MES-Attacks: Software-Controlled Covert Channels based on Mutual Exclusion and Synchronization

Chaoqun Shen
Hunan University, China
shencq@hnu.edu.cn

Jiliang Zhang*
Hunan University, China
zhangjiliang@hnu.edu.cn

Gang Qu
University of Maryland, College Park, USA
gangqu@umd.edu

Abstract—Multi-process concurrency is effective in improving program efficiency and maximizing CPU utilization. The correct execution of concurrency is ensured by the mutual exclusion and synchronization mechanism (MESM) that manages the shared hardware and software resources. We propose MES-Attacks, a new set of software-controlled covert channel attacks based on MESM to transmit confidential information. MES-Attacks offer several advantages: 1) the covert channels are constructed at software level and can be deployed on any hardware; 2) the closed shared resource ensures the quality of the channels with low interference and makes them hard to be detected; and 3) the attack utilizes system’s software resources which are abundant and hence difficult to isolate. We report the covert channels we have built with the following MESMs on Linux and Windows: flock, FileLockEX, Mutex, Semaphore, Event and WaitableTimer. Experimental results demonstrate that these covert channels can achieve transmission rate of 13.105 kb/s, 12.383 kb/s, and 6.552 kb/s, respectively in the scenarios of local, cross-sandbox and cross-virtual machine, all with bit error rate under 1%.

Index Terms—covert channel, mutual exclusion, synchronization

I. INTRODUCTION

Modern processors share hardware and software resources among multiple cores, processes, and threads for performance enhancement and resource optimization. However, such resources sharing could bring serious security issues as an attacker can steal secret information via covert channels by manipulating and perceiving the state changes of shared resources [1], or perform transient execution attacks to disclose sensitive data [2]. Covert channels include hardware resource-based channels and software resource-based channels. In the former, hardware resources such as caches [3], memory [4] and branch predictors [5] are reverse engineered to build covert channels, which is normally complicated and can be alleviated effectively by isolation techniques [6], [7]. On the other hand, software resource-based covert channels [8], [9] overcome these drawbacks and are drawing more and more attentions.

We propose MES-Attacks, a set of new covert channels based on shared software resources. These channels exploit a basic process management mechanism in the operating system (OS), known as the mutual exclusion and synchronization mechanism (MESM) [10]. MESM is designed to ensure the correct concurrent execution of multi-processes by coordinating the resource usage among multiple processes. We find that the states of processes might be unintentionally exposed by MESM and hence covert channels could be built to leak sensitive information.

More specifically, MES-Attacks transmit information in a mutual exclusion-based covert channel as follows: one process manipulates another competing process’s execution by controlling the time it occupies the resource, the competing process can deduce the transmitted information by measuring the time for it to acquire the resource. Similarly, in a synchronization-based covert channel, one process manipulates the execution of the other waiting process by the time it satisfies the completion condition. The waiting process infers the transmitted information by measuring the time to release the waiting state.

The contributions of this work are summarized as follows.

- We investigate the classic MESM for the possibility of building new covert channels and implement several such channels using different MESMs on Linux and Windows, including flock, FileLockEX, Mutex, Semaphore, Event and WaitableTimer.
- We build the first cooperation-based covert channel. Compared to existing channels, it simplifies the preparation of attacks, reduces the attack’s impact on the system, and makes the attack difficult to be detected.
- Both the proposed mutual exclusion-based and synchronization-based covert channels are at software level and can be deployed on any hardware. They utilize the OS’s software resources which are abundant and hence are difficult to isolate.
- We perform rigorous experiments to demonstrate that, through the proposed covert channels, we can achieve high transmission rate (TR) and low bit error rate (BER) of 13.105 kb/s, 0.654%; 12.383 kb/s, 0.683%; and 6.552 kb/s, 0.713% respectively in the scenarios of local, cross-sandbox and cross-virtual machine (VM).

