
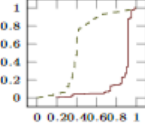


# Modeling filesystem usage

- Need to determine if there exists a significant-enough difference between **benign FS usage** and **malicious FS usage**
- The authors collected the benign model from 11 real-world computers
  - The **I/O Request Packets (IRPs)** from the Windows systems are logged
    - Basic unit of I/O Request in a Windows system
    - Used when communicating between the kernel and drivers
- They then collected samples of how 383 different ransomware samples interacted with a filesystem, based on the previous model

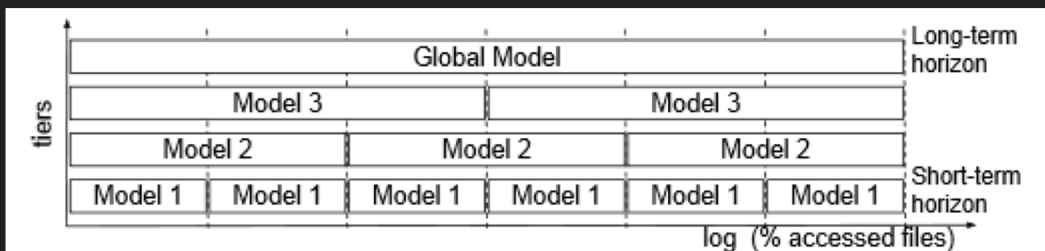


Table 3: We use these features for both our preliminary assessment (Section 3.2) and as the building block of the ShieldFS detector (Sections 3.4). ShieldFS computes each feature multiple times while monitoring each process, on various portions of filesystem activity, as explained in details in Section 3.1. We normalize the feature values according to statistics of the file system (e.g., total number of files, total number of folders). This normalization is useful to adapt ShieldFS to different scenarios and usage habits. The rightmost column shows a comparison of benign (---) vs. ransomware (—) programs by means of the empirical cumulative distribution, calculated on the datasets summarized in Table 1 and 2, respectively. We notice that ransomware activity is significantly different than that of benign programs according to our features, suggesting that there is sufficient statistical power to tell the two types of programs apart.

| Feature                   | Description   | Rationale   | Comparison   |
|---------------------------|---|---|--|
| <i>#Folder-listing</i>    | Number of folder-listing operations normalized by the total number of folders in the system.        | Ransomware programs greedily traverse the filesystem looking for target files. Although filesystem scanners may exhibit this behavior, we recall that ransomware programs will likely violate multiple of these features in order to work efficiently.  |   |
| <i>#Files-Read</i>        | Number of files read, normalized by the total number of files.                                      | Ransomware processes must read all files before encrypting them.  |   |
| <i>#Files-Written</i>     | Number of files written, normalized by the total number of files in the system.                     | Ransomware programs typically execute more writes than benign programs do under the same working conditions.  |   |
| <i>#Files-Renamed</i>     | Number of files renamed or moved, normalized by the total number of files in the system.            | Ransomware programs often rename files appending a random extension during encryption.  |   |
| <i>File type coverage</i> | Total number of files accessed, normalized by the total number of files having the same extensions. | Ransomware targets a specific set of extensions and strives to access <i>all</i> files with those extensions. Instead, benign application typically access a fraction of the extensions in a given time interval.   |   |
| <i>Write-Entropy</i>      | Average entropy of file-write operations.   | Encryption generates high entropy data. Although file compressors are also characterized by high-entropy write-operations, we show that the <i>combined</i> use of all these features will mitigate such false positives. Moreover, we notice that our dataset of benign applications contains instances of file-compression utilities. |  |

# Detecting Ransomware Filesystem Activity

- A **custom classifier** is developed to classify malicious FS activity
  - Uses both a **process-centric** and **system-centric model**
- Multiple classifiers are used concurrently to increase speed and accuracy
  - Classifiers observe the system in intervals, or **ticks**
  - Classifiers are run on different sets of data from within the system
  - Classifiers are **tiered** to focus on both the long-term and short-term history



**Figure 2: Example of the use of incremental models. At each interval, we check simultaneously multiple incremental models at all applicable tiers.**

