FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

MVX-systemen versnellen met moderne besturingssysteemextensies

Lennert Franssens
Studentennummer: 01702015

Promotoren: prof. dr. Bart Coppens, prof. dr. ir. Bjorn De Sutter
Begeleider: dr. ir. Bert Abrath

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Academiejaar 2021-2022

UNIVERSITEIT
GENT

Dankwoord

Met deze masterproef eindigt mijn opleiding industriële wetenschappen in de informatica. Met dit dankwoord zou ik graag een aantal mensen bedanken die een belangrijke rol hebben gespeeld tijdens mijn opleiding en bij het realiseren van deze masterproef.

In de eerste plaats wil ik mijn promotoren prof. dr. ir. Bjorn De Sutter en prof. dr. Bart Coppens bedanken. Ik ben jullie dankbaar dat ik mijn masterproef bij jullie heb mogen uitvoeren. Dankzij jullie leerde ik verder kritisch denken over de aanpak van mijn masterproef. Jullie stonden altijd voor me klaar als ik een vraag had en hebben ervoor gezorgd dat ik met tevredenheid, in een leuke sfeer deze masterproef heb kunnen afwerken.

Ook mijn begeleider dr. ir. Bart Abrath wil ik speciaal vermelden in dit dankwoord. U heeft me veel bijgeleerd hoe ik me kon inwerken in het grote project waarop mijn masterproef verdergaat. Ook u kon ik altijd contacteren als er problemen optraden. U heeft me geholpen om me in de juiste richting te sturen in het zoeken van oplossingen.

Naast de mensen die het dichtst bij me stonden bij het realiseren van deze masterproef, wil ik graag ook Willem Van Iseghem, Thomas Faingnaert en Waldo Verstraete bedanken van het Computer Systems Lab. Zij bezorgden me mee een fijne werkplek in het kantoor op het zevende verdiep van het iGent-gebouw.

Graag wil ik mijn lesgevers, en in het speciaal prof. dr. Helga Naessens, bedanken voor de uitleg en begeleiding tijdens mijn opleiding industriële wetenschappen in de informatica.

Daarnaast wil ik ook mijn medestudenten bedanken voor de leuke jaren die we als student samen beleefden. Dankzij jullie heb ik me helemaal kunnen ontwikkelen en heb ik het beste uit mezelf leren halen.

Als laatste wil ik mijn ouders, broer Jeffrey en vriendin Bettina bedanken. Bedankt om mij te blijven steunen gedurende mijn studies. Jullie zijn altijd in mij blijven geloven. Jullie hebben ervoor gezorgd dat ik mijn kansen maximaal heb leren en heb kunnen benutten.

Lennert Franssens, 9 juni 2022

Toelating tot bruikleen

"De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef."

Lennert Franssens, 9 juni 2022

MVX-systemen versnellen met moderne besturings-systeemextensies

door Lennert Franssens, studentennummer: 01702015

Promotoren: prof. dr. Bart Coppens, prof. dr. ir. Bjorn De Sutter
Begeleider: dr. ir. Bert Abrath

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Academiejaar 2021-2022

Samenvatting

Multi Variant Execution (MVX) systemen bieden een techniek voor het detecteren van veiligheidsproblemen met betrekking tot geheugencorruptie in software. Traditionele MVX-systemen zijn zeer veilig, maar werken relatief langzaam. Relaxed Monitoring (ReMon) is een ontwerp dat de uitvoering van MVX-systemen effectief versnelt. Een kernel-patch werd gebruikt om dat ontwerp te implementeren. Veel mensen geven er de voorkeur aan om geen kernelpatch op hun systeem toe te passen. Dat vermindert het gebruik van ReMon.

In dit proefschrift onderzoeken we of het gebruik van een kernelpatch bij de implementatie van het ReMon-ontwerp kan worden vervangen door moderne OS-extensies. Om de kernelpatch te vervangen maken we gebruik van seccomp-BPF filters. We passen het ontwerp aan en implementeren het. Door snelheidsmetingen te doen zien we een duidelijk potentieel voor het gebruik van het nieuw ontwerp. Verder onderzoek kan gedaan worden naar het verder uitdiepen van de ondersteunde functies en de daarbij horende veiligheid. Er kan ook nog verder onderzoek gedaan worden naar het gecombineerd gebruik van de nieuwe implementatie van ReMon met door ReMon te analyseren software dat ook gebruik maakt van de nieuwe seccomp-BPF technologie.

Trefwoorden

security, MVEE, MVX systems, Relaxed Monitoring, ReMon, modern OS extensions, seccomp-BPF

Boosting MVX Systems Through Modern OS Extensions

Lennert Franssens

Supervisors: prof. dr. ir. Bjorn De Sutter, prof. dr. Bart Coppens, dr. ir. Bert Abrath

***Abstract* — Multi Variant Execution (MVX) systems provide a technique for detecting security problems related to memory corruption in software. Traditional MVX systems are very secure but work relatively slowly. Relaxed Monitoring (ReMon) is a design that effectively speeds up the execution of MVX systems. A kernel patch was used to implement that design. Many people prefer not to apply a kernel patch to their system. This reduces the use of ReMon. In this dissertation we investigate whether the use of a kernel patch in the implementation of the ReMon design can be replaced by modern OS extensions. We will demonstrate that we can create a new design to replace the kernel patch. The preliminary implementation has potential to achieve similar speeds to the original implementation after further expansion.**

***Keywords* — security, MVEE, MVX systems, Relaxed Monitoring, ReMon, modern OS extensions, seccomp-BPF**

I. Introduction

MVX systems are used to detect memory-related vulnerabilities during a program execution. To discover such vulnerabilities, several variants of an application are executed simultaneously. The behavior of these variants is compared on the level of system calls by a monitor. A monitor is a ptrace process that manages the execution of different variants.

A ptrace process, the monitor in the case of an MVX system, introduces a large overhead during program execution. This is because a ptrace process does a few context switches per system call that it intercepts. To reduce this overhead, a new design was made. That design is Relaxed Monitoring or ReMon for short. A set of non-safety-sensitive system calls are not analyzed by the monitor using the ptrace process. Instead, they are analyzed by a monitor that does not use the ptrace technology and is therefore much faster. That faster monitor is mapped in the address space of the executing variant to reduce context switches.

The original ReMon design was implemented using a kernel patch for the in-kernel broker functions. These are used to choose which monitor should analyze a program's system calls. The intention of this dissertation is to replace the design of ReMon and the operation of the kernel patch by using modern OS extensions. It is important that a high level of security is maintained, and that the execution speed is on a par with the original ReMon design and its implementation.

The original design of ReMon is shown in Figure 1. When a variant makes a system call (1), the in-kernel-broker interceptor (IK-Broker interceptor) will intercept it. The IK-Broker interceptor is a component that lives in the kernel and thus is implemented through the kernel patch. The interceptor will choose what to do with the system call. If it is a security-sensitive system call, it is forwarded to the cross-process monitor (CP-MON) for further analysis (2a). CP-MON is a — slow but highly reliable — monitor based on a ptrace process.

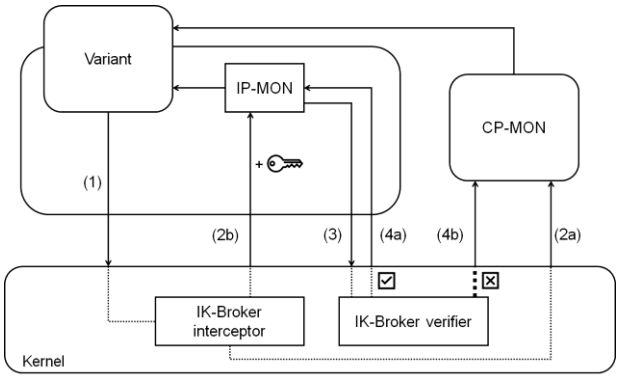


Figure 1. Original ReMon design

II. ReMon

When the IK-Broker interceptor decides that it is not a security-sensitive system call, it will generate a secret that is forwarded along with the system call to the in-process monitor (IP-MON) (2b). In IP-MON, some security checks are done. After that, IP-MON will try to execute the system call. That attempt is again intercepted by a component in the kernel (3), namely the in-kernel-broker verifier (IK-Broker verifier). The verifier will check, based on the secret obtained in step (2b), whether the system call was forwarded by the interceptor to IP-MON. In addition, it will also check if the system call still has the same properties as when it passed through the interceptor. If the secret matches, and the system call has the same properties as before, IP-MON may analyze the system call during its actual execution (4a). In all other cases, the system call will be analyzed by CP-MON (4b).

III. seccomp-BPF

seccomp-BPF is a technology in the Linux kernel that allows the delegation of the further course and or execution of system calls that are made by the process where seccomp-BPF is enabled and installed. To do this, it uses a programmable Berkeley Packet Filter (BPF). A Berkeley Packet Filter is a filter that makes decisions based on the number of a system call and its arguments. The following list summarizes some — for this dissertation interesting — decisions:

SECCOMP_RET_ALLOW — The system call can be executed as usual.

SECCOMP_RET_ERRNO — A 12-bit errno value is returned from the kernel without executing the system call.

SECCOMP_RET_TRACE — The kernel will notify the tracer process, which uses ptrace technology, of the program

being filtered even before the system call is executed. The tracer will delegate the remainder of the system call.

As presented in Figure 2, a program will make system calls such as read, write.... The seccomp-BPF mechanism will intercept these system calls and have them evaluated by the pre-programmed BPF filter. The BPF filter will determine the further course of the system call based on its statements. The filter explicitly describes which system calls will pass through directly and which will go to a ptrace process (tracer) for analysis by the signals SECCOMP_RET_ALLOW, and SECCOMP_RET_TRACE respectively.