II. BACKGROUND

A. Covert Channels

A covert channel normally consists of two malicious processes: Trojan and Spy. Trojan can access secret data, but cannot communicate directly with other processes. In contrast, Spy can access the covert channel but not secret data [11]. The covert channel is established on the collaboration of Trojan and Spy processes where the Trojan process (sender) induces
the system resource changes to encode sensitive information such as cryptography key, which the Spy process (receiver) would observe and infer the encoded information [12]. Covert channels can be categorized into volatile channels and persistent channels, depending on whether the state of the altered resource is retained on the system [13]. In volatile channels, the Trojan and the Spy run concurrently and both dynamically compete for hardware resources. In persistent channels, the Trojan and the Spy can execute asynchronously and the state of the shared resource is retained for a period of time.

### B. Mutual Exclusion and Synchronization

Process mutual exclusion is an indirect constraint between processes. When a process accesses critical resource and enters the critical section, another process that needs to use the resource must wait. Only when the process currently occupying the critical resource exits the critical section, then the waiting process can be released from the blocking state. Process synchronization is a direct constraint relationship between processes that refers to multiple processes waiting or passing information to coordinate the order of their work at synchropoint. For ease of illustration, we refer to the MES states collectively as the constraint states.

### III. Threat Model

We adopt the common covert channel model [8], [9], where a Trojan process has collected secret data from a restricted environment but cannot communicate with a Spy process directly. More specifically, we consider the following three restricted environments: local where the Trojan wants to send data to another process on the same machine, cross-sandbox where the Trojan is inside a sandbox and wants to send data out of the sandbox, and cross-VM where the Trojan wants to send data from one VM to another VM where the Spy is.

It is our intention to remain as general as possible and hence we do not require the attackers (Trojan and Spy) to possess any specific capabilities except that: in the local and cross-VM scenarios, Trojan and Spy can agree on, before the covert channel is established, the shared resources which they have the read permission, but write permission is not needed; in the cross-sandbox scenarios, Trojan in the sandbox cannot write directly to external resources due to the security policy of the sandbox itself. In summary, in all scenarios, we assume that the Trojan and Spy processes cannot write directly to shared resources, otherwise there is no need to design a covert channel for data transmission.

### IV. The Proposed MES-Attacks

The MESMs of Linux OS are all inter-thread operations, except for flock, which can be used directly between processes. To implement MESMs, the two threads need to be connected to shared memory. However, the shared memory must be set to be readable and writable, violating the principle of covert channel, so we can only use the flock to carry out the attack. Similarly, Windows OS can build channels using kernel objects such as FileLockEX, Mutex, Semaphore, Event, WaitableTimer. In this paper, only the six cases mentioned are used, and other MESMs are not discussed. Among them, flock, FileLockEX, Mutex and Semaphore are mutually exclusive mechanisms, Event and WaitableTimer are synchronous mechanisms. This section uses flock and Event as examples.

#### A. Why Choose Process MESMs

Process communication is classed into low-level and high-level communication, depending on the amount and efficiency of exchanged information. Low-level communication can only pass state and control information, including MESM, signal, etc. High-level communication can transfer specific data information directly, including shared memory, pipe, message, etc., but it is susceptible to the direct constraints of OSs. Thus, we focus on the low-level communication – MESMs. This communication methods send signals between processes that can only be used to coordinate the execution of processes by informing them whether to wait or proceed, and which do not contain any data information themselves. To break this restriction, we make the signals sent by MESMs have data information by encoding the time at which a process releases its constraint state (see Section IV.C).

#### B. The Underlying Principle of MES-Attacks

We utilize kernel objects in Windows. Kernel objects are the basic interface between user-mode and kernel-mode code and are data structures maintained by the OSs. To avoid direct modification of the kernel object by user-level programs, processes access it via a created handle. Each process has a private handle table in which only the handles to kernel objects are stored. Handles that point to the same kernel object do not have the same value among different processes, and handles with the same handle value usually point to different kernel objects. The details are shown in Fig. 1.

The kernel objects we used need to work with WaitForSingleObject (abbreviated to WFSO) to implement the MESMs. The function of WFSO is to wait for a specified object until it is in a signaled state or exceeds a set wait interval. In our experiments, the wait interval is set to infinity so that the execution of the WFSO is determined only by the object’s state. The ‘state’ here is a data member of the kernel object.

![Fig. 1: The underlying principle of channels in Windows.](image-url)
and different kernel objects have some different data members, i.e., the red part of Fig. 1.