# Detecting Cryptographic Primitives in Process Memory

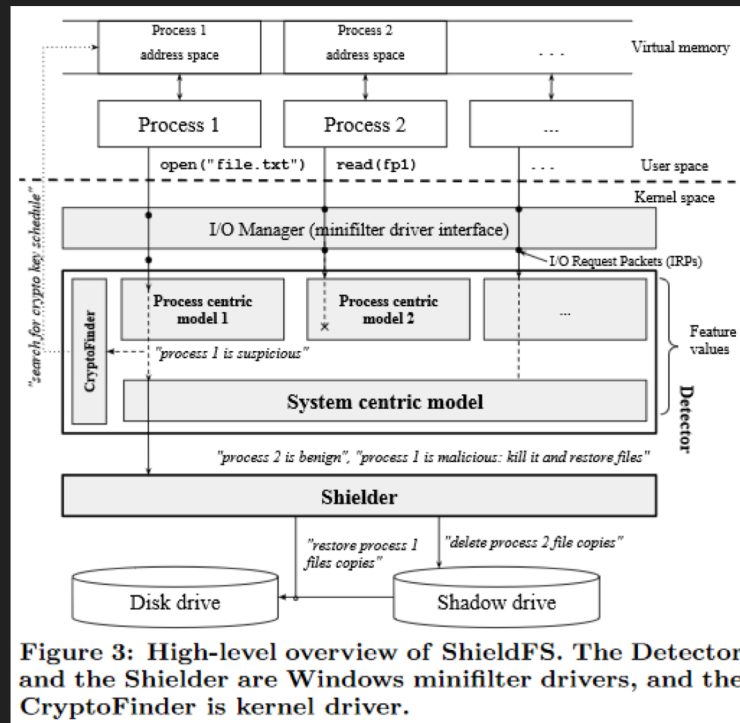
- Scanning a process's memory for its **key schedule** can also lead to flagging a process as ransomware
  - Ransomware is known to use symmetric encryption per-file
  - Ransomware wants to encrypt as many files as fast as possible
- Discovering these keys can also help to further save a user's files

# Automatic File Recovery

- If a process is classified as “*ransomware*” for at least 1 tick, program memory is scanned
  - If a key is found, the process is **suspended** and the **offended files are restored**
  - Otherwise, the system waits for K ticks of “*ransomware*” classification before taking action
- If a process is classified as “*suspicious*”, the **system-centric model is queried**
  - “Suspicious” is given when the process-centric model cannot make a determination
- Any new process is classified in an “*unknown*” state until enough data is gathered on it
  - During this time, the first file that the process attempts to write to or delete is **held in quarantine** by the system

# ShieldFS Implementation

- Classifiers are implemented as **random forests of 100 trees**
  - Each tree outputs **-1** (benign) or **+1** (malicious)
  - Sum of all results gets us final values between **-100** (highly benign) to **+100** (highly malicious)
    - Ties are **"suspicious"**
- 28 tiers of classifiers
  - File set size intervals between 0.1% and 100%
- Whitelisting certain folders/files for performance



# Testing ShieldFS

- **Cross-validation** to assess classifiers
- **Infect real users' computers** to assess the system
  - Ransomware was **correctly identified in all tests** and original files were correctly restored
- Test ShieldFS against ransomware it had **never seen before**
- Assess **performance overhead** of the system

# Cross Validation Results

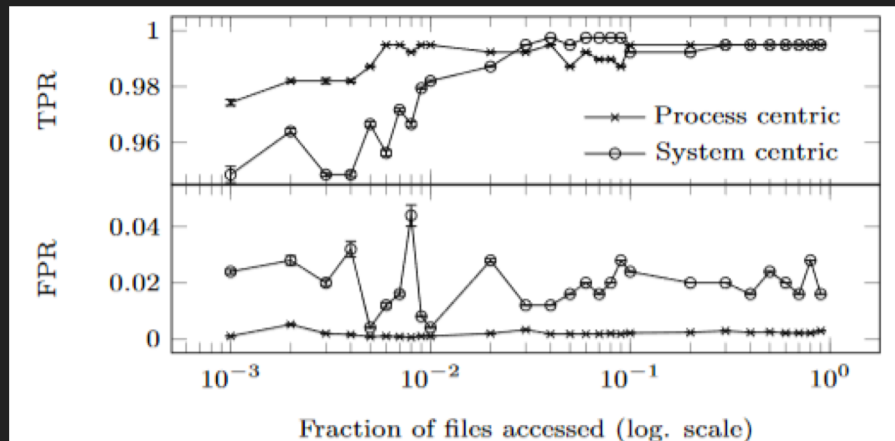


Figure 4: 10-fold Cross Validation: Average and standard deviation of TPR and FPR with process- vs. system-centric detectors.

