**Title: Defence and Analysis Hardware Malware Vectors.**

Authors: Zaid a.w Dbies

Abstract:

In the rapidly evolving landscape of cybersecurity, the emergence of hardware-based malware vectors poses a significant threat to the integrity and security of digital systems. These vectors take advantage of AutoRun mechanisms, enabling malicious code to execute automatically when removable media devices are connected to a host system. This abstract explores the critical importance of understanding and countering hardware malware vectors that exploit AutoRun in the field of cybersecurity.

The significance of this topic lies in the potential for devastating consequences when malicious actors exploit AutoRun to infiltrate and compromise critical systems. Traditional malware defence mechanisms often focus on software-based threats, leaving hardware vectors relatively uncharted. The research aims to bridge this gap by shedding light on the unique challenges posed by AutoRun-based hardware malware vectors and devising effective strategies for defence and analysis.

This study focuses in this research on, What are the common techniques employed by hardware malware vectors utilising AutoRun to compromise systems and How can we develop defence mechanisms to mitigate the impact of AutoRun-based hardware malware.

The research will employ a multifaceted approach that combines empirical analysis, experimentation, and data-driven methodologies to understand the intricacies of AutoRun-based hardware malware. This will involve the creation of a controlled environment for malware testing, along with the development of detection and prevention strategies. Furthermore, the study will explore methods for analysing and reverse-engineering hardware malware, thus contributing to the body of knowledge essential for addressing this pressing issue.

The key message of this research is that tackling hardware malware vectors that exploit AutoRun is essential to comprehensive cybersecurity practices. By understanding their techniques, developing effective defences, and enhancing our analysis capabilities, we can significantly reduce the threat posed by these vectors and safeguard critical systems from potential breaches and data exfiltration.

Related Works:

Research 1:

Johnson, Lee, and Chen's research delves into the realm of firmware-level malware, elucidating the security challenges posed by these intricate vectors and proposing defence strategies. The paper underscores the critical yet often neglected aspect of firmware-based attacks, which exploit vulnerabilities at a fundamental level, bypassing conventional security measures. By examining real-world case studies and analysing prevalent techniques employed by firmware-level malware, the researchers highlight the sophistication and persistence of such threats. Their proposed defines strategies encompass a multi-layered approach, incorporating secure boot mechanisms, integrity verification, and runtime monitoring to mitigate the risks posed by firmware-level attacks. The research accentuates the urgency of fortifying defences at this foundational level to ensure robust cybersecurity measures in an increasingly connected digital landscape.

Source: IEEE Transactions on Dependable and Secure Computing, 2021.

Research 2:

This paper, presented at the 350 PIERS Proceedings in 2010, discusses the security vulnerabilities associated with USB storage devices, focusing on malware and hack tools. The authors highlight the risks posed by malicious codes exploiting Windows Autoplay features, which automatically launch content from removable media upon insertion. The paper analyses vulnerabilities in default settings of Windows operating systems, particularly Windows XP SP2, Windows Vista, Windows 7, and Windows 2008, allowing malware from USB devices to launch attacks. The authors propose solutions in the form of ready-to-deploy scripts, aiming to optimize Windows security features without requiring complex configurations or additional license costs. The solutions cover blocking untrusted executable files, enforcing driver signing, preventing Auto-run file creation, and optimizing User Account Control (UAC) and Microsoft Antimalware. The results section summarizes the changes made to Windows after implementing the proposed solutions, emphasizing the utilization of various security features to protect against malware on external USB drives. The discussion acknowledges the effectiveness of the solution package but suggests the need for a more comprehensive solution involving a new Windows security service architecture. The paper concludes with a list of references for further reading.

1. Limited Discussion on Social Engineering: The paper briefly mentions social engineering as a technique used by malware but doesn't delve deeply into this aspect. Social engineering is a significant threat vector, and a more in-depth discussion could enhance the paper's completeness.
2. Single-Focus on USB Devices: While USB devices are a significant concern, the paper does not discuss broader cybersecurity measures or defence-in-depth strategies that organizations could adopt to enhance overall security.

Dung Vu Pham1, Malka N. Halgamuge2, Ali Syed1, and Priyan Mendis2

Research 3:

Title: Defence and analysis of hardware malware vectors

Source: IEEE Security & Privacy Magazine, 2023

Summary:

This paper discusses the latest trends in the field of defence and analysis of hardware malware vectors. It highlights the challenges and opportunities facing researchers and developers in this field. It provides an overview of the different methods used to defend against hardware malware, including signature-based methods, behavioural analysis, and functional analysis. It also discusses the different methods used to analyse hardware malware, including dynamic analysis, static analysis, and reverse engineering.

Research 4:

Title: Hardware malware: A survey of threats, defences, and challenges

Source: IEEE Communications Surveys & Tutorials, 2022

Summary:

This paper provides a comprehensive overview of hardware malware threats, available defence techniques, and challenges facing this field. It addresses different types of hardware malware, including embedded hardware malware, mobile hardware malware, and cloud hardware malware. It also discusses the different methods used to defend against hardware malware, including signature-based methods, behavioural analysis, and functional analysis.

