TikTok: Kernel TOCTTOU Protection*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Uroš Tešić
HexHive
EPFL
Lausanne, Switzerland
uros.tesic@epfl.ch

2nd Mathias Payer
HexHive
EPFL
Lausanne, Switzerland
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

System call wrappers enable administrators to define system call execution policies. Such policies could prevent the execution of a system call based on the ID of the call and its arguments. By setting only necessary access policies for all processes, we would reduce the damage in case that any of the processes gets taken over by an adversary. Filtering could also restrict calls to the exploitable system calls. By excluding some combinations of arguments, the administrator could mitigate certain vulnerabilities until a patch is available.

Unfortunately, system call wrappers suffer from a flaw in design. System calls are independent from the filters, and execute after them. After a filter reads the arguments and approves the invocation, the system call reads them in again. In-between these two reads, the attacker could have swapped values, leading to an execution of a forbidden call. This is called a **time-of-check to time-of-use** bug. It is a consequence of a **double-fetch** and is notoriously hard to mitigate.

We present TikTok - a mitigation for double-fetch bugs in Linux kernel. On every access The contribution of this paper are:

- TikTok: An extension to the Linux kernel that prevents any changes to the system-call arguments by using the virtual memory protection mechanisms.
-
- A security analysis of our system and results showing that it is successfull against known double-fetch bugs
- A performance analysis of our system where we show that the performance penalty is comparable to recently merged patches

Identify applicable funding agency here. If none, delete this.

II. Background

A. Interprocess Communication

The two main types of communication between processes are: **shared memory** and **message passing**.

Shared memory relies on processes having a section of memory that both can access. Data tranfer is fast. However, synchronization is problematic. Processes must monitor shared memory for changes, leading to unnecessary busy waiting.

Message passing consists of one process calling send, and another one calling receive to fetch the message. Synchronization is guaranteed, with parties waiting for their calls to be server. However, unnecessary message copying can occur between processes on the same system.

Modern operating systems support both of these approaches. The downsides are usually offset by mixing adding the bare minimum of the other approach (e.g. shared memory with semaphores, or message passing with shared buffers). However, only one of the methods is usually used between two processes. Communication by sending messages and writing to the shared memory at the same time is quite rare.

B. Virtual Memory and Page Tables

Operating systems (OS) provide an illusion that every process is executing alone on the processor. To accomplish this, OS needs to restore the program state on the context switch between two processes (e.g. CPU registers) and to prevent user processes from accessing each other's memory. Memory is protected by virtualizing it. Processes use virtual addresses that get mapped to the actual (physical) addresses. When the OS moves data to a different physical address, the virtual address refering to the data remains the same.

Virtual memory can be implemented by storing different processes' memory at different offsets in physical memory, and limiting the accesses to the corresponding chunk. Each process's memory chunk is called a segment and the implementation is called **segmented virtual memory**.

Considering that segments need to be continous, the free physical memory could be fragmented such that the OS cannot find a chunk large enough to store a new process.

Paged virtual memory is more flexible. The physical memory is partitioned into fixed-size pages (usually 4 kB). A page in the virtual memory space gets mapped to the corresponding page in the physical memory. The mapping function is defined for each process by a page-table. Page-tables store the number of the physical memory (**page frame**) the virtual page maps to. It also keeps the permissions the process has when accessing the page (**read**, **write**, **execute**, **user**, **superuser**). In case of low memory, rarely accessed pages can be swapped to disk, and replaced by immediately needed pages. This process is called **swapping**. Similarly, when processing a file, pages don't need to be loaded immediately, but on the first access (**on-demand paging**). The **present** bit is added to facilitate this.

Virtual memory spaces are quite large today (2^{64} bytes), so they are organized in a tree with multiple levels. Only the allocated pages use memory for book-keeping, while unused subtrees (memory ranges) are marked so. Every time the processor accesses memory, it needs to perform address translation. Different bits in a virtual address correspond to the path page table traversal needs to take to obtain the page frame number (e.g. the first 8 bits tell CPU which entry on the first level it needs to derefernce). Considering that this process needs to be fast, it is implemented in hardware by the **memory management unit** (MMU). Reading the page table from memory is slow, so a small cache is added to the MMU to store frequently accessed page entries - **translation-lookaside buffer** (TLB). On modern processors TLB consists of several levels, and can be even be backed by another MMU cache.