As shown in Fig. 1, Process A creates the kernel object Event_2, which is automatically reset and has an initial state of non-signalled, and waits for a signal change in Event_2 via WFSO. Process B opens Event_2 via handle 8, and when some operation is completed, the process sets Event_2 to the signalled state. At this point, the WFSO of Process A can end its wait and continues the execution. Therefore, we can construct a covert channel by modulating the time when a Spy process pointing to the same kernel object as the Trojan process is able to request the object.

In Linux, we use file locks which work similarly to kernel objects. In short, each open file corresponds to a system-level data structure, the i-node table, that can be shared between processes. It stores real file information, e.g., file type, access rights, address table, and information about file locks. We design the covert channel by modulating the time that processes pointing to the same i-node can lock the file.

C. Overview of MES-Attacks

The Trojan can encode data through regulating the execution time of the Spy. Concretely, when encoding ‘1’, the Trojan enters a mutually exclusive or synchronous state and takes up resources for longer time. When encoding ‘0’, in the mutual exclusion case, the Trojan enters sleep state directly for shorter time. In the case of synchronization, the Trojan still enters the synchronization state, it only takes up resources for shorter time than sending ‘1’. Thus, the Spy can clearly distinguish the ‘1’ from the ‘0’ by measuring the time to exit the constraint state, i.e., high latency for logical ‘1’ and low latency for logical ‘0’. Fig. 2 shows transmission procedure of MES-Attacks. In Fig. 2 (a), the Trojan (Process 1) controls the blocking time of the Spy (Process 2), and the Spy can infer the data sent by the Trojan based on the time to release the synchronization state as $t = t_{blocking} + t_{occupy}$ or $t = t_{occupy}$. In Fig. 2 (b), the Trojan changes its time to wake up the Spy, and the Spy can infer the data sent by the Trojan based on the time to release the synchronization state as $t = t_{waiting1}$ or $t = t_{waiting2}$.

D. Covert Channel based on Mutual exclusion: flock

Mutually exclusive file locks guarantee that only one process can access file at any given time. This mechanism makes reading and writing files more secure. In Linux, flock is described as an example, which uses LOCK_EX and LOCK_UN to build a covert channel. Other locks that can perform similar functions, such as read lock, are not discussed here.

The relevant communication protocol of flock is Protocol 1 in Fig. 3. When encoding ‘1’, the Trojan enters mutual exclusion state; otherwise, the Trojan enters sleep state. To maintain communication synchronization, $FERIOD_1 = FERIOD_2$.

E. Covert Channel based on synchronization: Event

Due to the uncertainty of the process execution sequence, errors may occur. Therefore, to ensure that processes are executed in the correct order, a process that does not reach the setting condition should block itself. The OS achieves this by means of a synchronization mechanism.

Taking Event as an example, it indicates whether the process can occupy the handle by its “signalled” and “non-signalled”. WFSO allows a thread to block its own execution state.
until the specified non-signalled object is set to signalled. Concretely, the Trojan process calls the `SetEvent` function to set the state to ‘signalled’, which wakes up the `WFSO` of Spy process, thus releasing the Spy process from its blocking state and continuing its execution.

The communication protocol of `Event` is the Protocol 2 in Fig.3. The sender enters the synchronization state when either it sends ‘0’ or ‘1’. However, the time to release the constraint state is different for sending ‘0’ and ‘1’, i.e., `ERESTRUCTION_1 ≠ ERESTRICTION_2`.

**F. Characteristic Analysis**

MES-Attacks are new volatile covert channels. Existing volatile covert channels are all implemented based on inter-process contention, but we found something different. Under the synchronization mechanism, the Spy can only continue execution after the Trojan has met a certain condition; they do not compete with each other for shared resources, but rather modulate the time spent in a constrained state based on a cooperative relationship. Thus, the synchronization-based channel we implemented is a new type of covert channel, a cooperation-based volatile covert channel. It has three advantages: a) Simpler preparation. First, the channel eliminates the need to restore the resource state between the transmissions of each bit. Secondly, the channel only requires a very short synchronization sequence (e.g., one bit); b) Smaller BER. In the channel, each bit is measured in the same initial state. Due to the bit independence, the previous bit error does not affect the subsequent bit correctness; c) Higher system efficiency. The covert channel uses only one shared variable (kernel object) and transmits data by controlling the timing of sleep in different states, which reduces the utilization of system resources and improves efficiency.

