Secure and Efficient Application Monitoring and Replication

Stijn Volckaert[*†] *Bart Coppens*[†] *Alexios Voulimeneas*[*] *Andrei Homescu*[‡]
Per Larsen[*‡] *Bjorn De Sutter*[†] *Michael Franz*[*]

[*]*UC Irvine* [†]*Ghent University* [‡]*Immunant, Inc.*

Abstract

Memory corruption vulnerabilities remain a grave threat to systems software written in C/C++. Current best practices dictate compiling programs with exploit mitigations such as stack canaries, address space layout randomization, and control-flow integrity. However, adversaries quickly find ways to circumvent such mitigations, sometimes even before these mitigations are widely deployed.

In this paper, we focus on an "orthogonal" defense that *amplifies* the effectiveness of traditional exploit mitigations. The key idea is to create multiple *diversified* replicas of a vulnerable program and then execute these replicas in lockstep on identical inputs while simultaneously monitoring their behavior. A malicious input that causes the diversified replicas to diverge in their behavior will be detected by the monitor; this allows discovery of previously unknown attacks such as zero-day exploits.

So far, such *multi-variant execution environments* (MVEEs) have been held back by substantial runtime overheads. This paper presents a new design, ReMon, that is non-intrusive, secure, and highly efficient. Whereas previous schemes either monitor every system call or none at all, our system enforces cross-checking only for security critical system calls while supporting more relaxed monitoring policies for system calls that are not security critical. We achieve this by splitting the monitoring and replication logic into an in-process component and a cross-process component. Our evaluation shows that ReMon offers same level of security as conservative MVEEs and run realistic server benchmarks at near-native speeds.

1 Motivation

Low-level memory errors can lead to reliability and security problems in systems software implemented in C/C++. In principle, we can eliminate such errors by enforcing spatial and temporal memory safety properties at run time [28, 29]. However, the resulting performance overheads prohibit widespread deployment of such solutions in practice [39].

The ubiquity of multi-core CPUs makes multi-variant execution environments (MVEEs) increasingly attractive to improve the reliability and security of code likely to contain memory corruption vulnerabilities [7, 8, 12, 16, 17, 26, 35, 40, 20]. The idea is to monitor the execution of multiple diversified program replicas for divergence in their observable behavior when an exploit triggers implementation-specific, unintended behavior [21].

Security-oriented MVEEs execute replicas in lockstep and typically perform monitoring at a system call granularity, suspending replicas before system calls and checking their arguments for equivalence. In case of divergence, execution is terminated to limit the effects of an attack. Such MVEEs use operating system processes for isolation between the replicas and the host system and between the replicas and the monitoring component as shown in Figure 1(a). Unfortunately, cross-process monitoring (CP) designs incur substantial performance overheads due to frequent context switching and the resulting translation-lookaside buffer (TLB) and cache flushes.

Hosek et al. [17] developed an alternative reliability-oriented MVEE, VARAN, using in-process (IP) rather than CP monitoring (see Figure 1(b)). VARAN outperforms CP monitors by removing the need for context switching and trades lockstep execution for a loosely synchronized execution model. VARAN, however, does not protect the host system from compromised replicas and is therefore less suitable for security-oriented use cases.

This paper proposes a new, hybrid MVEE design—*ReMon*—that uses an existing, isolated CP monitor (GHUMVEE [42]) to enforce lockstep execution for all sensitive system calls. To increase efficiency, we augment GHUMVEE with a compact, security-hardened IP monitor (IP-MON) that enables efficient replication of non-sensitive calls without context switching. As a result, our design (see Figure 1(c)) unites the strengths of the previous approaches. It provides security guarantees that are comparable to those of existing security-oriented MVEEs while approaching the efficiency of VARAN.

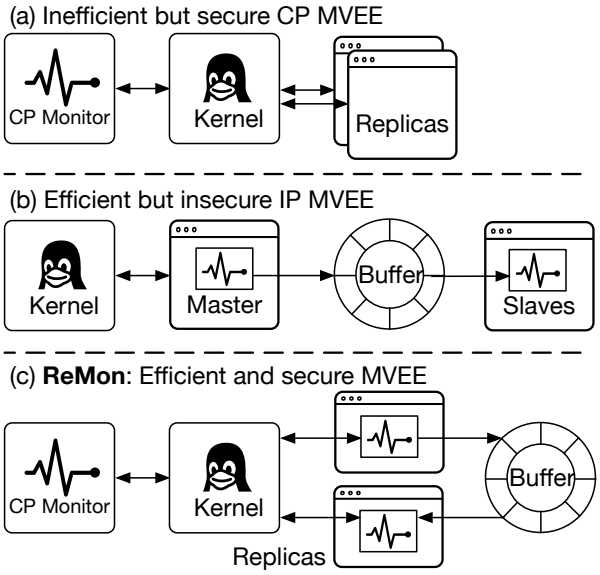

Figure 1: Three MVEE designs running two replicas.

Our design is motivated by the fact that a security policy of monitoring *all* system calls is overly conservative [14, 32]. Many system calls cannot affect any state outside of the process making the system call. Only a small set of *sensitive* system calls are potentially useful to an attacker. Thanks to the IP-MON component, ReMon supports configurable *relaxation* policies that allow non-sensitive calls to execute without being cross-checked against other replicas. Section 5 evaluates the performance impact of a range of relaxation policies inspired by the classification of system calls used in OpenBSD [25].

In summary, our paper contributes:

- **A Novel MVEE Design** ReMon unifies the strengths of previous approaches: the security properties of traditional cross-process MVEEs [8, 12, 35, 40], and the efficient replication mechanism of in-process reliability-oriented MVEEs [17].
- **Relaxed Monitoring Techniques** We leverage our split-monitor design to support relaxed monitoring policies. The IP-MON component lets replicas make certain system calls without cross-process monitoring to increase efficiency.
- **Extensive Evaluation & High Performance** We implemented a full-fledged prototype of our ReMon design and perform a careful and detailed evaluation under different relaxed monitoring policies. Our evaluation shows that ReMon compares favorably to previous work and allows server applications to run in lockstep at near-native speeds.

2 Background

Several MVEE designs have been explored in the past decade. Broadly speaking, two key factors distinguish them. First, an MVEE can run entirely in kernel space [12], or run in user space. Second, some MVEEs execute within the context of the replicas' processes [12, 17], whereas other run in a separate process [10, 35, 40]. These designs make different trade-offs. In-kernel designs are problematic from a security standpoint because MVEE monitors typically have a large attack surface and are prone to memory corruption themselves. The large attack surface arises from the need for the monitor to interpose on every system call executed by a replica, whereas the possibility of memory corruption is due to the plethora of specialized functions that compare and copy complex data structures, such as I/O and message vectors. Successful attacks on MVEE monitors cannot be ruled out, and, in the case of in-kernel monitors, they could easily compromise the entire system.

User-space designs, on the other hand, contain the damage that an attacker can inflict in case the monitor is compromised. Some designs place the MVEE monitor inside the replicas' processes (in-process monitoring / IP), whereas the majority of designs isolate the monitoring process (cross-process monitoring / CP). An IP implementation as shown in Figure 1(b) allows monitor-replica interaction without context switching, but lacks a hardware-enforced protection boundary to isolate the replicas from the monitor. Misbehaving replicas might therefore interfere with the monitor, unless they are augmented with Control-Flow Integrity (CFI) [1] or Software-based Fault Isolation (SFI) [43]. CFI and SFI would, however, reduce or negate the performance benefits of IP monitoring.