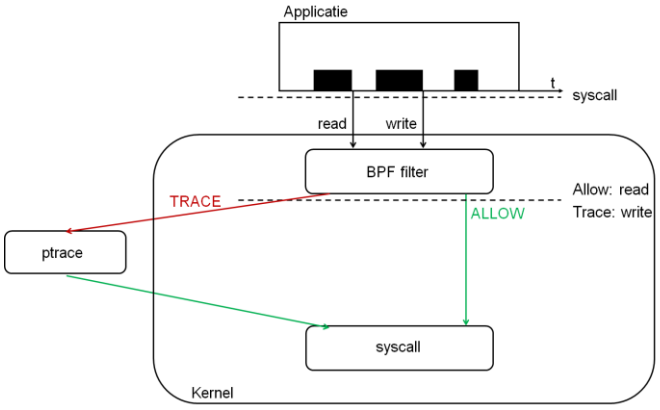


Figure 2. Flow of filtering system calls with seccomp-BPF

One minor drawback is that Berkeley Packet Filters cannot be used to dereference the arguments of a system call, such as strings. The filter will only be able to evaluate direct values, and not pointers.

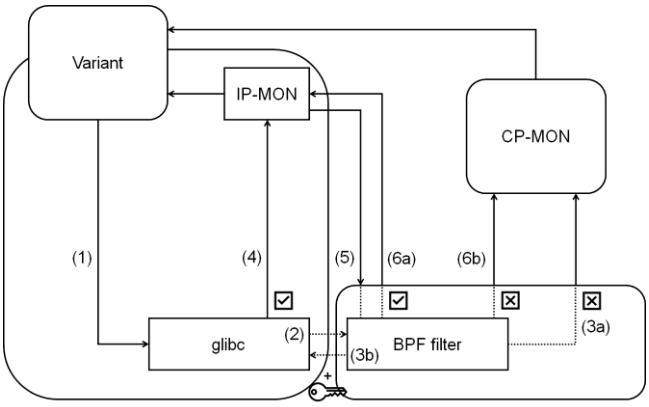


Figure 3. New ReMon design with seccomp-BPF

IV. New Design

We modify the MVEE, IP-MON and a modified version of glibc to replace the kernel patch. Figure 3 is the new design and shows the interaction between the aforementioned components.

System calls coming from a variant are converted to the correct form by glibc and then forwarded to the kernel (1). The first step is to catch the system call in glibc. In glibc, we can modify the function that performs the system call. Through those modifications, a two-stage system can be made.

The first step in that system is to execute the system call by calling the syscall instruction for the first time. That ensures that the system call must pass through the seccomp-BPF filter (2). That filter determines whether the system call should be executed by the cross-process monitor or the in-process monitor.

If the filter decides that it is a security-sensitive system call, it will forward the system call to CP-MON (3a). If the filter decides that it is a non-security-sensitive system call, it will send a secret back to glibc (3b). Through the secret, we indicate that we should execute the second step in the system in the custom glibc function. In addition, we use that secret to let glibc know the address of the in-process monitor. That way, glibc knows which address to jump to in order to go to IP-MON (4).

In IP-MON, we invoke the second step of the two-step system by executing the syscall instruction again (5). The seccomp-BPF filter will catch this call again and reevaluate it. If the filter sees that the system call is made from a specific, predefined address IP-MON and it is still a non-security sensitive system call, the system call will simply be executed and analyzed by IP-MON (6a).

System calls that do not come from IP-MON can never be executed and analyzed directly in IP-MON. CP-MON will analyze them instead. If the system call comes from the IP-MON but the arguments or the number of the system call is changed, it will also be forwarded to CP-MON (6b).

V. Implementation

To make the new design work, two crucial changes were made. The first change is in IP-MON itself. IP-MON — which is mapped in the address space of the variant that is being analyzed — enables and installs a seccomp-BPF filter directly into the variant. This filter is set up to work with the two-stage system from glibc. Listing 1 shows in pseudocode the operation of the filter. We see a filter that allows only harmless, non-security-sensitive system calls to be analyzed by IP-MON. The filter evaluates to a redirect to CP-MON in the case of a security-sensitive system call. The first if-else clause refers to the second stage from the two-stage system. The second if-else clause, in turn, refers to the first stage from the two-stage system introduced in glibc.

```
set bpf_allowed_syscalls = [__NR_getpid, ..., __NR_gettid]

if ipmon_syscall_address_ptr == seccomp_data.instruction_pointer do
   if seccomp_data.nr in bpf_allowed_syscalls do
      return SECCOMP_RET_ALLOW
   end else do
      return SECCOMP_RET_TRACE
   end
end else do
   if seccomp_data.nr in bpf_allowed_syscalls do
      return SECCOMP_RET_ERRNO
   end else do
      return SECCOMP_RET_TRACE
   end
end
```

Listing 1. Operation of the seccomp-BPF filter in pseudocode

Through the SECCOMP_RET_ERRNO action, we can pass secrets from the seccomp-BPF filter to glibc. That secret is the address that glibc must jump to in order to get into IP-MON.

The second change is in glibc. There we modify the system call macro to a custom system call function to be executed. Listing 2 gives the instructions contained in that function. Again, we see the two-stage system coming back. An initial system call is executed. When the seccomp-BPF filter

evaluates to SECCOMP_RET_TRACE, it will go to the end of the custom system call function and the system call will be evaluated by CP-MON. When the seccomp-BPF filter evaluates to SECCOMP_RET_ERRNO, which will only return a custom non-predefined errno value greater than 0x07FF in its first execution, the continuation of the custom system call function is executed. There, the address of IP-MON will be stored, then jumped to, after which the second stage in the two-stage system will be entered. The number of times we return an errno value may increase as we seek to transmit larger secrets.

```
    movq %rax,%r12
    syscall
    cmpq $-0x07FF,%rax
    jge glibc_custom_syscall_exit

    neg %rax
    shl $12,%rax
    movq %rax,%r11
    movq %r12,%rax
    call %r11

glibc_custom_syscall_exit:
    nop
```

Listing 2. Operations in the custom system call function in glibc

VI. Security Evaluation

Care must be taken with the modified version of glibc in conjunction with the seccomp-BPF filter. When a system call enters the filter for the first time, it is either redirected to IP-MON by returning a secret or it goes directly to CP-MON. This is a decision that should only happen if we are sure that the system call first passes through the modified and loaded into the variant version of glibc. If we don't impose restrictions on when the secret may be returned, the secret may leak. One possibility is that there is inline assembler written into the program, which will not pass through glibc but will pass through the seccomp-BPF filter to find out the secret.

To prevent potential abuse of that secret, the idea is to potentially redirect only system calls that pass through our modified glibc to IP-MON by only then sending the secret along. If it does not pass through our custom version of glibc, it will be sent to the cross-process monitor anyway. That can be implemented by mapping glibc on a known address. When a system call enters the seccomp-BPF filter for evaluation, we can compare the instruction pointer with a precalculated pointer of where our custom system call function is mapped.

VII. Performance Evaluation

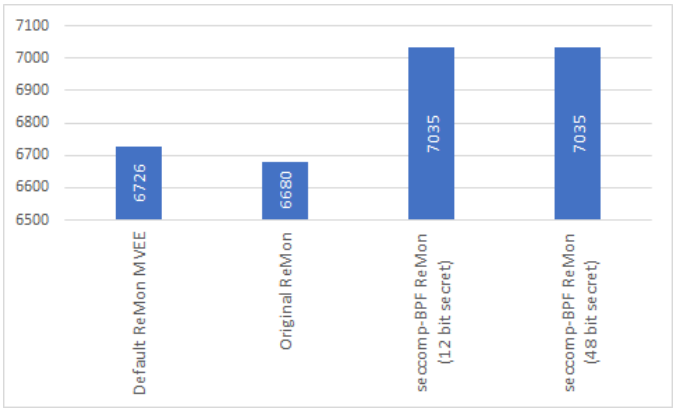
In the first measurement, we want to measure the overhead of evaluating a system call in a seccomp-BPF filter, and that relative to the execution time of the simplest possible system call that does not execute any other system calls behind it. Therefore, we choose to take measurements on a getpid system call. To calculate the execution time of a getpid system call as accurately as possible, we write a program that will execute the getpid system call repeatedly.

The measurements are performed on four versions. A default MVX system, ReMon that does not use the IP-MON component. A second version is the original version of ReMon that uses IP-MON. The third and fourth versions are versions of the new implementation with seccomp-BPF. A distinction is made in the size of the secret that is passed. In the version with a 12-bit secret, the secret is passed in 1 pass. In the 48-bit version, the filter is passed 4 times in order to pass the full secret.

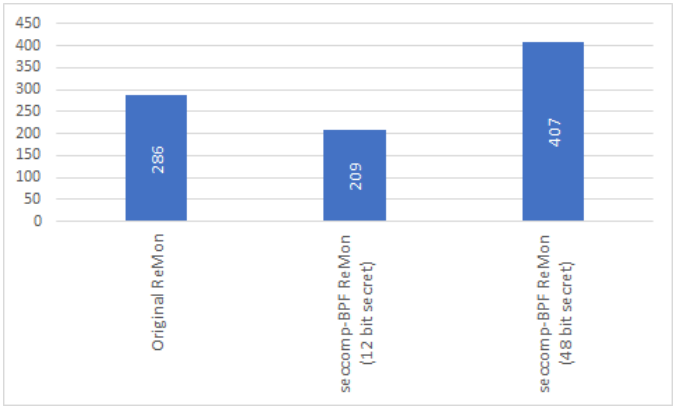
Measurements were performed for two cases: analysis of getpid in CP-MON and analysis of getpid in the faster IP-MON monitor.

In Figure 4 we see the results of the measurements. In Figure 4 (a) we can see that the new implementation introduces a small overhead by having to execute the filter and evaluate to a response through which the getpid system call is forwarded to CP-MON for analysis.