C. x86-64 Page Tables

x86-64 architecture officially supports paged virtual memory model with 5 levels of page tables:

- page global directory (**PGD**)
- page fourth directory (**P4D**)
- page upper directory (**PUD**)
- page middle directory (**PMD**)
- page table entry (**PTE**)

Every level corresponds to 8 bits in the virtual address, with the remaining 12 bits identifying the offset of the byte in the actual page frame. Page table entry includes the following information:

- Present bit (**P**) denotes if the page is present in memory
- Read/Write bit (**R/W**) denotes if the page is writable, or only readable
- User/Superuser bit (**U/S**) denotes if the page can be accessed by the user, or only by the superuser
- Not Executable bit (**NX**) denotes if the code stored on the page can be executed
- Page Frame Number denotes the page frame the page maps to
- Four bits free for the OS to use

D. Page-Fault

On an invalid access (e.g. wrong permissions, page not present) MMU will trigger a page-fault. The fault is a type of a synchronous interrupt which executes in the context of the faulting (accessing) thread. The page-fault handler loads an absent page from the disk. In case of writes temporarily shared pages it creates an independent copy of the page (**copy-on-write**) After the fault finishes execution, the faulting instruction is executed again. In case of a permission violation the page-fault handler kills the thread.

E. Copy-from-User and Copy-to-User

Linux uses swapping only for user memory. When executing in kernel context, kernel memory is mapped and present. A page fault on kernel memory access is therefore always fatal. However, the kernel needs to access user memory, which can cause a page-fault. User memory is also limited to the lower half of virtual memory space, requiring checks on every access.

To force and encapsulate those checks Linux provides functions and macros for user-memory access from the kernel-space. A page-fault generated in them is handled like a fault in the user process which called the executing system call.

The interface for reading from userspace:

- **copy_from_user**
- **__copy_from_user**
- **get_user**
- **__get_user**
- **user_strcpy**
- **user_strlen**

The interface for writing to userspace:

- **copy_to_user**
- **__copy_to_user**
- **put_user**
- **__put_user**

BSD also provides a similar interface using **copy_in** and **copy_out** functions.

F. Double Fetch Bugs

III. Design

In this section we describe a high-level overview of TikTok.

A. Protecting System Call Arguments from Writes by the User

System calls access user memory via `copy_from_user` and its variants. When that happens, we **mark** the entire page storing the argument as **read-only** in all virtual memory spaces mapping it. Multiple system calls can mark a page at the same time. Marking a page does not affect reads from userspace in any way. When all the system calls that use the page exit, the page is **unmarked**.

Writes to marked pages will trigger a page-fault. In the page-fault handler we intercept these writes and make them wait for the page to get unmarked. The faulting thread attempts to perform the write again.

B. Protecting System Call Aguments from Writes by the Kernel

Watson mentions in [**?**] that the system call wrappers he analyzed don't handle writes from kernelspace properly. Unlike those solutions, TikTok doesn't copy arguments to separate pages, leading to complex redirection of userspace pointers. This unables us just to defer the writes until it is safe to perform them.

However, deferring kernel writes the same way as user writes would lead to deadlocks. System call `rt_sigaction` needs to write to a page it previously marked. Pausing execution would leave the thread in a state where it is waiting for itself to exit unmark the page. Temporarily unmarking the page would enable an adversary to edit arbitrary data on it.

Allowing the writes for the kernel is not an option. The adversary could abuse this to change the marked memory. They would execute a read system call into the marked page. System calls execute in the kernel context and would be able to bypass protection. The read system call would then write arbitrary data into the protected area.

Our solution is based on the fact that we already provide partial checkpointing of the system call's view of RAM. During its execution, the system call can only see the state of the memory as it was at the beginning of the call. All writes from userspace become visible only when the call has finished execution. TikTok extends this policy to writes from kernelspace. Considering that we need to continue execution after a write to a marked page, we buffer all the writes until the system call finish. At that point we unmark the pages and allow them to proceed normally.

C. Ignored System Calls

Some system calls (e.g. pollfd) rely on writes from userland for some of their functionality. Marking their arguments would lead to deadlocks, so they are ignored. Considering that these calls use real-time polling of userland memory, we don't consider this a deficiency of TikTok.

Other system calls can be ignored as an optimization. Any unformatted data that is passed to the kernel doesn't need to be marked by TikTok. Overwriting this data is equivalent to passing different data to the system call. Considering that the write call takes unformatted data as one of its arguments, this optimization leads to a significant reduction of pages marked.

D. Two Axes of Linux Memory

Memory in Linux can be **file-backed** and **anonymous**. File-backed pages have map to a corresponding file. Anonymous pages don't have a backing file (e.g. stack and heap).

Another classification is based on privacy: **private** and **shared**. Private memory is part of only one virtual memory space. This memory space can be accessed by multiple threads in a process, but no threads outside the process have access. Shared memory can be accessed from multiple processes.