In contrast, a CP MVEE (Figure 1(a)) does not require program transformations that slow down the replicas throughout the entire execution. Interaction between a CP MVEE and its replicas does require context switching, however, which is a costly operation due to the need to switch page tables and flush the TLB. When implementing a security-oriented MVEE, the choice between an IP or CP monitoring design is ultimately a trade-off between efficient interaction between the monitor and the replicas (IP design) or faster execution of the replicas (CP design).

2.1 Transparent I/O Replication

The MVEE's monitor ensures that the replicated execution is transparent to the end user. Apart from timing, an outside observer should not notice any differences between native execution of a single replica and Multi-Variant Execution of multiple replicas. The MVEE therefore guarantees that externally observable I/O operations execute only once, while at the same time ensuring that all replicas receive consistent I/O results.

ReMon handles this transparent I/O replication using a master/slave model, similarly to several existing MVEEs. One replica is designated as the master; all other replicas become slaves. Whenever replicas invoke an I/O-related system call, ReMon allows only the master to complete the call. When the call in the master has returned, the

system call results are copied to the slaves' memory and all replicas are resumed.

This mechanism also ensures that all replicas receive consistent input. Consistency and transparency are not identical concerns. Many system calls, e.g., those that query the state of a process, do not have effects that are observable to the end user, but they might still return different results if the monitor does not intervene.

Many programs communicate with other processes over shared physical memory pages. This is generally not safe in an MVEE, however, because (i) it prevents the MVEE from providing consistent input to all replicas, since shared memory can be accessed without system calls, and (ii) shared memory allows for unmonitored bi-directional communication channels between replicas. Such communication channels are a challenge to all security-oriented MVEEs. The ability for the replicas to communicate freely increases the likelihood that attackers can mount an asymmetrical attack, in which they provide different inputs to different replicas.

Security-oriented MVEEs, including ours, therefore typically impose restrictions on the use of shared memory. ReMon rejects any request to set up shared memory pages that can form a bi-directional channel. Typically, this restriction does not break programs, because nearly all of them fall back to alternative communication mechanisms when their requests to map shared memory get rejected. We refer to earlier work for a discussion on solutions to support programs that do not have such a fall-back mechanism [10].

2.2 Consistent Signal Delivery

Whereas synchronous signals (such as SIGSEGV) as a direct result of the executing instruction streams can safely be delivered to all replicas, asynchronously delivered signals can cause the replicas to diverge if their corresponding signal handlers are not invoked at the same point in their execution. Most MVEEs therefore defer the delivery of asynchronous signals until all replicas are suspended in equivalent states. ReMon implements the same strategy. It uses the `ptrace` API to discard signals when they are initially delivered and to re-initiate delivery once all replicas have reached a synchronization point.

2.3 Multi-Threaded Replicas

Non-determinism is a common problem in multi-threaded replicas. Non-deterministic replicas might execute different system call sequences, even if they are given the same inputs and if related system calls are prevented from interleaving. To resolve this, ReMon embeds a small Record/Replay agent in each replica to force them to execute user-space synchronization operations in the same order, thereby enforcing equivalent behavior in all replicas. We refer to the literature for an extended discussion on non-determinism in multi-threaded programs as well as available solutions [4, 6, 13, 22, 30, 33, 34].

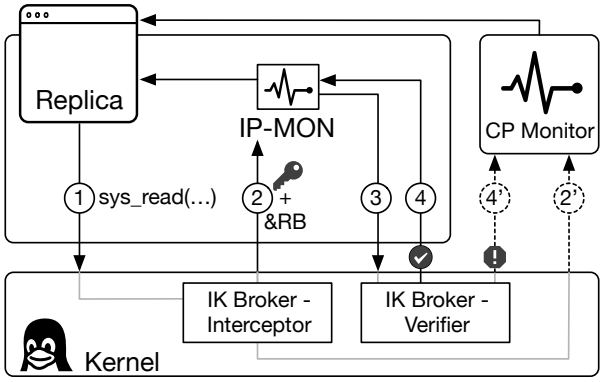

Figure 2: ReMon's major components and interactions.

3 ReMon Design and Implementation

ReMon supervises the execution of an arbitrary number of diversified program replicas that run in parallel. ReMon's main goals are (i) to monitor all of the security-sensitive system calls—hereafter referred to as "monitored calls"—issued by these replicas, (ii) to force monitored calls to execute in lockstep, (iii) to disable monitoring and lockstepping for non-security-sensitive system calls—hereafter referred to as "unmonitored calls", thus allowing the replicas to execute these calls as efficiently as possible while still providing them with consistent system call results, and (iv) to support configurable monitoring relaxation policies that define which subset of all system calls are considered non-security-sensitive, and should therefore not be monitored. ReMon[1] uses three main components to attain these goals:

1. **GHUMVEE** A security-oriented CP monitor implemented as discussed in Section 2. Although GHUMVEE can be used standalone, it only handles monitored calls when used as part of ReMon.

2. **IP-MON** An in-process monitor loaded into each replica as a shared library. IP-MON provides the application with the necessary functionality to replicate the results of unmonitored calls.

3. **IK-B** A small in-kernel broker that forwards unmonitored calls to IP-MON and monitored calls to GHUMVEE. IK-B also enforces security restrictions on IP-MON, and provides auxiliary functionality that cannot be implemented in user space. The broker is aware of the system calls that IP-MON handles and of the relaxation policy that is in effect.

These three components interact whenever a replica executes a system call, as shown in Figure 2.

Our kernel-space system call broker, IK-B, intercepts the system call ① and either forwards it to IP-MON ②, or

[1] `https://github.com/stijn-volckaert/ReMon`

to GHUMVEE ②'. The call is forwarded to IP-MON only if the replica has loaded an IP-MON that can replicate the results of the call, and if the active relaxation policy allows the invoked call to be executed as an unmonitored call. If these two criteria are not met, IK-B uses the standard `ptrace` facilities to forward the call to GHUMVEE instead, which handles it exactly as a regular CP-MVEE.

In the former case, IK-B forwards the call by overwriting the program counter so that the system call returns to a known "system call entry point" in IP-MON's executable code. While doing so, IK-B gives IP-MON a one-time authorization to complete the execution of the call without having the call reported to GHUMVEE. The broker grants this authorization by passing a random 64-bit token ② as an implicit argument to the forwarded call. IP-MON then performs a series of security checks and eventually completes the execution of the forwarded call by restarting it ③. IP-MON can choose to restart the call with or without the authorization token still intact. If the token is intact upon reentering the kernel, IK-B allows the execution of the system call to complete, and returns the call's results to IP-MON ④. If the token is not intact, or if IP-MON executes a different system call, or if the first system call executed after a token has been granted does not originate from within IP-MON itself, IK-B revokes the token and force the call to be forwarded to GHUMVEE ④'.

IP-MON generally executes unmonitored system calls only in the master replica, and replicates the results of the system call to the slave replicas through the replication buffer (RB) discussed in Section 3.2. The slaves wait for the master to complete its system call and copy the replicated results from the RB when they become available.

Although IP-MON allows the master replica to run ahead of the slaves, it still checks if the replicas have diverged. To do so, the master's IP-MON deep copies all of its system call arguments into the RB, and the slaves' IP-MONs compare their own arguments with the recorded ones when they invoke IP-MON. This measure minimizes opportunities for asymmetrical attacks (cf. Section 4).