In Figure 4 (b) we see the results when a getpid is analyzed by IP-MON. In that the case we see speeds of the new implementation that are in the same order of magnitude as those of the original implementation. At a secret of 12 bits, the new implementation is even faster. The 48-bit version is more secure but has a larger overhead due to running the seccomp-BPF filter multiple times to obtain the full secret.



(a) getpid is analyzed by CP-MON [ns]



(b) getpid is analyzed by IP-MON [ns]

Figure 5. Time to execute one getpid system call

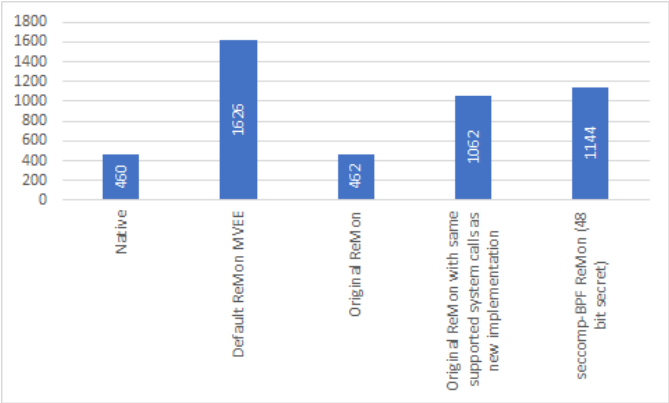
The second benchmark is the execution of a web server program nginx under the watchful eye of the MVX system. The benchmark was executed in five different ways. A native execution and an execution under the supervision of ReMon without using IP-MON. There is also an execution in the original implementation of ReMon. The latter two versions are both the new implementation of ReMon with seccomp-BPF and the original version of ReMon that uses IP-MON with the same set of supported system calls as those of the new implementation.

In that benchmark, the latency that is introduced by how the application is executed was measured. In addition, throughput

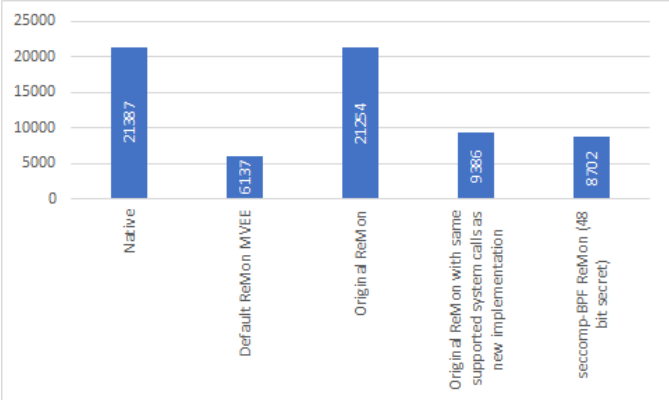
was also measured, which has an inverse relationship with latency.

We see that the native execution and the execution under the original implementation of ReMon with IP-MON achieve equal execution time, and thus latency and throughput. The default MVEE ReMon, which does not use IP-MON is clearly the slowest and has the highest latency and the lowest throughput.

The new implementation is substantially slower than the original implementation of ReMon but remains faster than the default MVEE version of ReMon. To verify that this is solely due to the limited set of supported system calls that can be analyzed in IP-MON in the new version, we also did the measurement for the old version with an equal set of supported system calls for IP-MON. We see that the speed of both versions are similar to each other. We can say that the extra latency is introduced by not yet supported system calls in the new version of ReMon with IP-MON. There is also another small overhead (~80 μs), which we can attribute to the use of a large secret that requires the seccomp-BPF filter to be executed multiple times.



(a) Latency [μs]



(b) Throughput [*requests/s*]

Figure 5. Web server (nginx) benchmark results

VIII. Drawbacks

The new implementation of ReMon has only a limited set of system calls that can be analyzed in IP-MON.

When multiple filters are installed, by performing clone or execve in the variant, the evaluation of the filter runs incorrectly. This is because all set filters will be executed. The first action with the highest priority that was evaluated in the set of filters is taken as the return value. Because of that feature, the evaluation potentially goes wrong because a difference in priority of return values is normally bridged by the two-stage system we have implemented. A solution to this is to map IP-MON to the same place in memory as the parent process after a clone or execve.

Multiple filters can also be installed by the program to be monitored itself. For example, modern browsers use seccomp-BPF filters. When such applications are monitored by ReMon, they set up new, additional seccomp-BPF filters, in the process where we installed our own filter. Again, this potentially leads to problems due to a difference in priority of the return value of the seccomp-BPF filter.

IX. Conclusion

The new design and implementation, which uses seccomp-BPF filtering, is promising. The new implementation achieves speeds comparable to the original implementation for the system calls that already have support for the new implementation. In addition to the speed aspect, efforts were also made to maintain a high safety aspect, which is also met. That by expanding the new implementation to support larger-sized secrets by returning multiple errno values as a secret. However, an overhead in speed must be taken into account, which grows linearly as the size of a secret grows.

The set of system calls that already are supported in the new implementation is not yet ready for similar performance as the original ReMon implementation. But there is potential in this new version, which can be said to be comparable in speed to the original implementation when we compare only the already supported system calls of the new implementation.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van listings

1

Inleiding

Context

Kwetsbaarheden in applicaties worden veel gebruikt om ongeoorloofd toegang te krijgen tot gegevens met een hoge waarde voor personen, bedrijven, overheden... Het basisprobleem bij de meeste kwetsbaarheden in applicaties is het gebruik van onveilige programmeertalen zoals C en C++. Deze programmeertalen worden als onveilig beschouwd omdat de ontwikkelaar zelf het geheugenbeheer kan doen en directe toegang heeft tot hardware. Dit soort programmeertalen wordt toch vaak gebruikt omdat een applicatie daarmee uitermate performant kan gemaakt worden.

Multi-Variante Uitvoering (MVU) is een techniek die gebruikt kan worden om software te beschermen tegen kwetsbaarheden met betrekking tot het geheugen. Die techniek is doeltreffend en werkt veilig maar relatief traag. Als antwoord daarop werd een nieuw ontwerp Monitoring Relaxation [1] ontworpen. Monitoring Relaxation zorgt voor een versnelde werking van MVU, zonder in te boeten op het veiligheidsaspect.

Probleemstelling

Relaxed Monitoring (ReMon) is een applicatie die het Monitoring Relaxation ontwerp implementeert. De implementatie maakt gebruik van een kernelpatch. Omdat het toepassen van zo'n kernelpatch lang duurt en specifieke toevoegingen doet aan de kernel die de gebruiker ook voor andere toepassingen gebruikt, is dat iets wat veel mensen liever niet doen. Dat beperkt dus ook het gebruik van ReMon.

Doelstelling

In deze thesis gaan we het ontwerp en de implementatie van Monitoring Relaxation aanpassen zodat de functionaliteit van de kernelpatch wordt vervangen door functionaliteit die gebruik maakt van nieuwe technologieën in de Linux kernel. Zo

wordt de kernelpatch overbodig. Het is belangrijk dat na die aanpassingen dezelfde uitvoeringssnelheden behaald kunnen worden als in de originele implementatie.

Opbouw van het verslag

In het tweede hoofdstuk wordt de technologieverkenning toegelicht. Hierin staat een beschrijving van de bestaande applicaties en technieken die gebruikt worden in deze thesis.

Het derde hoofdstuk wordt gebruikt om het nieuwe ontwerp voor te stellen. Ook bespreken we de implementatie, met het gebruik van nieuwe technologieën, in het tweede hoofdstuk.

In het vierde hoofdstuk wordt het oude ontwerp naast het nieuwe ontwerp gelegd. Er staat beschreven hoe en welke functionaliteiten overgenomen zijn en welke impact dat heeft op het veiligheidsaspect van ReMon.

Een vergelijking van de prestaties staat beschreven in het vijfde hoofdstuk van deze thesis. Het originele ontwerp en implementatie wordt vergeleken met het nieuwe ontwerp en implementatie.

Het zesde hoofdstuk wordt gebruikt om de tekortkomingen van het nieuwe ontwerp en implementatie uit te leggen.

In het laatste hoofdstuk staat de conclusie van deze thesis beschreven. We bespreken welke SDGs[2] van toepassing zijn op dit werk, welke problemen er zijn opgetreden, welke lessen er geleerd zijn uit deze thesis en welk verder onderzoek er nog kan gedaan worden naar het toevoegen van nieuwe besturingssysteemextensies in MVX-systemen.

2

Technologieverkenning

In dit hoofdstuk wordt dieper ingegaan op een MVUO, of Multi-Variante Uitvoeringsomgeving, waarin nieuwe technologieën geïmplementeerd moeten worden. Daarnaast wordt ook de functionaliteit van de nieuwe technologieën seccomp-BPF[3] en Syscall User Dispatch[4] in dit hoofdstuk beschreven.

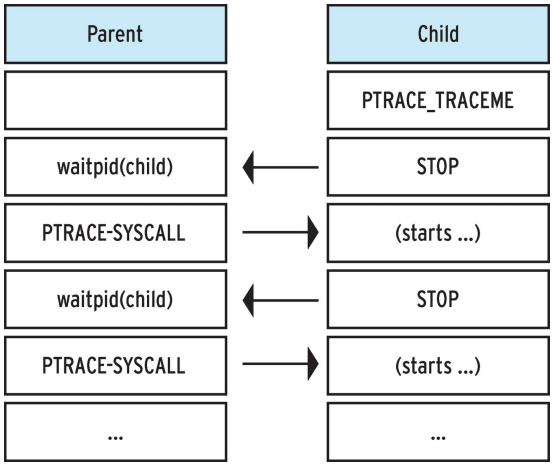
2.1 ptrace

Een proces kan in Linux een ander proces debuggen door gebruik te maken van de ptrace technologie[5] die terug te vinden is in de Linux kernel. Deze technologie laat het met andere woorden toe om systeemaanroepen van processen te traceren en onderscheppen. Door ptrace kunnen de stappen van een proces gevolgd worden vanuit het standpunt van de kernel.