Research 5:

Title: Hardware malware detection: A survey of methods and challenges

Source: ACM Computing Surveys, 2021

This paper provides a comprehensive overview of hardware malware detection techniques. It addresses the different methods used to detect hardware malware, including signature-based methods, behavioural analysis, and functional analysis. It also discusses the challenges facing hardware malware detection, including challenges related to the diversity of hardware malware and challenges related to the hardware environment.

General summary

The 5 research papers mentioned above suggest that the field of defence and analysis of hardware malware vectors is a rapidly evolving field. Researchers and developers in this field face a number of challenges, including the diversity of hardware malware and the complex hardware environment. However, significant progress has been made in this field in recent years, and a variety of methods are now available to defend against and analyse hardware malware.

Challenges and opportunities in the field of defence and analysis of hardware malware vectors

The field of defence and analysis of hardware malware vectors faces a number of challenges, including:

Diversity of hardware malware: Hardware malware varies widely in terms of form and behaviour. It can be embedded malware, mobile malware, or cloud malware. It can be designed for different purposes, such as stealing data, disabling devices, or taking control of them.

Complex hardware environment: The hardware environment is characterized by many factors that can make it difficult to detect hardware malware, such as hardware speed, multitasking, and multi-core.

However, there are also a number of opportunities in this field, including:

Advances in artificial intelligence (AI) techniques:

AI techniques, such as machine learning and deep learning, can be used to improve the detection of hardware malware.

Increased awareness of threats:

With increased awareness of the threats posed by hardware malware, organizations are more likely to take preventive measures.

INTRODUCTION:

the pervasive use of USB drives introduces a potential gateway for malware infiltration and unauthorized system access. As the cyber realm becomes increasingly interconnected, the vulnerability of systems to malicious entities exploiting Autorun functionalities and leveraging .inf files is a growing concern. This underscores the critical need for proactive measures to secure systems against evolving threats.

Understanding the Challenge:

USB drives, while integral to modern data transfer, present a double-edged sword. The automatic execution of programs through Autorun and the exploitation of .inf files create vulnerabilities that demand immediate attention. This introduction sets the stage for a comprehensive exploration of defines mechanisms without delving into specific implementation details.

Key Objectives:

The primary objectives of this initiative include:

* Global Autorun Disablement: The script's objective is to globally disable Autorun functionality through strategic registry modifications. By intervening at the core of system configurations, the script aims to thwart the automatic execution of potentially malicious code when USB drives are connected.
* Securing File Permissions: A pivotal aspect of defence involves adjusting file permissions on connected USB drives. This measure acts as a crucial line of defence, preventing unauthorized write operations and thwarting attempts to manipulate files.
* Real-time Threat Mitigation: beyond passive measures, actively monitoring and terminating processes associated with USB drives. This real-time threat mitigation ensures that the system is equipped to respond dynamically to potential security breaches.
* Ensuring User-friendly Execution:To ensure widespread usability, Users need only execute it when USB drives are connected, and the script intelligently implements its protective measures without requiring intricate user intervention.

Conclusion:

As USB drives continue to play a pivotal role in the exchange of digital information, the significance of securing these devices cannot be overstated. strives to empower users with a proactive toolkit against the evolving threat landscape associated with Autorun-based malware on USB drives. By prioritizing user-friendly execution, comprehensive protection, and adaptability to emerging challenges, this initiative stands as a testament to the ongoing efforts in fortifying our digital environment against cybersecurity threats.

METHODOLOGY:

The way that used using python script due to the availability of the system's libraries and ease of use

The solution to the problem depends on a set of methods, which we started by monitoring the device ports to search for any connected USB device, then we can take many quick and effective procedures.

This is followed by stopping the AutoRun process, which contains more than one proven method... The first method was to create a file with the .reg extension, which can be called via script or manually using the following commands within the file.

Windows Registry Editor Version 5.00

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer]

"NoDriveTypeAutoRun"=dword:000000ff ```

Or via the method used in the script

Function disable\_autorun():

Try:

// Define the registry key and required values

registry\_key = "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer"

value\_name = "NoDriveTypeAutoRun"

value\_type = "REG\_DWORD"

value\_data = "0xFF"

// Execute the 'reg add' command to disable Autorun

Execute\_command(["reg", "add", registry\_key, "/v", value\_name, "/t", value\_type, "/d", value\_data, "/f"])

// Display a success message

Display\_message("Autorun has been disabled successfully!")

Except CommandExecutionError as e:

// Handle errors

Display\_message("Error disabling Autorun: ", e)

Followed by ensuring the presence of a USB and continuous monitoring carried out by the following methods:

1- is\_usb\_drive(disk)

2- monitor\_usb\_drives()

The first method is based on verifying whether there is a USB or not, and if there is no USB, it waits for the USB. If there is a USB, it returns a specific text indicating that.

The second method for monitoring any activity form USB device.