Furthermore, MES-Attacks utilize ‘closed’ shared resources. The previous hardware resource-based covert channels exploit ‘open’ shared resources that could be easily accessed by all other processes, hence introducing considerable interference. Conversely, in MES-Attacks, the shared software resource being exploited is a specific object or file that has been negotiated by Trojan and Spy in advance. This ‘closed’ resources setup makes it difficult for other processes to access and exhaust. Thus, the interference can be greatly reduced.

**V. Evaluation**

**A. Experimental Setup**

We conduct experiments on an Intel i5-7400 with 4 cores. The operating system is Ubuntu 16.04 LTS with Linux kernel 4.15.0 and Windows 10. We built covert channels in three scenarios: local, cross-sandbox, and cross-VM.

**B. The synchronization of communication**

To ensure that the Spy process successfully receives the secret data sent by Trojan in covert channels, we implement synchronization between the Trojan and Spy once the communication begins. One way to implement such synchronization is as follows: the Trojan sends an `n`-bit pre-negotiated bitstream (such as “10101010”), called “synchronization sequence”, before transmitting an `m`-bit secret data. The Spy verifies whether the first `n`-bit of the received data are equal to the “synchronization sequence”. Only when they are equal, the Spy considers the subsequent `m`-bit as secret data. Otherwise, the Spy discards the received data and prepares for the next round. At the end of the communication, a long delay is used as a synchronization signal fed to the Trojan when the Spy receives the `n + m` bits, which indicates that this round is over and a new round of communication could be launched.

Besides, we also require a fine-grained synchronization of each bit for the following reasons: the order of the processes executing will affect the correctness of communication; the process scheduling strategy of the OS affects the access to critical resources; the execution time of the instructions themselves affects the accuracy of the measurement.

**C. Performance**

The time parameter determines the channel’s TR and BER. Intuitively, the small time parameter implies high channel performance. However, we found that giving too small time cannot provide sufficient timing difference to correctly recognize what the sent bit is. In this subsection, we discuss how to set the proper time parameters for the three different scenarios.

1) **Local**

In a mutual exclusion-based covert channel, we take `flock` on Linux as an example. Since the process scheduling mechanism requires around 60 µs to wake up the sleep function, we directly set `t_{10}`, i.e., the time to send ‘0’, as 60 µs. However, this restriction is not applied in Windows, and the time to send “0” can be set more flexibly. Thus, there is only one time parameter in this example: the time `t_{11}` to send ‘1’. The experimental results are shown in Fig.4. The overall error rate changes with the time setting, which shows a relationship similar to the “concave function”. When `t_{11} ≤ 160` µs, the BER gradually increases with the decreasing of `t_{11}` due to the limited accuracy of the Spy to distinguish data. When `t_{11} ≥ 220` µs, the number of times that the system is blocked will increase and the spy infers the shorter lock times detected during system blocking are ‘0’s, so the error rate increases. When `160` µs ≤ `t_{11}` ≤ `220` µs, the BER stabilizes below 1%. Meanwhile, the smaller the `t_{11}`, the greater the TR, so we
recommend $t_{t1} = 160 \mu s$ and observe the TR of 7.182 kb/s with a BER of 0.615%.

There are two time parameters in the synchronization-based covert channel: $t_{w_0}$, the time for sending ‘0’; $t_i$, the interval between sending ‘0’ and sending ‘1’. Taking Event as an example, experimental results are shown in Fig.5. Fig.5(a) shows the relationship between time and BER. When $t_{w_0} < 15 \mu s$, the error rate exceeds 1%. It is difficult for the Spy to capture the ‘0’ due to the small $t_{w_0}$, resulting in a larger error rate. When $t_i \leq 30 \mu s$, the error rate also exceeds 1%. Because $t_i$ determines the resolution of the Spy, only when the $t_i$ makes the gap between sending ‘1’ and sending ‘0’ is large enough, the random error caused by the system no more seriously influences the Spy, and the BER remains stable at a low level. Fig.5(b) shows the relationship between time and TR. With the same $t_{w_0}$, a larger $t_i$ indicates that it takes longer to transmit ‘1’, so the rate will be smaller. In the case of keeping $t_i$ unchanged, the larger $t_{w_0}$, the longer the time required to transmit ‘0’ and ‘1’, and hence the lower the rate. Therefore, to keep the error rate below 1% and get the maximum TR, we choose $t_{w_0} = 15 \mu s$ and $t_i = 70 \mu s$, at which point the TR is 13.105 kb/s.