Figuur 2.1 geeft de interactie weer tussen een ouder- en kindproces waarbij de ouder de tracer is en het kind de tracee. Na een fork() zal het kindproces het PTRACE_TRACEME-commando uitvoeren, gevolgd door een stopcommando. Het ouderproces zal wachten tot het kindproces door de kernel wordt gestopt door waitpid(pid, &status, 0) uit te voeren. Het ouderproces zal daarna ptrace(PTRACE_SYSCALL, pid, 0, 0) uitvoeren om de volgende systeemaanroep van het kindproces te onderscheppen en de uitvoering te pauzeren. Het ouderproces zal de onderschepte systeemaanroep onderzoeken. De systeemaanroep wordt na het onderzoek gewoon uitgevoerd waarna deze stappen herhaald worden voor de volgende systeemaanroep van het kindproces.

2.2 MVUO

De uitbuiting van kwetsbaarheden in applicaties moet voorkomen worden. Om dat probleem aan te pakken werden al veel technieken uitgevonden, zoals ASLR[6]. Dit soort technieken zijn echter beperkt in het type kwetsbaarheid waartegen ze werken en kunnen nog steeds omzeild worden. Daarom werd een MVUO bedacht. Dit is een omgeving om applicaties beschermen tegen kwetsbaarheden met betrekking tot het geheugen.



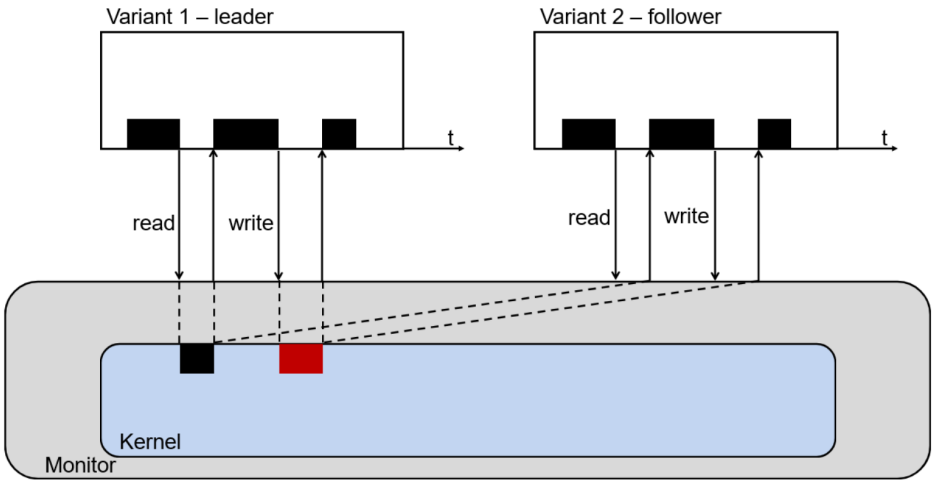
Figuur 2.1: Interactie tussen ptrace-proces (tracer) en tracee

Figuur 2.2: Originele werking monitor met PTRACE_SYSCALL

Een MVUO voert meerdere instanties van een programma gelijktijdig uit. Die varianten doen systeemaanroepen naar de kernel. Die systeemaanroepen zijn een verzoek van de varianten om welbpeaalde taken uit te voeren zoals het verkrijgen van de procesidentificatie of het uitschrijven van karakters naar een uitvoerkanaal. In de MVUO zit een cross-process monitor, wat eigenlijk een ptrace-proces is, die een laag vormt tussen de verschillende varianten en de kernel. De cross-process monitor onderschept systeemaanroepen van alle verianten en kan op die manier het gedrag van die varianten vergelijken. Er wordt voor elke variant een monitor voorzien. Na het uitvoeren van een clone-aanroep, zal een nieuwe variant gestart worden onder het toezicht van een nieuwe monitor die het verdere verloop van die variant ook zal monitoren.

De cross-process monitor (tracer) gebruikt een PTRACE_SYSCALL actie om een variant (tracee) te pauzeren bij een syscall-entry-stop event, dat plaatsvindt net voor de uitvoering van een systeemaanroep door de kernel. De monitor gaat op dat moment naar een syscall handler om de systeemaanroep voor zijn uitvoering te analyseren. In de handler wordt daarna opnieuw een PTRACE_SYSCALL actie gebruikt om net na de uitvoering van een systeemaanroep door de kernel, wanneer een syscall-exit-stop event plaatsvindt, te stoppen. Na de uitvoering van de rest van de handler gebruikt de monitor opnieuw de PTRACE_SYSCALL actie om te pauzeren na het volgende syscall-entry-stop event. De werking is vereenvoudigd gevisualiseerd in Figuur 2.2. In Bijlage 1 staat een gedetailleerde visualisatie van die werking.

De monitor in ReMon gebruikt voor elke systeemaanroep in een variant twee opeenvolgende PTRACE_SYSCALL acties. Er kan gezegd worden dat de monitor de staat bijhoudt van de variant door gebruik te maken van het een dubbele PTRACE_SYSCALL actie per systeemaanroep. Tussen de twee PTRACE_SYSCALL acties per systeemaanroep voert de monitor een analyse voor de uitvoering van de systeemaanroep uit. Na de tweede PTRACE_SYSCALL actie voert de monitor een analyse na de uitvoering van de systeemaanroep uit.



Figuur 2.3: Blokdiagram uitvoering GHUMVEE met verschillende varianten

Het tweemaal pauzeren van een variant per systeemaanroep zorgt ervoor dat er over de hele uitvoering van een variant heen veel contextswitches gedaan worden.

2.2.1 GHUMVEE

Binnen de onderzoeksgroep Computer Systems Lab aan de Universiteit Gent is zo'n MVUO gemaakt, namelijk GHUMVEE[7] (GHent University MultiVariant Execution Environment). Zoals bij de meeste MVUOs, monitort en vergelijkt die het gedrag van de verschillende varianten door tussen te komen bij elke systeemaanroep. Daarvoor zal de monitor elke variant even stopzetten en de systeemaanroepen apart uitvoeren. Potentieel gevaarlijke systeemaanroepen worden enkel uitgevoerd als ze consistent zijn over alle varianten heen, wat lockstep wordt genoemd. De monitor wacht tot alle varianten op hetzelfde uitgangspunt zijn gekomen om het gedrag dan te vergelijken en de uitvoering van elke variant verder te laten gaan.

Zoals bij andere MVUOs wordt bij GHUMVEE een leader/follower model gebruikt. Eén variant is de leader en alle andere varianten zijn followers. Systeemaanroepen die effect hebben op de rest van het besturingssysteem, zoals I/O-operaties, worden enkel door de leader variant uitgevoerd. Op het moment van zo'n systeemaanroep zal de monitor dit detecteren. De leader zal zijn uitvoering gewoon verderzetten, maar de followers niet. Hun systeemaanroepnummer wordt eerst aangepast naar het nummer van een systeemaanroep dat geen effect heeft op het besturingssysteem. Bij het terugkeren van de systeemaanroep naar de applicatie moet de juiste waarde ook bij de followers teruggegeven worden. De followers krijgen daarvoor een gekopieerde waarde van de terugkeerwaarde van de leader in hun adresruimte.

Een nadeel is dat GHUMVEE een grote runtime overhead heeft. Dat komt door het lockstep-synchronisatiemodel voor sys-

teemaanroepen, de contextswitches en omdat beveiliging belangrijker is dan de snelheid van uitvoering.

2.3 ReMon

Om het nadeel van GHUMVEE aan te pakken, is er een MVUO met een nieuw design bedacht, namelijk ReMon[1]. Dat staat voor Relaxed Monitoring en zal de runtime overhead verkleinen terwijl het veiligheidsniveau hoog blijft. De basiscomponenten die verder besproken worden staan in Figuur 2.4.

Veel systeemaanroepen hebben geen effect op de werking buiten het proces waarin de systeemaanroep gebeurt. Daarom moet slechts een kleine subset van veiligheidsgevoelige systeemaanroep volledig gemonitord worden in de cross-process monitor. Alle andere systeemaanroepen worden niet aan de cross-process monitor voorgelegd, maar wel aan een in-process monitor. De in-process monitor zit mee in de adresruimte van de variant. Daardoor worden er geen onnodige contextswitches uitgevoerd, waardoor die monitor aan een veel sneller tempo de uitvoering zal analyseren.