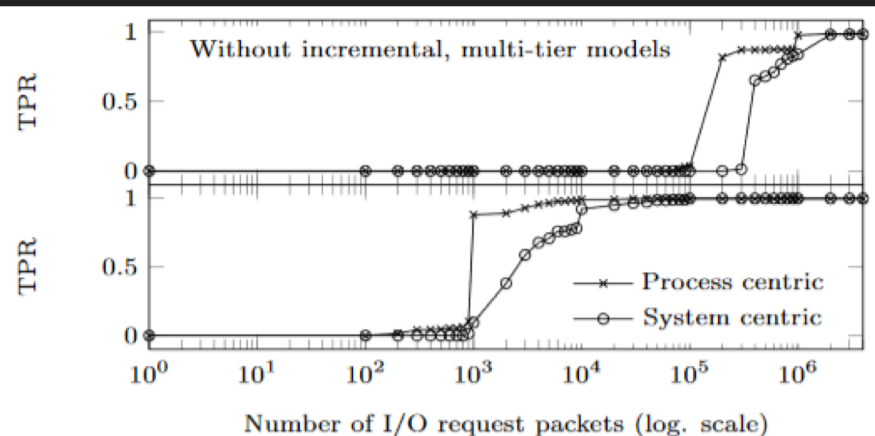


Figure 5: 10-fold Cross Validation: TPR of process- and system-centric detectors, with and without the incremental, multi-tier approach. FPR ranges from 0.0 to 0.0015.

# Cross Validation Results

Table 4: FPR with One-machine-off Cross Validation

| User    | False positive rate [%] |        |             |
|---------|-------------------------|--------|-------------|
| Machine | Process                 | System | Outcome     |
| 1       | 0.53                    | 23.26  | <b>0.27</b> |
| 2       | 0.00                    | 0.00   | <b>0.00</b> |
| 3       | 0.00                    | 0.00   | <b>0.00</b> |
| 4       | 0.00                    | 1.20   | <b>0.00</b> |
| 5       | 0.22                    | 45.45  | <b>0.15</b> |
| 6       | 0.00                    | 4.76   | <b>0.00</b> |
| 7       | 0.00                    | 88.89  | <b>0.00</b> |
| 8       | 0.00                    | 0.00   | <b>0.00</b> |
| 9       | 0.00                    | 0.00   | <b>0.00</b> |
| 10      | 0.00                    | 0.00   | <b>0.00</b> |
| 11      | 0.00                    | 0.00   | <b>0.00</b> |

Table 5: 10-fold Cross-Validation: Choice of  $K$ .

| $K$      | FPR           | TPR         | IRPs         |
|----------|---------------|-------------|--------------|
| 1        | 0.208%        | 100%        | 35664        |
| 2        | 0.076%        | 100%        | 43822        |
| <b>3</b> | <b>0.038%</b> | <b>100%</b> | <b>67394</b> |
| 4        | 0.019%        | 99.74%      | 80782        |
| 5        | 0.019%        | 99.74%      | 104340       |
| 6        | 0.000%        | 99.74%      | 135324       |



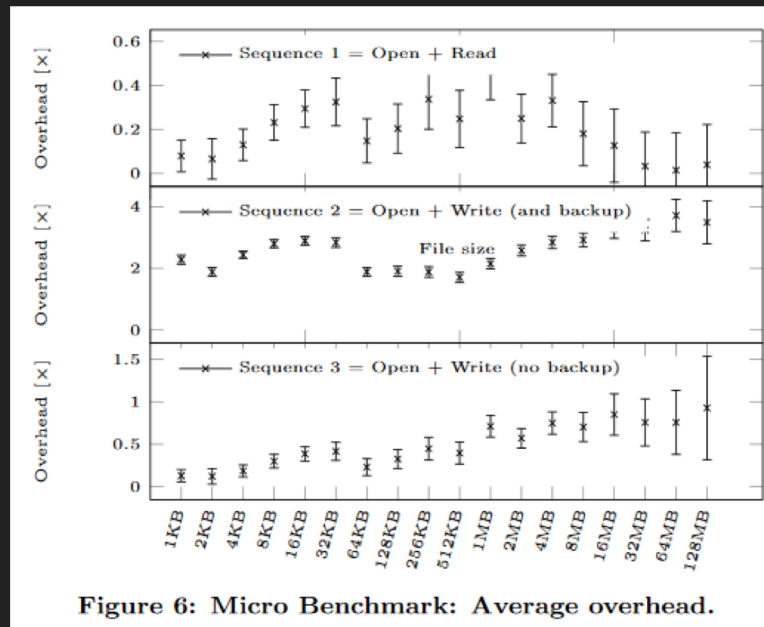
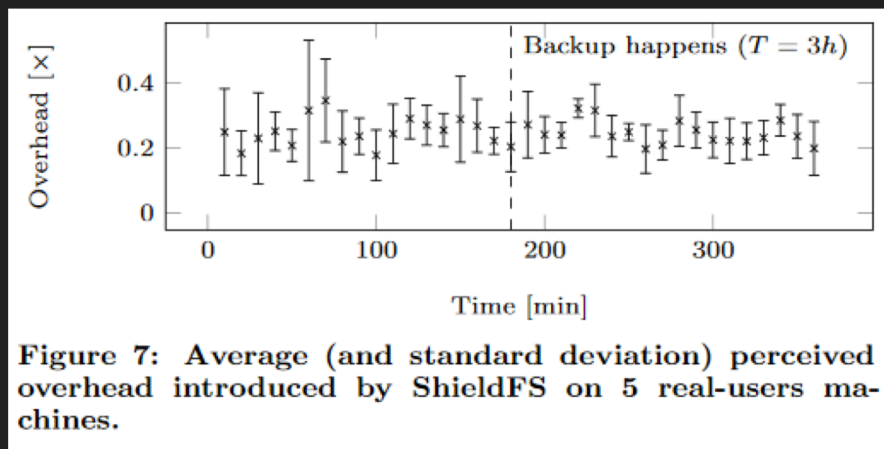
# Testing Against Unseen Samples