Unlike private memory and shared anonymous memory, shared file-backed memory can be mapped and unmapped at will. It also preserves its state. This can enable an adversary to map a page as writable and edit it, after it has been marked by another thread. TikTok intercepts mapping of memory and checks the page frame being mapped. The page is then mapped with appropriate permissions into the virtual memory space.

Devices are treated as files in Linux and can be memory mapped. However, hardware may change its registers at will. There are no conceivable ways from protecting from TOCTTOU attacks if the adversary stores his arguments in device mapped memory. Considering that mapping device memory to userland is considered bad practice, we rely on Descretionary Access Control (DAC) to prevent users from mapping devices in the first place.

E. File Writes

Files in Linux can be accessed in two ways:

- by mapping the file to memory
- by using system calls to modify the file (e.g. write)

Watson has noticed that protected file-backed pages could still be edited by a write call. TikTok prevents this attacks by pausing the write to the corresponding file as long as it has any marked mapped pages.

F. TikTok Deadlocks

TikTok adds additional synchronization points to multithreaded programs. It is possible for these points to introduce previously non-existant deadlocks to programs. However, deadlocking threads would need to communicate using both shared memory (for TikTok to stop one of them) and message passing system calls (for TikTok to mark memory).

Figure XXX shows an example of a such communication pattern. Thread 1 has tried to write to a page A marked by

Thread 2. Thread 2 is still in a system call S that marked A, and is waiting for Thread 1 to reach line 2 to proceed. This situation is perculiar:

- The page A is shared between Thread 1 and Thread 2
- Access to page A isn't protected by a mutex, or a semaphore
- System call S is a blocking system call that receives a signal from another thread
- System call S reads its arguments from the page A
- Thread 1 needs to write to the same page where the arguments for S are stored (page A)
- Even though Threads 1 and 2 can communicate using shared memory, Thread 2 needs to invoke S

While a a synchronization call (locking or signaling) would be a good candidate for S, they are lightweight and their arguments are passed in registers, not in memory. A message-passing call fits the description better. Data from page A would need to be marked, as it is read by the call. Message-passing can also be synchronous, requiring the other thread to receive the message before proceeding. However, why would two threads communicate using both message passing and shared memory?

While it is possible to create deadlocking sequences, they require mixing different inter-process communication paradigms for the same data. During testing we haven't encountered a single permanent deadlock.

Similar sequences can be constructed using the write system call protection presented in Subsection III-E. The same argument can be applied in that case - the program would need to write to the same data to the file using both memory mapping and a system call. We haven't encountered such a problem.

IV. Implementation

A. Storing the Page Frame Information

Linux divides physical memory into page frames. Each page frame is represented by `struct page`. Considering that that this structure is replicated millions of times, every additional field has a tremendeous impact on memory consumption. What's even worse, the cost would not be constant, but linear. Systems with more memory would also waste proportionally more RAM.

To keep the memory consumption low, TikTok uses a single bit in `struct page` to mark page frames. Considering that the prototype is implemented on x86-64, we decided to use one of the flag bits for this purpose. Architectures which have fewer flag bits (such as x86) could decide to recycle some of the bits used by other, incompatible features (e.g. Kernel Shared Memory). The marking information is stored separately - in a hashmap. Access to these entries is protected by separate mutexes.

B. Keeping Track of Marked Pages

When TikTok marks a page on x86-64, it uses an extra bit in the PTE to differentiate it from non-marked PTEs. Another bit is used to remember old R/W permission. Depending on which exact bits are used, some kernel features may need to be disabled. In the prototype TikTok shares one of the page table bits with page tracking.

V. Related Work

Literature relating to TikTok can be broadly divided in 2 groups. The first group are system call wrappers and filters whose main vulnerability TikTok is mitigating. The second one are the mitigations and solutions for double-fetches, which are a superclass of TOCTTOU bugs. We describe both groups in this section and discuss the benefits TikTok brings to the first group, and the advantages over the second group.

A. System Call Wrappers and Filters

Watson in [**?**] compared security of different system call wrappers. All of the systems were vulnerable to the TOCTTOU attack TikTok is mitigating. In a short paragraph he mentioned that Pawel Dawidek, the creator of CerbNG [**?**] that he experimented with marking arguments read-only. To our knowledge nothing came out of those experiments. For completeness, CerbNG relies on copying arguments to newly allocated memory pages for protection.

Watson briefly discusses problems the memory marking systems need to solve:

- unnecessary page-faults
- bypassing memory marking using IO system calls
- mapping shared memory late
- handling system calls that write to memory correctly

TikTok addresses all of these problems. Unnecessary page-faults are rare and they are used to make the offending thread wait for completion. After the page has been unmarked, the write proceeds without any consequences. Write system call does not proceed until there are no marked pages of the file. Memory is marked if needed when it is mapped. TikTok also postpones all writes to marked pages coming from kernelspace, while letting the writing system calls execute correctly.