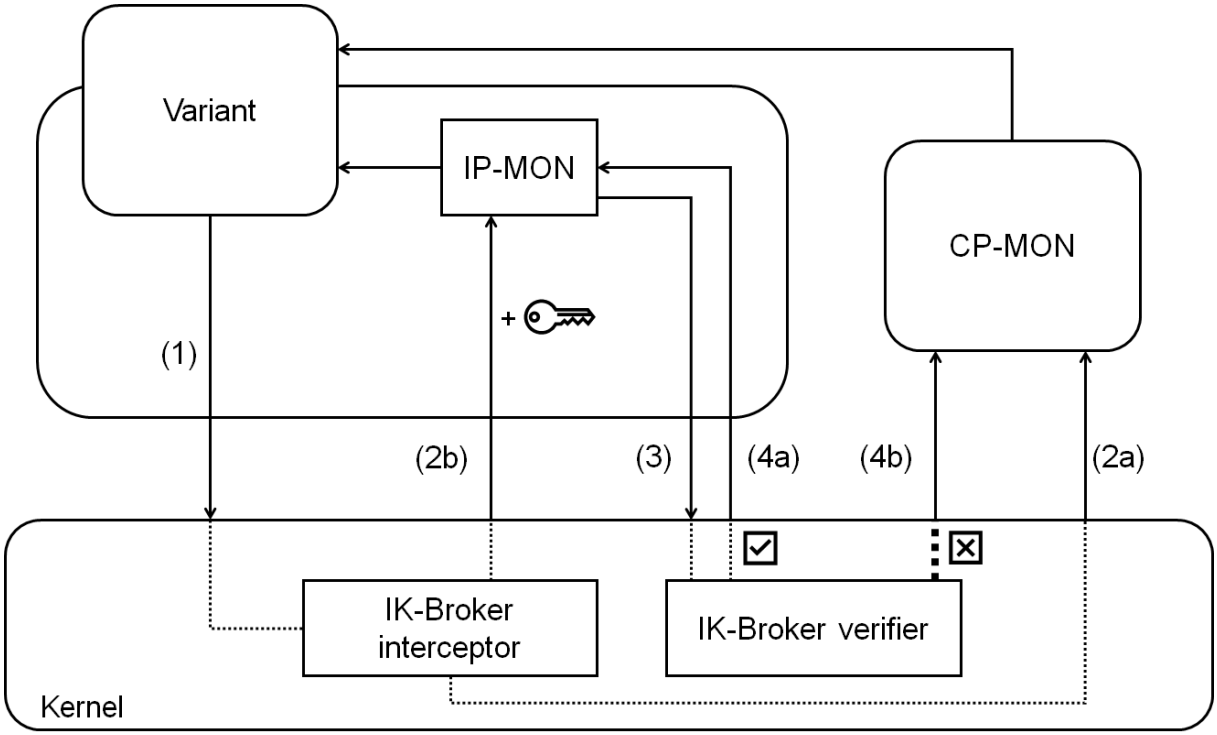
Bij ReMon controleert de cross-process monitor of de in-process monitor het gedrag van de systeemaanroep moet analyseren, afhankelijk van het systeemaanroepnummer. Om die beslissing te maken zal in de kernel een stukje code zitten, de in-kernel broker (IK-Broker) interceptor. Dit stukje zal systeemaanroepen onderscheppen die van een variant komen en hun verdere verloop delegeren.

De in-kernel broker stuurt veiligheidsgevoelige systeemaanroepen door naar de cross-process monitor GHUMVEE. Die zal alle varianten even on hold zetten, de systeemaanroepen uitvoeren en hun gedrag vergelijken. Daarna kunnen de varianten hun uitvoering verderzetten.

In het geval dat de in-kernel broker beslist dat het niet om een veiligheidsgevoelige systeemaanroep gaat, zal die de systeemaanroep niet naar de cross-process monitor doorsturen, maar wel naar een in-process monitor. Wanneer een systeemaanroep naar de in-process monitor gaat, zal de programmateller aangepast worden naar een systeemaanroep beginpunt in de code van de in-process monitor. De broker zal op dat moment ook een sleutel meegeven.

De in-process monitor gebruikt ook een leader/follower-model. Enkel de leader zal de systeemaanroep effectief uitvoeren. De followers wachten tot de leader klaar is met zijn systeemaanroep en die het resultaat van de systeemaanroep gekopieerd heeft in de Replication Buffer. De Replication Buffer repliceert daarna de teruggeefwaarde naar de followers. De Replication Buffer moet voor het veiligheidsaspect van de MVEE op een veilige, onbekende plaats in het geheugen zitten. Om dat te garanderen, zal de kernel het adres van de Replication Buffer slechts op één plaats bijhouden.

In de in-process monitor vinden bij het binnenkomen van een systeemaanroep enkele veiligheidschecks plaats waarna de systeemaanroep opnieuw uitgevoerd wordt. Bij deze stap wordt er terug naar de kernel gegaan. Niet de interceptor broker wordt aangesproken, maar wel een nieuwe component, namelijk de in-kernel broker (IK-Broker). Die zal de sleutel uit de interceptor verifiëren. Als de sleutel juist is, mag de teruggeefwaarde naar de in-process monitor gestuurd worden. Als de sleutel niet juist is, er een andere systeemaanroep uitgevoerd werd of de systeemaanroep vanuit de in-process monitor komt,



Figuur 2.4: Basiscomponenten ReMon

wordt de originele sleutel als ongeldig gezet. De systeemaanroep gaat daardoor naar de cross-process monitor GHUMVEE voor verdere analyse.

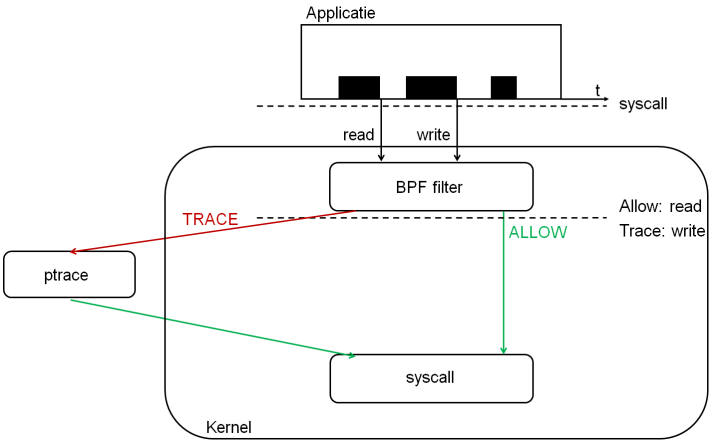
2.4 seccomp-BPF

seccomp-BPF[3] is een technologie in de Linux kernel die het mogelijk maakt om het verdere verloop en of de uitvoering van systeemaanroepen te delegeren. Het maakt daarvoor gebruik van een programmeerbare Berkeley Packet Filter (BPF). Dat is een filter die beslissingen maakt aan de hand van het nummer van een systeemaanroep en zijn argumenten .

De seccomp-BPF filter wordt in een proces in userspace geprogrammeerd en wordt uitgevoerd tijdens de hele levensloop van het proces, inclusief zijn kindprocessen. De programmatie van de filter geeft aan welke actie uitgevoerd moet worden bij het optreden van een welbepaalde systeemaanroep. Onderstaande lijst geeft een overzicht van enkele mogelijke acties:

SECCOMP_RET_TRAP Er wordt een SIGSYS-signaal vanuit de kernel teruggestuurd zonder de systeemaanroep uit te voeren. De programmateller zal aangepast worden zodat het lijkt alsof de systeemaanroep uitgevoerd is.

SECCOMP_RET_TRACE De kernel zal het tracer-proces, dat gebruik maakt van de ptrace-technologie, van het programma dat wordt gefilterd, verwittigen nog voor de systeemaanroep wordt uitgevoerd. De tracer zal het verdere verloop van de systeemaanroep delegeren.



Figuur 2.5: Blokdiagram filteren met seccomp-BPF filter

SECCOMP_RET_ALLOW De systeemaanroep mag gewoon uitgevoerd worden.

2.4.1 Gebruik met ptrace

De actie SECCOMP_RET_TRACE[8] duidt erop dat BPF-filters standaard ondersteuning hebben om samen te werken met de ptrace-technologie. De ptrace-technologie wordt ook gebruikt in ReMon (Sectie 2.3) als cross-process monitor.

Zoals voorgesteld in Figuur 2.5 zal een programma systeemaanroepen doen zoals read, write... Het seccomp-BPF-mechanisme zal deze systeemaanroepen onderscheppen en laten evalueren door de voorgeprogrammeerde BPF-filter. De BPF-filter zal aan de hand van zijn statements het verdere verloop van de systeemaanroep bepalen. In de filter staat expliciet beschreven welke systeemaanroepen rechtstreeks doorgelaten worden en welke naar een ptrace-proces (tracer) zullen gaan voor analyse, door respectievelijk de signalen SECCOMP_RET_ALLOW en SECCOMP_RET_TRACE.

In Listing 1 staat een voorbeeld van een BPF-filter die hoort bij Figuur 2.5. Deze filter staat in het te analyseren proces (tracee) geprogrammeerd. Afhankelijk van de uitkomst van een jump-instructie wordt een statement of het vervolg van de filter uitgevoerd. Als de filter uit het voorbeeld doorlopen is, zal de systeemaanroep op traditionele wijze uitvoeren.

```
struct sock_filter filter[] = {
        BPF_STMT(BPF_LD+BPF_W+BPF_ABS, offsetof(struct seccomp_data, nr)),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, __NR_read, 0, 1),
        BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW),
        BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, __NR_write, 0, 1),
        BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_TRACE),
        BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW),
};
```

Listing 1: Code van een BPF-filter

Het proces waarop de seccomp-BPF filter werkt, stelt zelf de filter in aan de hand van een systeemaanroep.