3.1 Securing the Design

The IK-B Verifier only allows replicas to complete the execution of unmonitored system calls if those calls originate from within an IP-MON instance having a valid one-time authorization token. As only the IK-B Interceptor can generate valid tokens, this mechanism **forces every unmonitored system call to go through IK-B.** At the same time, it also ensures that **IP-MON can only execute unmonitored system calls if it is invoked by IK-B and it is invoked through its intended entry point**. This mechanism is, in essence, a form of Control-Flow Integrity [1]. It also allows us to hide the location of the RB, thereby preventing the RB from being accessed from outside IP-MON. Protecting the RB is of critical importance to the security of our MVEE, as we will discuss in Section 4. To fully hide the location of the RB, while still allowing benign accesses, we ensure that the pointer to the RB is only stored in kernel memory.

IK-B loads the RB pointer and the token into designated processor registers whenever it forwards a call to IP-MON, and IP-MON is designed and implemented such that it does not leak these sensitive values into user space-accessible memory. First, we compile IP-MON using `gcc` and use the `-ffixed-reg` option to remove the RB pointer and authorization token's designated registers from `gcc`'s register allocator. This ensures that the sensitive values never leak to the stack, nor to any other register. Second, we carefully crafted specialized accessor functions to access the RB. These functions may temporarily load the RB pointer into other registers, e.g., to calculate a pointer to a specific element in the RB, but they restore these registers to their former values upon returning. We also force IP-MON to destroy the RB pointer and authorization token registers themselves upon returning to the system call site. Finally, we use inlining to avoid indirect control flow instructions from IP-MON's system call entry point. This ensures that IP-MON's control flow cannot be diverted to a malicious function that could leak the RB pointer or authorization token.

ReMon further prevents discovery of the RB through the /proc/maps interface: It forcibly forwards all system calls accessing the maps file to GHUMVEE and by filtering the data read from the file. This requires marking the maps file as a special file, as described in Section 3.6.

To prevent IP-MON itself from being tampered with, we also force all system calls that could adversely affect IP-MON to be forwarded to GHUMVEE. These calls (e.g. `sys_mprotect` and `sys_mremap`) are then subject to the default lockstep synchronization mechanism.

3.2 The IP-MON Replication Buffer

IP-MON must be embedded into all replicas, so it consists of multiple independent copies, one per replica. These copies must cooperate, which requires an efficient communication channel. Although a socket or FIFO could be used, we opted for a shared replication buffer (RB) stored in a memory segment shared between all replicas.

To increase the scalability of our design, we opted not to use a true circular buffer. Instead, we use a linear RB. When our RB overflows, we signal GHUMVEE using a system call. GHUMVEE then waits for all replicas to synchronize, resets the buffer to its initial state, and resumes the replicas. Involving GHUMVEE as an arbiter avoids costly read-write sharing on RB variables that keep track of where data starts and ends in the RB. Instead, each replica thread only reads and writes its own RB position.

3.3 Adding System Call Support

ReMon currently supports well over 200 system calls. To provide a fast path, IP-MON supports a subset of 67

```
/* read(int fd, void * buf, size_t count) */
MAYBE_CHECKED(read) {
  // check if our current policy allows us to dispatch read
  // calls on this file as unmonitored calls
  return !can_read(ARG1);
}

CALCSIZE(read) {
  // reserve space for 3 register arguments
  COUNTREG(ARG);
  COUNTREG(ARG);
  COUNTREG(ARG);
  // one buffer whose maximum size is in argument 3 of syscall
  COUNTBUFFER(RET, ARG3);
}

PRECALL(read) {
  // compare the args each replica passed to the call.
  // if they match, we allow only the master to complete the call,
  // while the slaves wait for the master's results.
  CHECKREG(ARG1);
  CHECKPOINTER(ARG2);
  CHECKREG(ARG3);
  return MASTERCALL | MAYBE_BLOCKING(ARG1);
}

POSTCALL(read) {
  // replicate the results
  REPLICATEBUFFER(ARG2, ret);
}
```

Listing 1: Replicating the `read` system call in IP-MON.

system calls. However, adding support to IP-MON for a new system call is generally straightforward. IP-MON offers a set of C macros to easily describe how to handle the replication of the system call and its results.

As an example, listing 1 shows IP-MON's code for the `read` system call. The code is split across four handler functions that each implement one step in the handling of a system call using the C macros provided by IP-MON.

First, the MAYBE_CHECKED function is called to determine if the call should be monitored by GHUMVEE. If the MAYBE_CHECKED handler returns `true`, IP-MON forces the original system call to be forwarded to GHUMVEE (④) by destroying the authorization token and restarting the call. We use this handler type to support conditional relaxation policies, as shown in Table 1.

IP-MON uses a fixed-size RB to replicate system call arguments, results, and other system call metadata. Prior to restarting the forwarded call, we therefore need to calculate the maximum size this information may occupy in the RB. If the size of the data as calculated by the CALCSIZE handler exceeds the size of the RB, IP-MON forces the original system call to be forwarded to GHUMVEE. If the data size does not exceed the size of the RB, but it is bigger than the available portion of the RB, the master waits for the slaves to consume the data already in the RB, after which it resets the RB.

Next, if IP-MON has decided not to forward the original system call to GHUMVEE, it calls the PRECALL handler. In the context of the master replica, this function logs the forwarded call's arguments, call number, and a small amount of metadata into the RB. This metadata consists of a set of boolean flags that indicate whether or not the master has forwarded the call to GHUMVEE, whether or not the call is expected to block when it is resumed, etc. If the function is called in a slave replica's context, IP-MON performs sanity checking by comparing the slave's arguments with the master's arguments. If they do not match, IP-MON triggers an intentional crash, thereby signalling GHUMVEE through the `ptrace` mechanism, and causing a shutdown of the MVEE.

The return value of the PRECALL handler determines whether the original call should be resumed or aborted. By returning the `MASTERCALL` constant from the PRECALL handler, for example, IP-MON instructs the master replica to resume the original call, and the slave replicas to abort the original call. Alternatively, the original call may be resumed or aborted in all replicas.

Finally, IP-MON calls the POSTCALL handler. Here, the master replica copies its system call return values into the RB. The slave replicas instead wait for the return values to appear in the RB. Depending on the aforementioned system call metadata, the handler may wait using a spin-wait loop if the system call was not expected to block, or otherwise a specialized condition variable, whose implementation we describe in Section 3.7.

3.4 System Call Monitoring Policies

There are many ways to draw the line between system calls to be monitored by the CP-MVEE and system calls to be handled by IP-MON. We propose two concrete monitoring relaxation policies.

The first option is **spatial exemption**, where certain system calls are either unconditionally handled by IP-MON and not monitored by GHUMVEE, or handled by IP-MON only if their system call arguments meet certain criteria. Table 1 proposes several predefined levels of spatial exemption, which the program developer or administrator can choose from. Selecting a level enables unmonitored system calls for all calls in that level, as well as all preceding levels. This provides a performance-security trade-off, with lower levels in the table having lower overhead but being potentially less secure.