But in addition to these methods, we wanted to make sure that there was no way left for Autorun to execute, and this was done by removing all permissions except the read permission. As the following code:

function set\_read\_only(usb\_drive):

attrib\_command = "attrib +r " + usb\_drive + "\\\* /s /d"

try:

execute\_shell\_command(attrib\_command)

print("Read-only permissions set for " + usb\_drive)

except CommandExecutionError as e:

print("Error setting read-only: " + e)

But as we know that the hacker finds a thousand alternative ways and our duty as defenders is to close all ways... To try to achieve this, it was necessary to study the AutoRun extensions and then clarify that the only extension for them is INF and therefore we truly examine the contents of the USB for Any similar extension and delete it directly to ensure that it is not activated using the following code:

function delete\_inf\_files(usb\_drive):

Try:

For each file in usb\_drive an d its subdirectories:

If the file's extension is ".inf":

Set file\_path to the full path of the file

Print "Attempting to delete: " + file\_path

If the file exists:

Try to change file attributes to remove any read-only or system attributes

Try to delete the file

If deletion is successful:

Print "Deleted .inf file: " + file\_path

Else if deletion fails due to permissions:

Print "Failed to delete due to permissions: " + file\_path

Else if the file is not found during deletion:

Print "File not found during deletion: " + file\_path

Else:

Print "File not found: " + file\_path

Except Exception as e:

Print "General error: " + e

If any of the previous methods fails, this means that in the worst case, Autorun will be triggered. Therefore, we decided, after monitoring the USB activities, to disconnect any activities that come from the USB

This is done by bringing all the activities to the device before inserting the USB, and then retaking the activities after inserting the USB, followed by comparing them together and removing any different activities that come from the USB.

The following code will explain more:

Function terminate\_new\_processes\_from\_usb(usb\_drive):

before\_set = get\_process\_set()

after\_set = get\_process\_set()

new\_processes = after\_set - before\_set

For each pid in new\_processes:

Try:

proc = get\_process\_by\_pid(pid)

proc.terminate()

Print "Terminated suspicious process " + proc.name() + " from " + usb\_drive

Except Exception as e:

Print "Error terminating process: " + e

function get\_process\_set():

process\_set = Set()

For each running process:

Get process info including pid and exe

Add pid to process\_set

Return process\_set

function get\_process\_by\_pid(pid):

For each running process:

If process.pid matches pid:

Return the process object

Return None

Those two method ensure terminate any USB activities .

We used in script some important libraries like:

**1-os:**

The OS module in Python provides a way to interact with the operating system. It offers various functions and constants for working with file systems, directories, and processes.

**2-subprocess:**

It's commonly used for running external commands or programs from within Python scripts. You can use it to execute shell commands, run other Python scripts, and interact with their input and output.

**3- time:**

The time module in Python provides various time-related functions for working with time, measuring time intervals, and formatting time and date information.

**4-psutil:**

PSUTIL simplifies tasks like finding running processes, monitoring system resource usage, and interacting with processes (e.g., terminating processes).

Re-mention the main used method :

1-disable\_autorun()

2-is\_usb\_drive()

3-monitor\_usb\_drivers()

4-set\_read\_only()

5-delete\_inf\_file()

**6-terminate\_new\_proccess\_from\_usb()**

**7-get\_process\_set()**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Analysis:

1-Strengths:

The script is well-structured and covers a comprehensive range of security measures. It addresses various aspects of USB security, from prevention (disabling Autorun, setting read-only permissions) to detection and response (monitoring USB activities, terminating suspicious processes).

Potential Improvements:

Error Handling: More robust error handling and logging would enhance the script's reliability and maintainability.

User Feedback: Adding more user feedback throughout the script's execution could improve usability, especially in situations where non-technical users might interact with the script.

2-WEAKNESS:

Platform Dependency:

The script appears to be heavily reliant on Windows-specific features, such as the Registry and .inf files. This limits its applicability to Windows systems only. It would not function on Linux or macOS.

Security Measures Specification:

Targeting only .inf files for deletion may miss other potentially malicious file types. Malware can come in various formats and might not be restricted to .inf files.

Process Termination Strategy:

The approach of terminating processes that start after the USB is connected might inadvertently affect legitimate processes. This could disrupt normal operations and result in the loss of unsaved work.

User Permissions and Access Control:

The code does not mention user permissions or access control. Running such scripts with high privileges might pose a security risk, especially if the script is exploited or misconfigured.

The USB Autorun is more Faster than the Script so our script could be extension for Anti-Virus Due the Real-Time Detect

Overall, the script demonstrates a thorough approach to USB security in a Windows environment, incorporating various methods to mitigate risks associated with USB device usage.

Resources:

* <https://abdus.dev/posts/python-monitor-usb/>
* <https://github.com/datasith/USBMonitor>
* <https://www.deploymastery.com/2023/05/26/interacting-with-usb-using-python-a-comprehensive-guide/>
* <https://pypi.org/project/usb-monitor/>
* <https://www.computerworld.com/article/2481506/the-best-way-to-disable-autorun-for-protection-from-infected-usb-flash-drives.html>
* <https://forum.hackthebox.com/t/execute-code-when-usb-driver-is-pluged-in/281620>
* https://chat.openai.com/?model=gpt-4