De code van de tracer staat in Listing 2. Eerst wordt een ptrace-optie ingesteld om verwittigingen van de BPF-filter, PTRACE_O_TRACESECCOMP, te ontvangen. Telkens de waarde SECCOMP_RET_TRACE in de BPF-filter optreedt zal de tracer verwittigd worden. De tracer zal na analyse van een systeemaanroep de uitvoering van het kindproces (tracee) verder laten gaan.

```
ptrace(PTRACE_SETOPTIONS, pid, 0, PTRACE_O_TRACESECCOMP);
while (1)
{
        ptrace(PTRACE_CONT, pid, 0, 0);
        waitpid(pid, &status, 0);
        if (WSTOPSIG(status) == SIGTRAP) {
                long syscall_no = ptrace(PTRACE_PEEKUSER, pid, 8 * ORIG_RAX, NULL);
                printf("syscall number: %ld\n", syscall_no);
        }
}
```

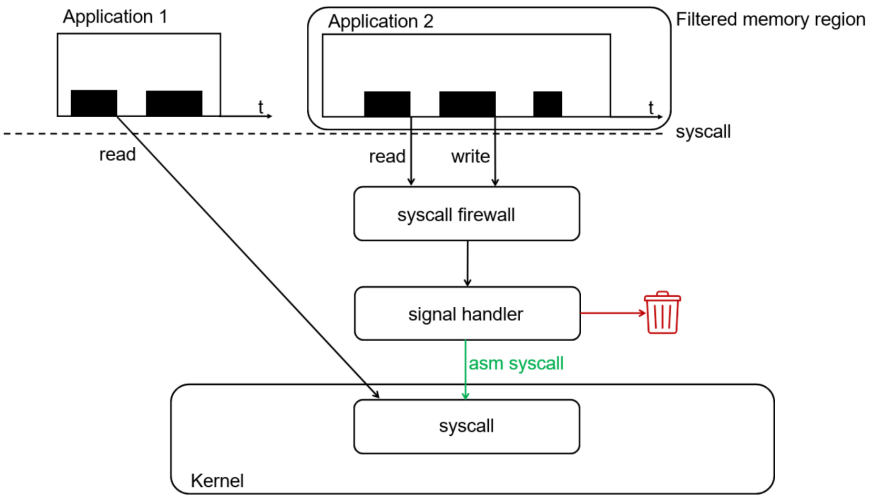
Listing 2: Tracer code voor tracee met BPF-filter

2.4.2 Tekortkomingen

Berkeley Packet Filters kunnen niet gebruikt worden om de argumenten van een systeemaanroep te derefereren, zoals strings. De filter zal enkel directe waarden, en geen pointers, kunnen evalueren. Systeemaanroepen waarbij argumenten die pointers zijn kritiek zijn voor de evaluatie, kunnen doorgestuurd worden naar een ptrace-proces voor verdere analyse.

2.5 Syscall User Dispatch

Een andere technologie in de Linux kernel is SUD[4] (Syscall User Dispatch). Deze technologie is ontworpen voor het emuleren van Windows code in Wine op Linux[9]. Zoals te zien is in Figuur 2.6 kan deze technologie de systeemaanroepen uit een bepaalde regio in de geheugenruimte filteren. Die regio heet de filtered memory region. Telkens een systeemaanroep gebeurt, zal de SUD-technologie die onderscheppen en een signaal uitsturen. Een signal handler zal dat signaal opvangen en kan daarna meer informatie verkrijgen over de systeemaanroep door de context op te vragen. Deze informatie bevat bijvoorbeeld het systeemaanroepnummer en de argumenten. Het al dan niet uitvoeren van de systeemaanroep is afhankelijk van een programma in de signal handler.



Figuur 2.6: Blokdiagram filteren met SUD

2.5.1 Tekortkomingen

Syscall User Dispatch is specifiek ontwikkeld voor het gebruik in Wine op Linux. Deze technologie werkt niet goed samen met seccomp-BPF en ptrace. seccomp-BPF en ptrace werden gemaakt voor het analyseren van systeemaanroepen en zullen ook alle systeemaanroepen uit de signal handler analyseren, hoewel dat niet altijd nodig is.

3

Ontwerp en implementatie

Hoofdstuk 3 gaat over het nieuwe ontwerp en de implementatie ervan in ReMon. Het nieuwe ontwerp lijkt op het originele design omdat vooral de implementatie van IP-MON wijzigt om enkel de kernelpatch te vervangen. Dat gebeurt door gebruik te maken van een seccomp-BPF filter.

3.1 Ontwerp

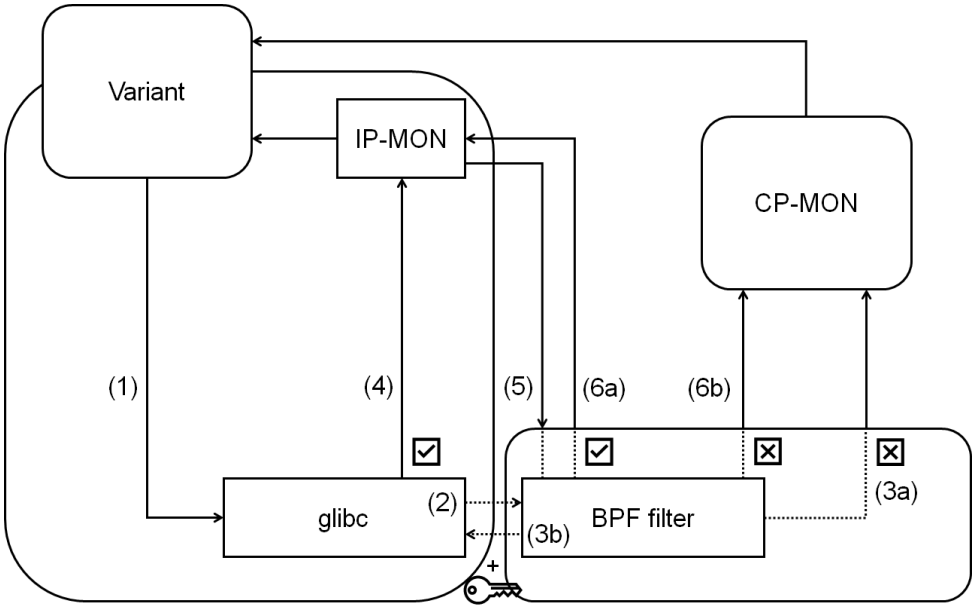
We passen de MVEE, IP-MON en een aangepast versie van glibc[10] aan om de kernelpatch te vervangen. Figuur 3.1 is het nieuwe ontwerp en geeft de interactie tussen voorgenoemde componenten weer.

Systeemaanroepen die vanuit de variant komen, worden door glibc omgezet naar de juiste vorm om vervolgens naar de kernel doorgestuurd te worden (1). De eerste stap is het opvangen van de systeemaanroep in glibc. In glibc kunnen we de functie die de systeemaanroep uitvoert aanpassen. Via die aanpassingen kan een tweetrapssysteem gemaakt worden.

De eerste stap daarin is om de systeemaanroep uit te voeren door de syscall-instructie een eerste keer op te roepen vanuit glibc. Dat zorgt ervoor dat de systeemaanroep via de seccomp-BPF filter moet passeren (2). Die filter bepaalt of de systeemaanroep door de cross-process monitor of de in-process monitor moet uitgevoerd worden.

Wanneer de filter beslist dat het om een veiligheidsgevoelige systeemaanroep gaat, zal die de systeemaanroep naar de cross-process monitor doorsturen (3a). In het geval dat de filter beslist dat het niet om een veiligheidsgevoelige systeemaanroep gaat, zal de filter een geheim terugsturen naar glibc (3b). Via het geheim geven we aan dat we de tweede stap van de aangepaste glibc functie moeten uitvoeren. Daarnaast gebruiken we dat geheim om aan glibc het adres van de in-process monitor te laten weten. Op die manier weet glibc naar welk adres het moet springen om naar de in-process monitor te gaan (4).

In de in-process monitor voeren we de systeemaanroep opnieuw uit (5). De seccomp-BPF filter zal deze aanroep opnieuw opvangen en opnieuw evalueren. Als de filter ziet dat de systeemaanroep vanop een specifiek, voorgedefinieerd adres in de in-process monitor gebeurt en het nog steeds om een niet veiligheidsgevoelige systeemaanroep gaat, zal de systeemaanroep



Figuur 3.1: Basiscomponenten nieuw ontwerp ReMon

gewoon uitgevoerd en geanalyseerd worden door de in-process monitor (6a).

Systeemaanroepen die niet vanuit de in-process monitor komen, kunnen nooit direct in de in-process monitor uitgevoerd worden.

In het geval dat de systeemaanroep vanuit de in-process monitor komt maar de argumenten of het nummer van de systeemaanroep gewijzigd is, wordt deze alsnog naar de cross-process monitor doorgestuurd (6b).

3.1.1 seccomp-BPF

In Listing 3 staat de pseudocode van de seccomp-BPF filter die aangemaakt moet worden. Deze filter zorgt ervoor dat de in-process monitor enkel ongevaarlijke systeemaanroepen zal evalueren. Daarvoor wordt een tweetrapssysteem gemaakt.

De eerste keer dat een syscall-instructie onderschept wordt, wat de eerste trap in het systeem is, zal de conditie "ipmon_syscall_address_ptr == seccomp_data.instruction_pointer"onwaar zijn. In de else-clause evalueert de seccomp-BPF filter het systeemaanroepnummer. Er zijn twee gevallen:

- Als het nummer in de lijst van ongevaarlijke systeemaanroepen (bpf_allowed_syscalls) zit, moet die naar IP-MON doorgestuurd worden. Om naar IP-MON tet kunnen springen, wordt een errno-waarde, die het adres van de IP-MON systeemaanroepcomponent bevat, teruggestuurd naar de caller van de syscall-instructie. In dit geval is de caller een aangepaste systeemaanroepfunctie in glibc.
- Als het nummer niet in de lijst van ongevaarlijke systeemaanroepen zit, wordt de cross-process monitor verwittigd

door een een TRACE-actie uit te sturen vanuit de seccomp-BPF filter.