The performance of MES-Attacks in the local scenario is summarized in Table I. Due to the different working mechanism, the cooperation-based channels are subject to less interference than the competition-based channels. As a result, the time to send a ‘1’ and ‘0’ can be set shorter and the TR is significantly higher in a cooperation-based channel. Compared to flock, FileLockEX and Mutex, Semaphore requires more instructions to perform a lock, so the TR of semaphore is lower than in the other cases.

2) Cross-sandbox

Sandboxes are often used to test potentially harmful programs or other malicious code that may have access to sensitive information. However, since the processes running inside the sandbox cannot communicate directly with processes on the system, we can build covert channels to transfer data. We utilize the Firejail in Linux and Sandboxie in Windows.

Table II shows the performance of the cross-sandbox scenario. Its changing trend of TR is the same as the local scenario but lower (see Table I). Because the Trojan and Spy need to ‘break’ the isolation mechanism provided by the sandbox, the cross-sandbox scenario takes longer time to transmit the data.

3) Cross-VM

We conducted experiments using KVM in Linux and Hyper-V in Windows. However, only FileLockEX is valid in Windows. The reason for the situation is that the kernel objects utilised by the FileLockEX-based channels point to read-only files that are shared between VMs; the other objects do not correspond to real resources just generate an identity, thus these objects only exist in their respective space, i.e., they are isolated between VMs. The solution is to find an object that is multiplexed between the VMs and the host, and then make the two VMs build a channel through that multiplexed object. Some system service applications satisfy this requirement in that they are multiplexed between VMs and host, but the names of the objects generated by these applications are concealed. The future work is to resolve the names of the shared objects.

Table III shows the performance in the cross-VM scenario. The reason of decreasing in TR is the Trojan and the Spy are...
In [8], the Trojan sends data by accessing or not the SetEven symbols. For example, when we are encoding a 2-bit symbol, using the example of an event in a local scenario on Windows. On how to increase the amount of information contained in discussion the case of binary code elements where ‘0’ and ‘1’ information each code element carries. The previous experiments elements transmitted per unit time and the amount of information achieves a higher TR, with experiments showing a peak TR number of judgement cases that need to be performed when encoding 3-bit (or more) symbols. Since there are 8 time-setting cases at this point, the no further increase in rate when encoding 3-bit (or more) symbols are observed. We note that encoding 2-bit symbols achieves a higher TR, with experiments showing a peak TR of approximately 15.095 kb/s, higher than the 13.105 kb/s achieved for 1-bit symbol transmission. However, there is S. K. Khatamifard et al., “POWERT channels: A novel class of covert communication exploiting power management vulnerabilities,” in IEEE HPCA, 2019. MESM is one of the fundamental mechanisms for ensuring the concurrent execution of multiple processes. However, since software resources are shared in the OS, MESM can disclose the resource usage among inter-constraint processes. Based on this, we construct covert channels called MES-Attacks, while proposing for the first time a category of channels based on cooperation mechanisms. More importantly, it is difficult to detect and mitigate MES-Attacks due to the specific features of closed resources. We have implemented MES-Attacks in local, cross-sandbox and cross-VM scenarios, and they can achieve TRs of 13.105 kb/s, 12.383 kb/s, and 6.552 kb/s, respectively, all with less than 1% error rate.

TABLE III: The channel performance in cross-VM scenario.

| Attack methods | flock | FileLockEX |
|----------------|-------|------------|
| Timeset(µs)    | t₁₁ = 200, t₁₀ = 70 | t₁₁ = 190, t₁₀ = 70 |
| BER(%)         | 0.832  | 0.713      |
| TR(kb/s)       | 5.893  | 6.552      |

Fig. 6: Multi-bit symbol transmission using 4 combination pairs to encode 2-bit symbols.