We picked these system calls so we could maintain a high level of security while still preserving the correctness of the replicas' execution and significantly improving our system's performance. System calls that relate to allocation and management of process resources and threads, as well as signal handling, are always monitored by GHUMVEE. This includes syscalls that (i) allocate, manage and close file descriptors (FDs), (ii) map, manage and unmap memory regions, (iii) create, control and kill threads and processes and (iv) all signal handling sys-

Level and description	*Unconditionally allowed calls*	*Conditionally allowed calls depending on*	
		file type	*op type*
BASE_LEVEL Read-only calls that do not operate on file descriptors and do not affect the file system.	gettimeofday, clock_gettime, time, getpid, gettid, getpgrp, getppid, getgid, getegid, getuid, geteuid, getcwd, getpriority, getrusage, times, capget, getitimer, sysinfo, uname, sched_yield, nanosleep		
NONSOCKET_RO_LEVEL Read-only calls on regular files, pipes, and non-socket file descriptors; read-only calls from file system; write calls on process-local variables.	access, faccessat, lseek, stat, lstat, fstat, fstatat, getdents, readlink, readlinkat, getxattr, lgetxattr, fgetxattr, alarm, setitimer, timerfd_gettime, madvise, fadvise64	read, readv, pread64, preadv, select, poll	futex, ioctl, fcntl
NONSOCKET_RW_LEVEL Write calls on regular files, pipes, and other non-socket file descriptors.	sync, syncfd, fsync, fdatasync, timerfd_settime	write, writev, pwrite64, pwritev	
SOCKET_RO_LEVEL Read calls on sockets.	read, readv, pread64, preadv, select, poll, epoll_wait, recfrom, recvmsg, recvmmsg, getsockname, getpeername, getsockopt		
SOCKET_RW_LEVEL Write calls on sockets.	write, writev, pwrite64, pwritev, sendto, sendmsg, sendmmsg, sendfile, epoll_ctl, setsockopt, shutdown		

Table 1: Monitor levels for spatial system call exemption.

tem calls. We distributed all remaining system calls over the aforementioned levels to allow the programmer/administrator to choose the appropriate balance between performance and security.

The second option is **temporal exemption**, where IP-MON probabilistically exempts system calls from the monitoring policy if similar calls were repeatedly approved by the monitor. We observe that many programs, especially those with high system call frequencies, often repeatedly invoke the same sequence of system calls. If a series of system calls is approved by GHUMVEE, then one possible temporal relaxation policy is to stochastically exempt some fraction of the following identical system calls within some time window or range. Note that temporal relaxation policies must be highly unpredictable; deterministic policies (e.g., "Exempt system calls X, Y, Z from monitoring after N approvals within an M millisecond time window") are insecure. In other words, care must be taken to ensure that temporal relaxation does not allow adversaries to coerce the MVEE into a state where potentially dangerous system calls are not monitored.

3.5 IP-MON Initialization

IK-B does not forward any system calls to IP-MON until IP-MON explicitly registers itself through a new system call we added to the kernel. When this call is invoked, the kernel first attempts to report the call to GHUMVEE, which receives the notification and can decide if it wants to allow IP-MON to register.

The registration system call expects three arguments. The first argument is the set of "unmonitored" calls supported by IP-MON. If the IP-MON registration succeeds, IK-B forwards any system call in this set to IP-MON from that point onwards, as we explained earlier. GHUMVEE can modify this set of system calls, or potentially prevent the registration altogether. The second and third arguments are a pointer to the RB and a pointer to the entry point function that should be invoked when IK-B forwards a call to IP-MON.

The RB pointer must be valid and must point to a writable region. IP-MON must therefore set up an RB that it shares with all other replicas. We use the System V IPC facilities to create, initialize, and map the RB [23]. GHUMVEE arbitrates the RB initialization process to ensure that all replicas attach to the same RB.

3.6 The IP-MON File Map

GHUMVEE arbitrates all system calls that create/modify/destroy FDs, incl. sockets. It thus maintains metadata such as the type of each FD (regular/pipe/socket/poll-fd/special). It also tracks which FDs are in non-blocking mode. System calls that operate on non-blocking FDs always return immediately, regardless of whether or not the corresponding operation succeeds.

Replicas can map a read-only copy of this metadata into their address spaces using the same mechanism we use for the RB. We refer to this metadata as the IP-MON file map. We maintain exactly one byte of metadata per FD, resulting in a page-sized file map. For some system calls, IP-MON uses the file map to determine if the call is to be monitored or not as per the monitoring policy.

3.7 Blocking System Calls

Its file map permits IP-MON to predict whether an unmonitored call can block or not. IP-MON handles blocking calls efficiently. If the master replica knows that a call will block, it instructs the slaves to wait on an optimized and highly scalable IP-MON condition variable until the results become available (as opposed to a slower spin-read loop). IP-MON uses the `futex (7)` API to implement wait and wake operations. This allowed us to implement several optimizations.

For each system call invocation, IP-MON allocates a separate structure within the RB. Each individual structure contains a condition variable. Slave replicas must only wait on the condition variable associated with the system call results they are interested in. Using separate condition variables for each system call invocation prevents an unnecessary bottleneck that would arise when using just a single variable, because the slave replicas might progress at different paces. Furthermore, IP-MON tracks whether or not there are replicas waiting for the results of a specific system call invocation. If none are waiting when the master has finished writing its system call results into the buffer, no `FUTEX_WAKE` operation is needed to resume the slaves. IP-MON does not have to

reuse condition variables because a new condition variable is allocated for each system call invocation. Thus, IP-MON does not have to reset condition variables to their initial state after it has used one to signal slave replicas.

3.8 Consistent Signal Delivery

Signals may introduce divergence among a set of executing replicas. MVEEs therefore typically defer the delivery of signals until they can assert that all replicas are in equivalent states, such as when they are all waiting to enter a system call, as discussed in Section 2.2.

The intricacies of the `ptrace` API make the correct implementation of consistent asynchronous signal delivery challenging, and it becomes even more complicated when introducing IP-MON. Because GHUMVEE does not see any system calls that are dispatched as unmonitored calls, it might indefinitely defer the delivery of incoming signals, thus violating the intended behavior of the replicas. GHUMVEE solves this problem via introspection. When a signal is delivered to the master replica, GHUMVEE first sets a *signals pending* flag, which is stored at the beginning of the RB. Next, GHUMVEE checks whether that replica was executing a system call through IP-MON. GHUMVEE does this by checking if the user-space instruction pointer points to a system call instruction inside the IP-MON executable region. If the master replica was executing a blocking system call, GHUMVEE aborts that call. The kernel automatically aborts blocking system calls, but normally restarts them after the signal handler has been invoked. However, GHUMVEE prevents the kernel from restarting the call. Instead, it resumes the master replica at the return site of the call. The master replica then inspects the *signals pending* flag and restarts the call as a monitored call, allowing it to be intercepted by GHUMVEE.

3.9 Support for `epoll (7)`

Linux 2.5.44 introduced the Linux-specific `epoll` API as a high-performance alternative to `select` and `poll`. Applications can use this interface to get notifications for FD events, e.g., when a socket has received new data or when a connection request has arrived. Modern Linux server applications use `epoll` to handle network requests efficiently on multiple threads.

To minimize the performance overhead, IP-MON needs to support the `epoll` family of system calls. This is not straightforward, however. When registering a FD with `epoll` functions, the application can associate an `epoll_event` structure with that FD. This structure may contain a pointer value that the kernel will return when an event on the FD gets triggered. The `epoll_event` structures are challenging to support in MVEEs. Diversified replicas are likely to use different pointer values for the same logical FD. Blindly replicating the results of a `sys_epoll_wait` event would then return the master's, rather than the calling replica's pointer values.

IP-MON solves this problem by maintaining a shadow mapping between FDs and pointers inside `epoll_event`. When a new FD is registered with `epoll`, IP-MON copies the associated pointer value from the `epoll_event` structure to the mapping. When replicating the results of an `epoll` call, IP-MON uses this mapping to store FDs, rather than pointer values in the master replica, and it maps these FDs back onto the associated pointer values in the slave replicas.

4 Security Analysis

Unlike previous MVEEs, ReMon eschews fixed monitoring policies and instead allows security/performance trade-offs to be made on a per-application basis.