- A virtual environment that modeled a real-user's system was setup
- 305 novel ransomware samples were acquired and tested
- **100% of ransomware-encrypted files were restored**

Table 6: Dataset of 305 unseen samples of 11 different ransomware families.

| Ransomware Family | No. Samples | Detection Rate |
|-------------------|-------------|----------------|
| Locky             | 154 (50.5%) | 150/154        |
| TeslaCrypt        | 73 (23.9%)  | 72/73          |
| CryptoLocker      | 20 (6.6%)   | 20/20          |
| Critroni          | 17 (5.6%)   | 17/17          |
| TorrentLocker     | 12 (3.9%)   | 12/12          |
| CryptoWall        | 8 (2.6%)    | 8/8            |
| Troldesh          | 8 (2.6%)    | 7/8            |
| CryptoDefense     | 6 (2.0%)    | 5/6            |
| PayCrypt          | 3 (1.0%)    | 3/3            |
| DirtyDecrypt      | 3 (1.0%)    | 3/3            |
| ZeroLocker        | 1 (0.3%)    | 1/1            |
| <i>Total</i>      | 305         | 298/305        |

# Assessing Overhead



# Assessing Overhead

**Table 7: Measured storage space requirements on real-users machines ( $T = 3h$ ) and cost estimation considering \$3¢/GB.**

| User | Period<br>[hrs] | Storage Required |           | Storage Overhead |             | Max Cost<br>[USD] |
|------|-----------------|------------------|-----------|------------------|-------------|-------------------|
|      |                 | Max [GB]         | Avg. [GB] | Max [%]          | Avg [%]     |                   |
| 1    | 34              | 14.73            | 0.63      | <b>4.29</b>      | <b>0.18</b> | 44.2¢             |
| 2    | 87              | 0.62             | 0.19      | <b>0.95</b>      | <b>0.29</b> | 1.86¢             |
| 4    | 122             | 9.11             | 0.73      | <b>8.53</b>      | <b>0.68</b> | 27.3¢             |
| 5    | 47              | 2.41             | 0.56      | <b>5.49</b>      | <b>1.29</b> | 7.23¢             |
| 7    | 8               | 1.00             | 0.39      | <b>3.35</b>      | <b>1.28</b> | 3.00¢             |

**Table 8: Influence of  $T$  on runtime and storage overhead.**

| $T$<br>[hrs] | Runtime Overhead |             | Storage Space Overhead |          |         |         |
|--------------|------------------|-------------|------------------------|----------|---------|---------|
|              | Avg [×]          | Std.dev [×] | Max [GB]               | Avg [GB] | Max [%] | Avg [%] |
| 1            | 0.263            | 0.0404      | 5.4838                 | 0.4040   | 4.353   | 0.586   |
| 2            | 0.262            | 0.0404      | 5.8402                 | 0.4875   | 4.762   | 0.720   |
| 3            | 0.261            | 0.0403      | 5.5768                 | 0.4994   | 4.522   | 0.746   |
| 4            | 0.260            | 0.0403      | 5.5927                 | 0.5150   | 4.545   | 0.766   |

# Potential Limitations

- **Targeted Evasion** is mostly infeasible
  - The cost of performing this is high enough to deter most attackers from attempting to pursue it
- **Multiprocess Malware** is slightly more feasible, but still unlikely
  - Would need to know the features values (T, K, etc...)
  - Would need to encrypt slowly as to go undetected
  - **User can still restore encrypted files manually**
- **Memory scanning** could be foiled by modern ISA extensions
  - Could easily be remedied by new functionality to the system
- Other smaller/less-likely things
  - Tampering with the Kernel
  - Preventing Denial of Service

# Conclusions

- System works at a low-level and shows that **preventing ransomware from happening in the first place is viable**
- Works like a backup service, but should generally be faster than some traditional backup services
  - **Essentially creates “instant backups” when a file is accessed**
- Overhead is minimal enough that **most users won't perceive a difference**
- Would like to see how such a system would work on other platforms/use-cases
  - macOS/Linux? What about Servers/Data Centers?