Enkel wanneer een errno-waarde werd teruggestuurd, kan IP-MON instaan voor een versnelde evaluatie van de systeemaanroep. Er wordt vanuit glibc naar de IP-MON systeemaanroepcomponent gesprongen. Daar wordt de systeemaanroep nogmaals uitgevoerd, de tweede trap in het systeem, die de seccomp-BPF filter opnieuw zal onderscheppen. De seccomp-BPF filter kent het adres van de syscall-instructie uit IP-MON en komt in de eerste if-clause terecht aangezien "ipmon_syscall_address_ptr == seccomp_data.instruction_pointer"waar zal zijn. In de if-clause voert de seccomp-BPF filter opnieuw een test aan de hand van het systeemaanroepnummer. Er zijn opnieuw twee gevallen:

- Als het nummer in de lijst van ongevaarlijke systeemaanroepen (bpf_allowed_syscalls) zit, wordt een TRACE-actie uitgestuurd waardoor de systeemaanroep zonder te tracen uitgevoerd wordt.
- Als het nummer niet in de lijst van ongevaarlijke systeemaanroepen zit, wordt een TRACE-actie uitgestuurd waardoor de cross-process monitor verwittigd wordt.

```
set bpf_allowed_syscalls = [__NR_getpid, ..., __NR_gettid]

if ipmon_syscall_address_ptr == seccomp_data.instruction_pointer do
    if seccomp_data.nr in bpf_allowed_syscalls do
        return SECCOMP_RET_ALLOW
    end else do
        return SECCOMP_RET_TRACE
    end
end else do
    if seccomp_data.nr in bpf_allowed_syscalls do
        return SECCOMP_RET_ERRNO
    end else do
        return SECCOMP_RET_TRACE
    end
end
```

Listing 3: Pseudocode van de seccomp-BPF filter in IP-MON

3.1.2 Veiligheidsaspect

De seccomp-BPF filter is de enige component, naast IP-MON, die op de hoogte is van het adres van de IP-MON systeemaanroepcomponent. De filter kan op een indirecte manier dat adres aan glibc laten weten, op het moment dat de spronginstructie

moet gebeuren, waardoor externe actoren geen directe verwijzing kunnen vinden naar de IP-MON systeemaanroepcomponent.

Het geheimhouden van het adres van die component is cruciaal. Wanneer aanvallers toch naar dat adres kunnen springen, kunnen ze ongeoorloofd gebruik maken van de minder veilige monitoring van IP-MON omdat die bijvoorbeeld geen argumenten kan vergelijken en analyseren. Om dat probleem tegen te gaan, wordt in de in-kernel broker een uniek geheim bijgehouden per systeemaanroep voor validatie bij de daadwerkelijke uitvoering van de systeemaanroep. Als dat geheim niet overeenkomt met een eerder uitgedeeld geheim van een nog niet afgehandelde systeemaanroep, zal de systeemaanroep naar de cross-process monitor worden doorgestuurd. Op die manier wordt de kans kleiner dat een aanvaller ongeoorloofd een systeemaanroep vanuit IP-MON kan uitvoeren omdat hij naast het adres van IP-MON, ook een tijdelijk uniek geheim moet te weten zien te komen.

3.2 Implementatie

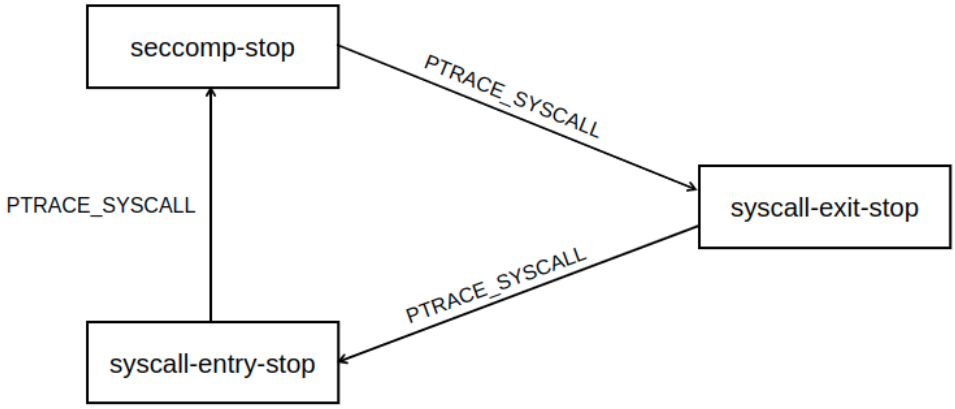
In de volgende paragrafen staat beschreven hoe we ReMon hebben aangepast om het nieuwe ontwerp te implementeren. Er zijn enkele uitdagingen die opgelost worden om het gebruik van seccomp-BPF filters te laten werken in ReMon.

3.2.1 MVEE

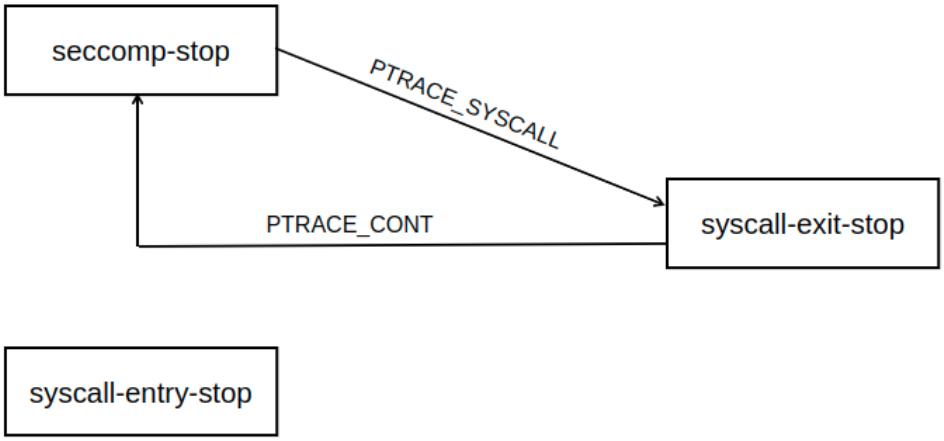
Gedrag van de monitor

Wanneer een seccomp-BPF filter gebruikt wordt, en die een TRACE-actie uitstuurt, treedt een bijkomstig event op na het syscall-entry-stop event en voor het syscall-exit-stop event. Dat event is het seccomp-stop[8] event. Door gebruik te maken van een PTRACE_SYSCALL actie, zal een monitor (tracer) ook bij dat event de variant (tracee) pauzeren. Dat impliceert dat de monitor de staat van de variant niet meer kan bijhouden door gebruik te maken van een dubbele PTRACE_SYSCALL actie per systeemaanroep. De werking is vereenvoudigd gevisualiseerd in Figuur 3.2. In Bijlage 2 staat een gedetailleerde visualisatie van die werking. Wanneer de monitor op deze manier werkt, zullen er fouten optreden omdat de staat per twee PTRACE_SYSCALL acties per systeemaanroep wordt bijgehouden, maar er in realiteit drie PTRACE_SYSCALL acties per systeemaanroep nodig zijn.

Om de huidige implementatie van de monitor niet te veel aan te passen, is het handig om een dubbele actie per systeemaanroep te behouden. Het syscall-entry-stop event en het seccomp-stop event vinden beide plaats voor de uitvoering van een systeemaanroep. De informatie die bij het afhandelen van beide events beschikbaar is en gebruikt wordt in de monitor is identiek. Daarom kan het afhandelen van het syscall-entry-stop event overgeslagen worden. In praktijk wordt daarvoor een PTRACE_CONT[5] actie gebruikt. Deze actie stopt wel bij het optreden een seccomp-stop event en niet bij een syscall-entry-stop event. De werking is vereenvoudigd gevisualiseerd in Figuur 3.3. In Bijlage 3 staat een gedetailleerde visualisatie van die werking.



Figuur 3.2: Werking monitor met PTRACE_SYSCALL en seccomp-stop event



Figuur 3.3: Nieuwe werking monitor met PTRACE_SYSCALL en PTRACE_CONT

Van zodra de IP-MON component in een variant opgestart is, zal de seccomp-BPF filter ervoor zorgen dat het bijkomend seccomp-stop event optreedt in de monitor (tracer). Zoals hiervoor beschreven staat, zal de monitor op dat moment moeten schakelen van een dubbele PTRACE_SYSCALL actie per systeemaanroep naar een PTRACE_CONT gevolgd door een PTRACE_SYSCALL actie per systeemaanroep. Hiervoor wordt een vlag per variant bijgehouden in de monitor. Voor het zetten van deze vlag, wordt de dubbele PTRACE_SYSCALL actie gebruikt aangezien de monitor op dat moment nog alle systeemaanroepen moet kunnen analyseren. Na het inschakelen van IP-MON in de variant, wordt de vlag gezet wat ervoor zorgt dat er geschakeld wordt naar een PTRACE_CONT gevolgd door een PTRACE_SYSCALL actie. Op die manier wordt op het juiste moment rekening gehouden met het bijkomstig seccomp-stop event.

Staat van de monitor na een clone-aanroep

Om het juiste gedrag van de monitor na een clone-aanroep te garanderen, moet de monitor dezelfde staat hebben als zijn oudermonitor. Op die manier zal de monitor van in het begin van zijn levensloop in een juiste synchronisatie tussen de verschillende systeemaanroepen werken, door af te wisselen tussen PTRACE_SYSCALL en PTRACE_CONT acties.

3.2.2 Glibc

Een aangepaste versie van glibc[10] wordt als systeembibliotheek in het geheugen geladen bij de opstart van een variant van de MVEE. Elke systeemaanroep die dan door de variant wordt uitgevoerd, passeert langs de glibc wrapper functie voor een systeemaanroep (syscall). De wrapper functie voor zo'n systeemaanroep is zodanig aangepast dat een systeemaanroep die IP-MON moet observeren, omgeleid wordt naar de in-process monitor van de variant.

Zoals in Sectie 3.1.1 beschreven staat, zal een errno-waarde gebruikt worden om aan te geven waar in het geheugen de IP-MON component geladen zit om systeemaanroepen te observeren. IP-MON zal in het geheugen gemapt worden op een plaats die met de 16-bits errno-waarde beschreven kan worden. Aangezien de errno-waarde ook een echte waarde kan zijn bij het optreden van een fout, moet de errno-waarde die het adres van de IP-MON systeemaanroepcomponent beschrijft, groter zijn dan de maximaal gebruikte waarde[11] als voorgedefinieerde errno-waarde.