Modern system call wrappers can be classified in two groups, based on how they approach the TOCTTOU attack. The first group eliminates all functionality vulnerable to the attack. Linux's SecComp and eBPF belong to this group. The second group moves the filter checks deeper into the system calls, eliminating the need to read the arguments twice. Landlock Linux and Google's KPSI embrace this technique.

1) Partial Solutions: SecComp uses BPF (Berekley Packet Filter) to provide small, programmable filters that execute before the system call. Based on the values in registers, Linux can decide whether to allow, or to prevent a system call. However, BPF cannot dereference pointers because an adversary would by able to bypass those checks due to the TOCTTOU problem. eBPF (extended Barekley Packet Filter) provides larger filters which can also dereference user pointers. However, eBPF cannot be used for security purposes because it cannot stop system calls from executing. It is completely read-only and can be used only for tracing.

2) LSM-based Solutions: Landlock and KPSI use Linux Security Module [?] (LSM) hooks to call filter checks after the arguments have already been copied into the kernel. LSM hooks have been imagined as a set of places where pointers to functions can be called to perform an arbitrary check. Execution proceeds only if the execution has been successful. Different security modules can provide different hooks to provide different guarantees (e.g. SELinux and AppArmor).

Both Landlock and KPSI attach eBPF filters to hooks, allowing users to provide custom rules for system calls. For this solution to work everywhere for perfect syscall filtering, LSM hooks would need to be manually added to all Linux drivers and ioctls. Unfortunately, this is highly impractical and requires a considerable effort from a large group of developers. TikTok is a generic solution that doesn't require modifying the drivers, nor the use of LSM hooks. Once it is deployed, double-fetches are eliminated from all the drivers.

B. Double-Fetch Solutions

Pengfei Wang et al showed in [?] showed that double-fetches appear not only in kernels, but wherever there is a trust boundary to cross (e.g. kernel – hypervisor). By analyzing reported CVEs they classified double-fetches in several groups based on the code context and their effects (i.e. severity). Another interesting point they raised is that double-fetches can appear in valid code as a result of compiler optimizations.

1) Static Analysis Work: Static analysis techniques analyze the source code to find double-fetch bugs. Wang et al [?] used pattern matching to find potential double-fetches. However, this technique results in a large number of false positives that need to be pruned manually. Xu et al's [?] proposed Deadline - an improved technique that is able to automatically determine if double-fetches result in a potential vulnerability. Instead of relying of pattern matching they compiled the code into the intermediate representation and symbolically executed the code.

While static analysis techniques have the benefits of being able to find the bugs in code that we cannot actually run (e.g. we are missing hardware to test the drivers), it can only detect the bug in the best case. The developers still need to fix it. In case of syscall wrappers, we are aware that the bug is present, but it is there by design. TikTok can prevent double-fetches in cases like this, or when we aren't even aware that they are present (e.g. compiler introduced).

2) Dynamic Analysis Work: Google Project Zero's Bochspwn [?] was an early work on fuzzing kernel code inside an emulator to detect double-fetches. It works on binaries and doesn't require access to the source code. However, it is limited to the detection of double-fetches. All found cases need to be manually triaged and fixed by developers. Furthermore, for a double-fetch to be detected it needs to be executed, limiting this techniques to the core kernel and to the drivers with the available hardware.

Another dynamic analysis technique has been presented by Schwartz et al [?]. The first part of the paper introduces DECAF - a framework that used side-channel attacks to create a fuzzing oracle for double-fetch bugs. Much like Bochspwn, this technique can detect double-fetches invisible to static analysis, but is orders of magnitude faster. DECAF also prunes false positives by trying to automatically exploit double-fetches it has detected.

However, they also discuss a real-time mitigation technique for these bugs - DropIt. Unfortunately, DropIt relies on Intel's Transactional Memory Extension (TSX). TSX introduces limits to the size of the code that can be protected and instructions that can be used inside the protected section. Because of its use in side-channel attacks, both Intel and Linux have dropped supprot for TSX. As a consequence, this technique doesn't work anymore. TikTok has none of the limitations of DropIt and relies only on page access control for protection - a technique which has been already been present for several decades.

VI. Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections VI-A–VI-E below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not number text heads—LaTeX will do that for you.

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".

- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

C. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

D. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

E. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

F. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

G. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

H. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

Fig. 1. Example of a figure caption.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks . . .”. Instead, try “R. B. G. thanks. . .”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first . . .”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.