With respect to integrity, we already pointed out that a CP MVEE monitor (and its environment) are protected by (i) running it in an isolated process space protected by a hardware-enforced boundary to prevent user-space tampering with the monitor from within the replicas; (ii) by enforcing lockstep, consistent, monitored execution of all system calls in all replicas to prevent malicious impact of a single compromised replica on the monitor; and (iii) diversity among the replicas to increase the likelihood that attacks cause observable divergence, i.e., that they fail to compromise the replicas in consistent ways.

With those three properties in place, it becomes exceedingly hard for an attacker to subvert the monitor and to execute arbitrary system calls. Nevertheless, MVEEs do not protect against attacks that exploit incorrect program logic or leak information though side-channel attacks. This is similar to many other code-reuse mitigations such as software diversity, SFI, and CFI.

In ReMon, monitored system calls are still handled by a CP monitor, so malicious monitored calls are as hard to abuse as they are in existing CP MVEEs. For unmonitored calls, IP-MON relaxes the first two of the above three properties. The master replicas can run ahead of the slaves and the system call consistency checks in the slaves' IP-MON, so an attacker could try to hijack the master's control with a malicious input to execute at least one, and possibly multiple, unmonitored calls without verification by a slave's IP-MON. An attacker could also attempt to locate the RB and feed malicious data to the slaves, in order to stall them or to tamper with their consistency checks. This way, the attacker could increase the window of opportunity to execute unmonitored calls in the master.

As long as the attacker executes unmonitored calls only according to a given relaxation policy, those capabilities by definition pose no significant security threat: unmonitored calls are exactly those calls that are defined by the chosen policy to pose either no security threat at all, or that pose an acceptable security risk. However, an attacker can also try to bypass IP-MON's policy verification checks on conditionally allowed system calls to let IP-MON pass calls unmonitored that should have been

monitored by GHUMVEE according to the policy. We therefore consider several aspects of these attack scenarios in the following paragraphs.

Unmonitored execution of system calls ReMon ensures that IP-MON can only execute unmonitored system calls if it is invoked by IK-B itself and through its intended system call entry point. When invoked properly, IP-MON performs policy verification checks on conditionally allowed system calls, as well as the security checks a CP monitor normally performs. An attacker that manages to compromise a program replica could jump over these checks in an attempt to execute unmonitored system calls directly. Such an attack would, however, be ineffective thanks to the authorization mechanism we described in Section 3.1.

Manipulating the RB We designed IP-MON so that it never stores a pointer to the RB, nor any pointer derived thereof, in user-space accessible memory. Instead, IK-B passes an RB pointer to IP-MON, and IP-MON keeps the RB pointer in a fixed register. To access the RB, the attacker must therefore find its location by random guessing or by mounting side-channel attacks. ReMon's current implementation uses RBs that are 16MiB and located on different addresses in each replica. This gives the RB pointer 24 bits of entropy per replica which makes guessing attacks unlikely to succeed.

Furthermore, because neither IP-MON, nor the application need to know the exact location of the RB and because every invocation of IP-MON is routed through IK-B, we could extend IK-B to periodically move the RB to a different virtual address by modifying the replicas' page table entries. This would further decrease the chances of a successful guessing attack.

Diversified Replicas Our current implementation of ReMon deploys the combined diversification of ASLR and Disjoint Code Layouts (DCL) [40]. ReMon, however, support all other kinds of automated software diversity techniques as well. We refer to the literature for an overview of such techniques [21]. The security evaluations in the literature, including demonstrations of resilience against concrete attacks, therefore still apply to ReMon.

5 Performance Evaluation

In this section, we first evaluate the performance of IP-MON's spatial relaxation policy on a set of widely-used benchmark suites, and then compare IP-MON with existing MVEEs by replicating some of the experiments previously described in the literature [16, 17, 26, 35, 40]. We conducted all of our experiments on a machine with two eight-core Intel Xeon E5-2660 processors each having 20MB of cache, 64GB of RAM and a Gigabit Ethernet connection, running the x86_64 version of Ubuntu 14.04.3 LTS. This machine runs the Linux 3.13.11 kernel, to which we applied the IK-B patches described in Section 3. These IK-B patches add 97 LoC to the kernel. We used the official 2.19 versions of GNU's `glibc` and `libpthreads` in our experiments, but did apply a small patch to `glibc` to reinitialize IP-MON's thread-local storage variables after each fork. We disabled hyper-threading as well as frequency and voltage scaling to maximize reproducibility of our measurements.

Address Space Layout Randomization (ASLR) was enabled in our tests and we configured ReMon to map IP-MON and its associated buffers at non-overlapping addresses in all replicas [40].

5.1 Synthetic Benchmark Suites

We evaluated ReMon on the PARSEC 2.1, SPLASH-2x, and Phoronix benchmark suite. (C. Segulja kindly provided his data race patches for PARSEC and SPLASH [36].) These benchmarks cover a wide range in system call densities and patterns (e.g., bursty vs. spread over time, and mixes of sensitive and non-sensitive calls) as well as various scales and schemes of multi-threading, the most important factors contributing to the overhead of traditional CP-MVEEs that we want to overcome with IP-MON.

We evaluated all five levels of our spatial exemption policy on some of the Phoronix benchmarks, and show the performance of the `NONSOCKET_RW_LEVEL` policy on the other suites. We used the largest available input sets for all benchmarks, and ran the multi-threaded benchmarks with four worker threads and used two replicas for all benchmarks. We excluded PARSEC's `canneal` benchmark from our measurements because it purposely causes data races that result in divergent behavior when running multiple replicas. This makes the benchmark incompatible with MVEEs. We also excluded SPLASH's `cholesky` benchmark due to incompatibilities with the version of the `gcc` compiler we used.

The results for these benchmarks are shown in Figures 3 and 4. The baseline overhead was measured by running ReMon with IP-MON and IK-B disabled. In this configuration, GHUMVEE runs as a standalone MVEE.

GHUMVEE generally performs well in these benchmarks. Our machine can run the replicas on disjoint CPU cores, which means that only the additional pressure on the memory subsystem and the MVEE itself cause performance degradation compared to the benchmarks' native performance. Yet, we still see the effect of enabling IP-MON. For PARSEC 2.1, the relative performance overhead decreases from 21.9% to 11.2%. For SPLASH-2x, the overhead decreases from 29.2% to 10.4%. In Phoronix, the overhead drops from 146.4% to 41.2%. Particularly interesting are the `dedup`, `water_spatial` and `network_loopback` benchmarks, which feature very high system call densities of over 60k system call invocations per second. In these benchmarks, the overheads drop from 252.9% to 69.4%, from 320% to 20.7%, and from

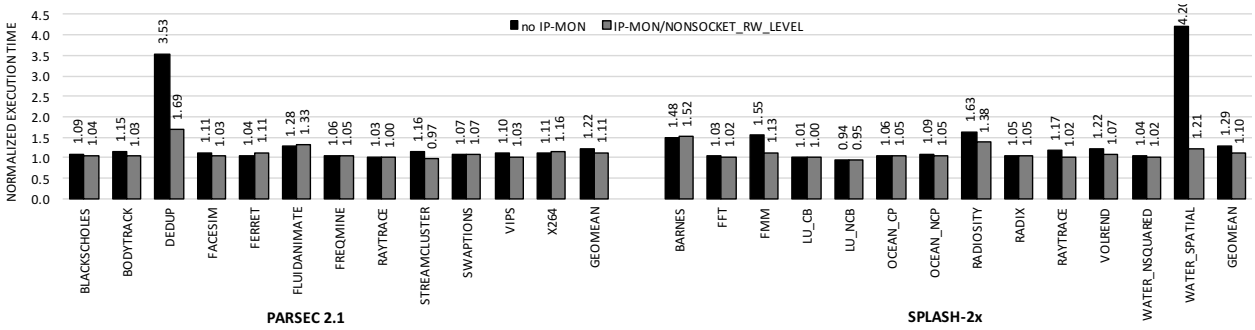

Figure 3: Performance overhead for two benchmark suites (2 replicas).