VI. SYMBOLS ENCODING MULTI-BITS

The TR of a channel is determined by the number of code elements transmitted per unit time and the amount of information each code element carries. The previous experiments discussed the case of binary code elements where ‘0’ and ‘1’ are transmitted with the same probability. This section focuses on how to increase the amount of information contained in a single transmitted element by using multiple-bit encoding, using the example of an event in a local scenario on Windows.

We set several different time parameters to encode different symbols. For example, when we are encoding a 2-bit symbol, the Trojan encodes ‘00’, ‘01’, ‘10’, ‘11’ by setting the time to perform SetEven to 15 µs, 65 µs, 115 µs and 165 µs. Fig. 6 shows the corresponding result, where all four distinct symbols are observed. We note that encoding 2-bit symbols achieves a higher TR, with experiments showing a peak TR of approximately 15.095 kb/s, higher than the 13.105 kb/s observed for 1-bit symbol transmission. However, there is no further increase in rate when encoding 3-bit (or more) symbols. Since there are 8 time-setting cases at this point, the number of judgement cases that need to be performed when transmitting 3-bit (or more) symbols. Since there are 8 time-setting cases at this point, the number of judgement cases that need to be performed when transmitting data increases and the time required to transmit data as ‘101’, ‘110’ and ‘111’ becomes larger. Thus, the TR does not increase in an integrated view. More sophisticated symbol encoding mechanisms can achieve even higher TR, and our main goal here is simply to demonstrate other methods that adversaries can exploit to achieve higher bandwidths.

VII. RELATED WORK

To the best of our knowledge, there are only two works directly related to the software resource-based covert channels, i.e., the page cache attack [8] and the container-based attack [9]. In [8], the Trojan sends data by accessing or not the specific OS page cache. [9] is launched in multi-tenant cloud services, and the Spy transmits data by reading dynamic identifiers and performance data of the memory-based pseudo-file. In contrast, MES-Attacks have distinct advantages. Page cache attacks [8] have to perform page cache eviction and are more complex to implement, while the MES-Attacks only require the execution of function calls; [8] exploits the OS page cache, whereas MES-Attacks can be implemented through more general resources in the system, such as kernel objects and files. These resources are harder to isolate and therefore more difficult to defend against. And compared to [9], MES-Attacks can be performed in a wider range of scenarios since the MESM is one of the most fundamental technologies in the OSs. In addition, [9] leaks host information through a memory-based pseudo-file, which fluctuates all the time and are sensitive to noise, so it has a higher error rate.

VIII. CONCLUSION

This work was supported by the National Natural Science Foundation of China under Grants No. U20A20202 and 62122023, the Science and Technology Innovation Program of Hunan Province under Grant No. 2021RC4019.

REFERENCES

[1] K. Suzaki et al., “Software side channel attack on memory deduplication,” in ACM SOSP, 2011.
[2] P. Kocher et al., “Spectre attacks: Exploiting speculative execution,” in IEEE S&P, 2019.
[3] H. Fang et al., “Reuse-trap: Re-purposing Cache Reuse Distance to Defend against Side Channel Leakage,” in ACM/IEEE DAC, 2020.
[4] P. Pessl et al., “DRAMA: exploiting DRAM addressing for cross-cpu attacks,” in USENIX Security, 2016.
[5] D. Evtuushkin et al., “Jump over ASLR: attacking branch predictors to bypass ASLR,” in IEEE/ACM MICRO, 2016.
[6] I. Vougioukas et al., “BRB: mitigating branch predictor sidechannels,” in IEEE HPCA, 2019.
[7] Z. Zhou et al., “A software approach to defeating side channels in last-level caches,” in ACM CCS, 2016.
[8] D. Gruss et al., “Page cache attacks,” in ACM CCS, 2019.
[9] X. Gao et al., “A study on the security implications of information leakages in container clouds,” IEEE TDSC, 2021.
[10] W. Vogler, “Efficiency of asynchronous systems that communicate asynchronously,” in ICATPN, 2000.
[11] S. K. Khattamifard et al., “POWERT channels: A novel class of covert communication exploiting power management vulnerabilities,” in IEEE HPCA, 2019.
[12] F. Yao et al., “Are coherence protocol states vulnerable to information leakage?” in IEEE HPCA, 2018.
[13] W. Xiong and J. Szefer, “Survey of transient execution attacks and their mitigations,” ACM Comput. Surv., 2021.