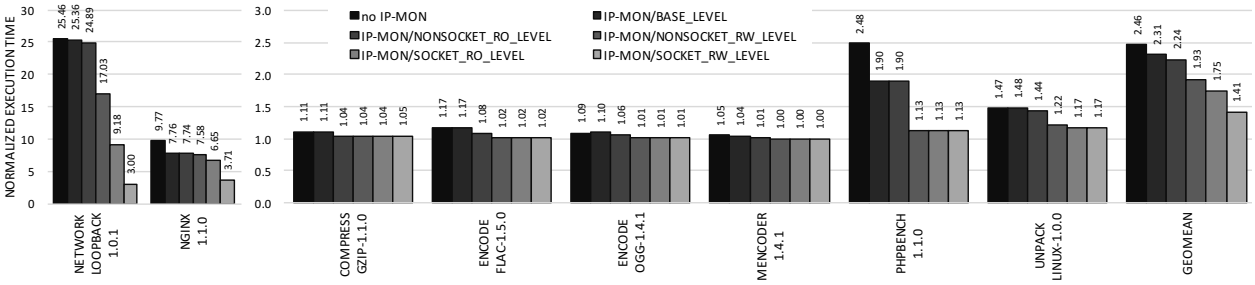

Figure 4: Comparison of IP-MON's spatial relaxation policies in a set of Phoronix benchmarks (2 replicas).

2446% to 200% respectively. Furthermore, the Phoronix results clearly show that different policies allow for different security-performance trade-offs.

5.2 Server Benchmarks

Server applications are great candidates for execution and monitoring by MVEEs because they (i) are frequently targeted by attackers and (ii) often run on many-core machines with idle CPU cores that can run replicas in parallel. In this section, we specifically evaluate our MVEE on applications used to evaluate other MVEEs. These applications include the `Apache` web server (used to evaluate Orchestra [35]), `thttpd (ab)` and `lighttpd (ab)` (used to evaluate Tachyon [26]), `lighttpd (http_load)` (used to evaluate Mx [16]), as well as `beanstalkd`, `lighttpd (wrk)`, `memcached`, `nginx (wrk)` and `redis` (used to evaluate VARAN [17]). We used the same client and server configurations described by the creators of those MVEEs.

We tested IP-MON by running a benchmark client on a separate machine that was connected to our server via a local gigabit link. We evaluated three scenarios. In the first scenario, we used the gigabit link as-is and therefore simulated an unlikely, worst-case scenario since the latency on the gigabit link was very low (less than 0.125ms). In the second scenario, we added a small amount of latency (bringing the total average latency to 2ms) to the gigabit link to simulate a realistic worst-case scenario (average network latencies in the US are 24–63ms [11]). In the third scenario, which we only evaluated to allow for comparison with existing MVEEs, we simulated a total average latency of 5ms. We used Linux' built-in `netem` driver to simulate the latency [24].

Figure 5 shows the worst-case and realistic scenarios side by side. For each benchmark, we measured the overhead IP-MON introduces when running between two and seven parallel replicas with the spatial exemption policy at the `SOCKET_RW_LEVEL`. We also show the overhead for running two replicas with IP-MON disabled. The latter case represents the best-case scenario without IP-MON.

5.3 Comparison with other MVEEs

Table 2 compares ReMon's performance with the results reported for other MVEEs in literature [16, 17, 26, 35, 40] and online [41]. As each MVEE was evaluated in a different experimental setup, the table also lists two features that have a significant impact on the performance overhead. These are the network latencies, because higher latencies hide server-side overhead, as well as the CPU cache sizes, as some of the memory-intensive SPEC benchmarks benefit significantly from larger caches, in particular with multiple concurrent replicas.

From a performance overhead perspective, the worst-case setup in which Mx and Tachyon were evaluated had the benchmark client running on the same (localhost) machine as the benchmark server. For VARAN two separate machines resided in the same rack and were hence connected by a very-low-latency gigabit Ethernet.

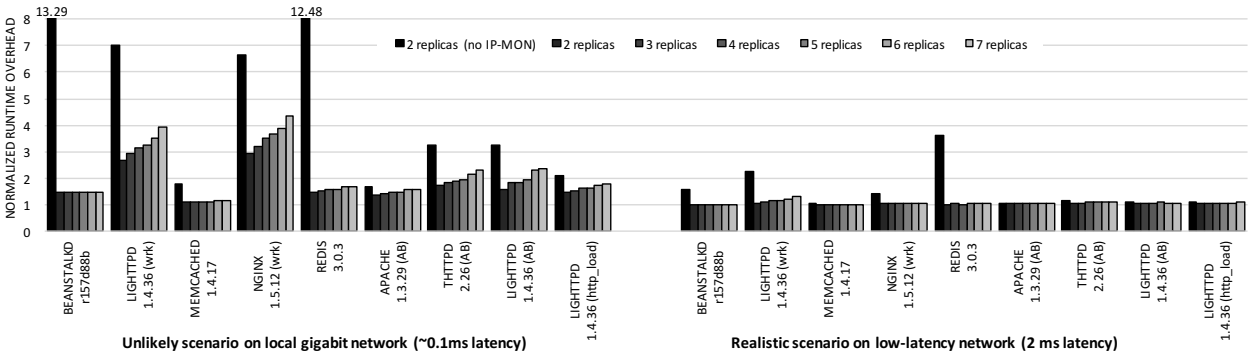

Figure 5: Server benchmarks in two network scenarios for 2–7 replicas with IP-MON and 2 replicas without IP-MON.

Orientation	Reliability						Security			
MVEE	Tachyon			Mx		VARAN	Orchestra	GHUMVEE	ReMon	
network	local-host	local few hops	coast-to-coast	local-host	USA-UK (150 ms)	same rack gigabit	local gigabit	n/a	local gigabit	local gigabit (5 ms)
CPU cache size				8 MB		8 MB		20 MB	20 MB	
reported overheads										
apache (ab)						**2.4%**	50%		34%	**2.4%**
lighttpd (ab)	790%	272%	30%			**0.0%**			55%	**0.0%**
thttpd (ab)	1320%	17%	**0%**			**0.0%**			73%	2.7%
lighttpd (httpld)				249%	4%	**1.0%**			45%	3.5%
redis				1572%	5%	6%			45%	**0.1%**
beanstalkd						52%			45%	**0.6%**
memcached						14%			**8.4%**	**0.3%**
nginx (wrk)						28%			194%	**0.8%**
lighttpd (wrk)						12%			169%	**0.7%**
SPEC CPU2006				17.9%				7.2%	**3.1%**	
SPECint 2006				17.6%		14.2%		12.1%	**3.9%**	
SPECfp 2006				18.3%				3.8%	**2.5%**	

Table 2: Comparison with other MVEEs (2 replicas).

The worst-case setups in which ReMon and Orchestra were evaluated consist of two separate machines connected by a low-latency gigabit link. In these unlikely, worst-case scenarios for servers, the differences in setups hence favor ReMon and Orchestra over VARAN, and VARAN over Tachyon and Mx.

In the best-case setups in which Mx and Tachyon were evaluated, one of the machines was located at the US west coast, while the other was located in England (Mx) or the US east coast (Tachyon). In ReMon's best-case setup, we used a gigabit link with a simulated 5 ms latency. So in the more realistic setups and for the server benchmarks, the differences favor Mx and Tachyon over ReMon.

This comparison demonstrates that ReMon outperforms existing non-hardware assisted security-oriented MVEEs while approaching the efficiency of reliability-oriented MVEEs.

6 Related Work

Directly intercepting system calls—known as **system call interposition**—to check if they are in line with a system call policy (often obtained through profiling and software analysis) predates MVEEs as a security sandboxing technique. The initial literature on the subject identified [15] the high overhead of `ptrace` on Linux (compared to similar techniques on other OSes), and kernel-based implementations were presented to overcome this overhead [31]. To reduce the impact on the kernel, ReMon performs most monitoring in-process, and requires only a small kernel patch to ensure its security.

Dune provides **in-process but across-privilege-ring monitoring** capabilities based on modern x86 hardware virtualization support such as VT-x and Extended Page Tables (EPT) [5]. Dune is, however, currently not thread-safe. This limits its practical applicability.

Cox et al. presented and evaluated an **IP kernel-space MVEE** implementation that deployed address-space partitioning as a diversification technique [12], which can be seen as a limited form of DCL [40]. They measured Apache latency increases of 18% on unsaturated servers, and throughput decreases of 48% on saturated servers, which exceed the corresponding overheads for ReMon.

Later **CP user-space** MVEEs, including the one by Bruschi et al. [8], Orchestra by Salamat et al. [35], and GHUMVEE [42] rely on, and suffer from, the properties of the `ptrace` and `waitpid` APIs. These MVEEs mainly differ from ReMon in the way they perform I/O replication. The Orchestra monitor executes I/O operations on behalf of the replicas, whereas most other MVEEs allow a designated master replica to execute I/O operations. Orchestra copies the results of I/O system calls to the replicas through a shared memory buffer, while Bruschi et al.'s MVEE uses `ptrace` to copy results. GHUMVEE initially relied on a custom `ptrace` implementation to copy data, but now uses the `process_vm_readv` API that was introduced in Linux 3.2.

VARAN takes this approach one step further, and also performs **IP user-space monitoring** [17] through shared ring buffers as shown in Figure 1(b) to avoid the overhead of `ptrace`. In VARAN, the direct master-slave communication is implemented by rewriting the system call instructions (incl. VDSO ones) in the binaries into trampolines to system call replication agents. The agents in the master replica execute the I/O system calls and

log them in the shared buffer. The agents in the slave replicas running behind the master then copy the results instead of executing the calls. Monitors embedded in replica processes check the system call consistency, and can even allow small discrepancies between the system calls behavior of the replicas. VARAN does not replicate user-space synchronization events, however, and hence cannot handle many typical client-side applications, most of which rely on user-space futexes.

With its support for small system call behavior discrepancies, as well as with some of its design and implementation options to minimize overhead, VARAN positions itself as a reliability-oriented MVEE that can support applications such as transparent failover, multi-revision execution (possibly to detect attacks, but not to prevent them), live sanitization, and record-replay [17]. With its in-process replication avoiding `ptrace`, VARAN significantly outperforms Tachyon [26] and Mx [16], two other reliability-oriented MVEEs.

As already noted by its authors, however, VARAN is less fit to protect against memory exploits. First, VARAN lets the master run ahead of the slaves, even for sensitive system calls, as it does not differentiate between sensitive and insensitive calls. This leaves a much larger window of opportunity to attackers than ReMon, including for the execution of sensitive calls. Although this window can be shortened by decreasing the size of VARAN's shared ring buffer, it is unclear what the impact on performance would be and whether that buffer adaptation closes the window completely or merely shortens it to one sensitive system call, which would clearly still be too much. Second, unlike the many protection techniques implemented for ReMon's IP-MON, VARAN's IP monitors are only protected from code-reuse attacks by ASLR, which has proven susceptible to attacks due to low entropy and granularity [3, 18, 37, 38]. This is all the more problematic as VARAN's IP monitors also monitor sensitive system calls. Finally, VARAN only rewrites explicit system call instructions in binary code into trampolines to its replication agents. ReMon, by contrast, intercepts all executed system calls, including any potential unaligned system call gadgets, which would not be identified by VARAN.

MvArmor leverages Dune's aforementioned hardware-assisted monitoring capabilties to offer secure in-process monitoring [20]. MvArmor's performance results are comparable to ReMon's, but due to limitations in Dune, it currently does not support multi-threaded replicas.

SFI [19, 27, 43, 44] and CFI [2, 1, 9] are two defenses that have received a lot of attention in literature which MVEEs can use to protect against memory exploits. Compared to MVEEs such as ReMon, they have the drawback of depending on relatively intrusive code transformations, most of which can only be applied when source code is available, and most of which, in particular those with stronger security guarantees, come with a significant performance penalty.

7 Conclusions

Designers of MVEEs face the mutually conflicting goals of security and runtime performance. Specifically, frequent interactions between cross-process MVEE monitors and program replicas require a high number of costly context switches. We demonstrate a best-of-both-worlds design, ReMon, in which an in-process monitor replicates inputs among the replicas and a cross-process monitor enforces lockstep execution of potentially harmful system calls; innocuous system calls, on the other hand, proceed without external monitoring to increase efficiency.

We present a careful and detailed security analysis and conclude that our introduction of an IP-MON component and relaxed monitoring of innocuous system calls still offers a level of security comparable to that of cross-process MVEEs. Our extensive performance evaluation shows that the overheads of ReMon ranges from 0-3.5% on realistic server workloads and compares very favorably to recent in-process MVEE designs.

Acknowledgments

The authors thank Brian Belleville, Haibo Chen, our reviewers, the Agency for Innovation by Science and Technology in Flanders (IWT), and the Fund for Scientific Research - Flanders.

This material is based upon work partially supported by the Defense Advanced Research Projects Agency (DARPA) under contracts FA8750-15-C-0124, FA8750-15-C-0085, and FA8750-10-C-0237, by the National Science Foundation under award number CNS-1513837 as well as gifts from Mozilla, Oracle, and Qualcomm.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), its Contracting Agents, or any other agency of the U.S. Government.

References

[1] ABADI, M., BUDIU, M., ERLINGSSON, ÚLFAR., AND LIGATTI, J. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security* (2005), ACM, pp. 340–353.

[2] ABADI, M., BUDIU, M., ERLINGSSON, ÚLFAR., AND LIGATTI, J. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC) 13* (2009), 4:1–4:40.

[3] BARRESI, A., RAZAVI, K., PAYER, M., AND GROSS, T. R. CAIN: Silently breaking ASLR in the cloud. In *USENIX Workshop on Offensive Technologies (WOOT)* (2015), WOOT'15.

[4] BASILE, C., KALBARCZYK, Z., AND IYER, R. A preemptive deterministic scheduling algorithm for multithreaded replicas. In *Proceedings of the 2002 IEEE International Conference on Dependable Systems and Networks (DSN'02)* (2002), pp. 149–158.

[5] BELAY, A., BITTAU, A., MASHTIZADEH, A., TEREI, D., MAZIÈRES, D., AND KOZYRAKIS, C. Dune: Safe user-level access to privileged CPU features. In *USENIX Symposium on Operating Systems Design and Implementation* (2012), OSDI '12, pp. 335–348.

[6] BERGAN, T., ANDERSON, O., DEVIETTI, J., CEZE, L., AND GROSSMAN, D. CoreDet: a compiler and runtime system for deterministic multithreaded execution. *ACM SIGARCH Computer Architecture News 38*, 1 (2010), 53–64.

[7] BERGER, E. D., AND ZORN, B. G. DieHard: probabilistic memory safety for unsafe languages. In *ACM SIGPLAN Notices* (2006), vol. 41, ACM, pp. 158–168.

[8] BRUSCHI, D., CAVALLARO, L., AND LANZI, A. Diversified process replicae for defeating memory error exploits. In *IEEE International Performance Computing and Communications Conference* (2007).

[9] BUDIU, M., ERLINGSSON, U., AND ABADI, M. Architectural support for software-based protection. In *Workshop on Architectural and System Support for Improving Software Dependability* (2006), ASID '06.

[10] CAVALLARO, L. *Comprehensive Memory Error Protection via Diversity and Taint-Tracking*. PhD thesis, PhD dissertation, Universita Degli Studi Di Milano, 2007.

[11] COMMISSION, F. C. Measuring broadband America - 2014. `https://www.fcc.gov/reports/measuring-broadband-america-2014`, 2014.

[12] COX, B., EVANS, D., FILIPI, A., ROWANHILL, J., HU, W., DAVIDSON, J., KNIGHT, J., NGUYEN-TUONG, A., AND HISER, J. N-variant systems: a secretless framework for security through diversity. In *USENIX Security Symposium* (2006), USENIX Association, p. 9.

[13] DEVIETTI, J., LUCIA, B., CEZE, L., AND OSKIN, M. DMP: deterministic shared memory multiprocessing. In *ACM SIGARCH Computer Architecture News* (2009), vol. 37, ACM, pp. 85–96.

[14] GARFINKEL, T., PFAFF, B., ROSENBLUM, M., ET AL. Ostia: A delegating architecture for secure system call interposition. In *NDSS '04* (2004).

[15] GOLDBERG, I., WAGNER, D., THOMAS, R., BREWER, E. A., ET AL. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography* (1996), vol. 6.

[16] HOSEK, P., AND CADAR, C. Safe software updates via multi-version execution. In *Proceedings of the 2013 International Conference on Software Engineering* (2013), IEEE Press, pp. 612–621.

[17] HOSEK, P., AND CADAR, C. VARAN the Unbelievable: An efficient n-version execution framework. In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2015), ACM, pp. 339–353.

[18] HUND, R., WILLEMS, C., AND HOLZ, T. Practical timing side channel attacks against kernel space ASLR. In *IEEE Symposium on Security and Privacy* (2013), S&P'13, pp. 191–205.

[19] HUNT, G. C., AND LARUS, J. R. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review 41*, 2 (2007), 37–49.

[20] KONING, K., BOS, H., AND GIUFFRIDA, C. Secure and efficient multi-variant execution using hardware-assisted process virtualization. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2016).

[21] LARSEN, P., HOMESCU, A., BRUNTHALER, S., AND FRANZ, M. SoK: Automated software diversity. In *Proceedings of the 35th IEEE Symposium on Security and Privacy* (2014), S&P '14.

[22] LEE, D., WESTER, B., VEERARAGHAVAN, K., NARAYANASAMY, S., CHEN, P. M., AND FLINN, J. Respec: efficient online multiprocessor replayvia speculation and external determinism. *ACM SIGARCH Computer Architecture News 38*, 1 (2010), 77–90.

[23] MAN-PAGES PROJECT, T. L. shmop(2) - linux manual page. `http://man7.org/linux/man-pages/man2/shmat.2.html`.

[24] MAN-PAGES PROJECT, T. L. tc-netem(8) - linux manual page. `http://man7.org/linux/man-pages/man8/tc-netem.8.html`.

[25] MANUAL PAGES, O. pledge - restrict system operations. `http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man2/pledge.2`.

[26] MAURER, M., AND BRUMLEY, D. Tachyon: Tandem execution for efficient live patch testing. In *USENIX Security Symposium* (2012), pp. 617–630.

[27] MCCAMANT, S., AND MORRISETT, G. Evaluating SFI for a CISC architecture. In *Usenix Security* (2006), p. 15.

[28] NAGARAKATTE, S., ZHAO, J., MARTIN, M. M., AND ZDANCEWIC, S. SoftBound: Highly compatible and complete spatial memory safety for C. In *ACM SIGPLAN Conference on Programming Language Design and Implementation* (2009), PLDI '09.

[29] NAGARAKATTE, S., ZHAO, J., MARTIN, M. M., AND ZDANCEWIC, S. CETS: Compiler enforced temporal safety for C. In *International Symposium on Memory Management* (2010), ISMM '10.

[30] OLSZEWSKI, M., ANSEL, J., AND AMARASINGHE, S. Kendo: efficient deterministic multithreading in software. *ACM Sigplan Notices 44*, 3 (2009), 97–108.

[31] PROVOS, N. Improving host security with system call policies. In *USENIX Security Symposium* (2002).

[32] PROVOS, N. Improving host security with system call policies. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12* (Berkeley, CA, USA, 2003), SSYM'03, USENIX Association, pp. 18–18.

[33] RONSSE, M., AND DE BOSSCHERE, K. RecPlay: a fully integrated practical record/replay system. *ACM Transactions on Computer Systems (TOCS) 17*, 2 (1999), 133–152.

[34] RUSSINOVICH, M., AND COGSWELL, B. Replay for concurrent non-deterministic shared-memory applications. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation (PLDI'96)* (1996), ACM.

[35] SALAMAT, B., JACKSON, T., GAL, A., AND FRANZ, M. Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space. In *Proceedings of the 4th ACM European conference on Computer systems* (2009), EuroSys'09, ACM, pp. 33–46.

[36] SEGULJA, C., AND ABDELRAHMAN, T. S. What is the cost of weak determinism? In *Proceedings of the 23rd international conference on Parallel architectures and compilation* (2014), ACM, pp. 99–112.

[37] SHACHAM, H., PAGE, M., PFAFF, B., GOH, E.-J., MODADUGU, N., AND BONEH, D. On the effectiveness of address-space randomization. In *ACM Conference on Computer and Communications Security* (2004), CCS '04.

[38] SIEBERT, J., OKHRAVI, H., AND SÖDERSTRÖM, E. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In *ACM Conference on Computer and Communications Security* (2014), CCS '14.

[39] SZEKERES, L., PAYER, M., WEI, T., AND SONG, D. SoK: Eternal war in memory. In *Proceedings of the 35th IEEE Symposium on Security and Privacy* (2013), S&P '13.

[40] VOLCKAERT, S., COPPENS, B., AND DE SUTTER, B. Cloning your gadgets: Complete ROP attack immunity with multi-variant execution. *IEEE Transactions on Dependable and Secure Computing PP*, 99 (2015).

[41] VOLCKAERT, S., AND DE SUTTER, B. GHUMVEE website. `http://ghumvee.elis.ugent.be`.

[42] VOLCKAERT, S., DE SUTTER, B., DE BAETS, T., AND DE BOSSCHERE, K. GHUMVEE: efficient, effective, and flexible replication. In *5th International Symposium on Foundations and practice of security (FPS 2012)* (2013), Springer, pp. 261–277.

[43] WAHBE, R., LUCCO, S., ANDERSON, T. E., AND GRAHAM, S. L. Efficient software-based fault isolation. In *ACM SIGOPS Operating Systems Review* (1994), vol. 27, ACM, pp. 203–216.

[44] YEE, B., SEHR, D., DARDYK, G., CHEN, J. B., MUTH, R., ORMANDY, T., OKASAKA, S., NARULA, N., AND FULLAGAR, N. Native Client: A sandbox for portable, untrusted x86 native code. In *IEEE Symposium on Security and Privacy* (2009), IEEE, pp. 79–93.