In Listing 4 staat de code van een aangepaste systeemaanroepfunctie die in de glibc systeemaanroep wrapper aangeroepen wordt. Er wordt eerst een systeemaanroep uitgevoerd. Wanneer deze geen errno-waarde terugkrijgt uit de seccomp-BPF filter, wil dat zeggen dat de cross-process monitor de systeemaanroep zal analyseren en er geen bijkomstige instructies uitgevoerd worden. Wanneer de seccomp-BPF filter wel een errno-waarde terugstuurt, kan die waarde in glibc opgehaald worden. Daarna vergelijkt de code of het om een echte errno-waarde of het adres van de IP-MON systeemaanroepcomponent gaat. In het eerste geval, gebeuren er geen bijkomstige instructies. In het tweede geval, gaat een call-instructie gebeuren naar de IP-MON systeemaanroepcomponent die de systeemaanroep verder zal analyseren.

```
movq %rax,%r12
syscall
```

```
    cmpq $-0x07FF,%rax
    jge glibc_custom_syscall_exit

    neg %rax
    shl $12,%rax
    movq %rax,%r11
    movq %r12,%rax
    call %r11

glibc_custom_syscall_exit:
    nop
```

Listing 4: Aangepaste systeemaanroep-functie in glibc

3.2.3 IP-MON

Tijdens het opstarten van de MVEE zit de IP-MON component nog niet in het geheugen van de varianten. De IP-MON component stuurt tijdens het opstarten van de MVEE een valse systeemaanroep door. Zo'n valse systeemaanroep wordt door de monitor in de ReMon applicatie gebruikt om tussen componenten te communiceren[1] en zal ook nooit echt doorgelaten worden naar de kernel. ReMon kan valse systeemaanroepen van echte onderscheiden en handelt ze op een speciale manier af.

De valse systeemaanroep vanuit IP-MON zorgt ervoor dat de MVEE weet dat IP-MON geactiveerd zal worden. De MVEE stelt in op welk adres de IP-MON component in elke variant wordt ingeladen en houdt een vlag bij die de monitor verwittigt dat seccomp-BPF actief is.

Na het uitvoeren van die valse systeemaanroep zal de IP-MON component deel zijn van het proces van de variant. Omdat IP-MON in het proces van de variant geladen zit, kan die component de seccomp-BPF filter in het proces laden.

3.2.4 seccomp-BPF

De concrete implementatie van de pseudocode die beschreven staat in Sectie 3.1.1 is weergegeven in Listing 5. Het BPF_STMT op instructielijn 21 in Listing 5 zorgt voor het terugkeren van het adres van de syscall-instructie in IP-MON naar glibc. Aangezien IP-MON de seccomp-BPF filter in het proces laadt, kent die component het adres van de syscall-instructie in IP-MON. Het adres wordt teruggestuurd via een errno-code.

De filter bevat tweemaal dezelfde sectie. De tweede sectie (instructielijnen 13-20) wordt tijdens de eerste systeemaanroep uitgevoerd. Wanneer die evalueert naar het terugsturen van een errno-code met het het adres van de IP-MON systeemaan-

roepcomponent, zal de filter tijdens de volgende systeemaanroep (die dan vanuit de IP-MON component komt) de eerste sectie (instructielijnen 3-10) uitvoeren. Die twee secties zullen naar exact hetzelfde resultaat evalueren, behalve het feit dat bij de tweede sectie de SECCOMP_RET_ERRNO actie vervangen is door een SECCOMP_RET_ALLOW actie. Als de systeemaanroep tussen de eerste en de tweede evaluatie zou wijzigen naar een veiligheidsgevoelige systeemaanroep, wordt die toch nog door de cross-process monitor geanalyseerd.

```
/* [0] Load the instruction pointer from 'seccomp_data' buffer into accumulator */
BPF_STMT(BPF_LD | BPF_W | BPF_ABS,
    (offsetof(struct seccomp_data, instruction_pointer))),
/* [1][A] Jump forward C-A-1 instructions if instruction pointer does not match
the ipmon syscall address. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K,
    (unsigned int)variants[variantnum].syscall_address_ptr, 0, (unsigned char)(C-A-1)),
/* [2] Load system call number from 'seccomp_data' buffer into accumulator. */
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
/* [3-9] Jump forward 0 instructions if system call number does not match '__NR_XXX'. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getpid, 7, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getegid, 6, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_geteuid, 5, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getgid, 4, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getpgrp, 3, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getppid, 2, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_gettid, 1, 0),
/* [10][B] Jump forward D-B-1 instructions if system call number does not match '__NR_XXX'.
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getuid, 0, (unsigned char)(D-B-1)),
/* [11] Execute the system call */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
/* [12][C] Load system call number from 'seccomp_data' buffer into accumulator. */
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
/* [13-19] Jump forward 0 instructions if system call number does not match '__NR_XXX'. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getpid, 7, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getegid, 6, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_geteuid, 5, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getgid, 4, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getpgrp, 3, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getppid, 2, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_gettid, 1, 0),
/* [20] Jump forward 1 instructions if system call number does not match '__NR_XXX'. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getuid, 0, 1),
```

```
/* [21] Return the ipmon syscall entry address */
BPF_STMT(BPF_RET | BPF_K,
    SECCOMP_RET_ERRNO | (ipmon_syscall_entry_address_ptr & SECCOMP_RET_DATA))
/* [22][D] Execute the system call in tracer */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_TRACE),
```

Listing 5: Implementatie van de seccomp-BPF filter in IP-MON

3.3 Implementatie met grotere geheimen

Om gebruik te kunnen maken van grotere geheimen in de nieuwe implementatie, zullen we de seccomp-BPF meerdere errno-waardes laten terugsturen. De glibc en seccomp-BPF componenten zullen een alternatieve implementatie krijgen om dat te verwezelijken. De implementatie met grotere geheimen is een uitbreiding op de implementatie van het nieuwe ontwerp. Fundamenteel zullen de componenten dezelfde flow volgen. Hier en daar worden aanpassingen gedaan om geheimen van 48 of meer door te kunnen sturen. Ook deze implementatie kan verder uitgebreid worden, op gelijke wijze als we ze nu beperkt zullen uitbreiden, om grotere of meerdere geheimen mogelijk te maken.

3.3.1 Glibc

Glibc maakt het tweetrapssysteem mogelijk waarbij na de eerste trap potentieel een geheim kan ontvangen worden, het adres van de IP-MON systeemaanroepcomponent. Dat geheim zit vervat in een errno-waarde die onze seccomp-BPF terugstuurt bij het evalueren naar een SECCOMP_RET_ERRNO actie.

We kunnen de seccomp-BPF filter meerdere keren laten uitvoeren om telkens een nieuw deel van een geheim door te sturen, per 12 bits. Om dat mogelijk te maken, moeten we vanuit glibc, die het geheim ontvangt en nadien gebruikt, iets ondernemen zodat de seccomp-BPF filter meerdere keren uitgevoerd zal worden.

Daarom zullen we meerdere systeemaanroepen uitsturen, met een systeemaanroepnummer dat niet voorgedefinieerd is voor een gelidge systeemaanroep. Dat systeemaanroepnummer is een waarde die de seccomp-BPF filter zal terugsturen wanneer de seccomp-BPF filter voor de eerste keer naar een SECCOMP_RET_ERRNO actie zal evalueren per systeemaanroep die naar IP-MON moet gaan. Dat systeemaanroepnummer wordt geïncrementeerd wanneer volgende delen van het geheim opgevraagd moeten worden. Zoals in Sectie3.2.2 beschreven staat, moet de eerste errno-waarde uit de seccomp-BPF filter ook groter zijn dan de maximaal gebruikte waarde[11] als voorgedefinieerde errno-waarde.

In Listing 6 staat de code van de aangepaste systeeemaanroepfunctie die een groter geheim opvraagt en ontvangt. Er wordt eerst een systeemaanroep uitgevoerd. Wanneer deze geen errno-waarde terugkrijgt uit de seccomp-BPF filter, wil dat zeggen dat de cross-process monitor de systeemaanroep zal analyseren en er geen bijkomstige instructies uitgevoerd worden.

Wanneer de seccomp-BPF filter wel een errno-waarde terugstuurt, kan die waarde in glibc opgehaald worden. Daarna vergelijkt de code of het om een echte errno-waarde of het startsignaal, en tevens het te gebruiken systeemaanroepnummer voor het opvragen en ontvangen van het grotere geheim dat het adres van de IP-MON systeemaanroepcomponent, bevat. In het eerste geval, gebeuren er geen bijkomstige instructies. In het tweede geval, wordt een nieuwe syscall-instructie uitgevoerd met als systeemaanroepnummer de verkregen errno-code. Vervolgens zal de filter een deel van het geheim doorsturen. Daarna wordt het gebruikte systeemaanroepnummer geïncrementeerd worden, waarna opnieuw het volgende deel van te filter wordt opgevraagd. Dat wordt herhaald tot de gewenste lengte van het geheim is bereikt. Daarna zal glibc het originele systeemaanroepnummer herstellen en naar de IP-MON systeemaanroepcomponent kunnen springen. IP-MON kan de systeemaanroep verder zal analyseren.

```
    movq %rax,%r12                  // save original system call number
    syscall                         // invoke system call for the first time
    cmpq $-0x07FF,%rax              // check if errno-value is a predefined one
    jge glibc_custom_syscall_exit   // jump to exit if errno-value is a "real" one

    neg %rax                        // get start key of secret exchange
    movq %rax,%r13                  // store key of secret exchange

    syscall                         // invoke system call to get a part of the secret
    neg %rax                        // get value that represents a part of the secret
    shl $12,%rax                    // shift address 12 bits to left to align with enclave
    movq %rax,%r15                  // store start of the secret in a register
    add $1,%r13                     // increment key of secret exchange
    movq %r13,%rax                  // set new key of secret exchange as system call number

    ...                             // perform some more iterations

    syscall                         // invoke system call to get a part of the secret
    neg %rax                        // get value that represents a part of the secret
    shl $XX,%rax                    // shift partial secret XX bits to left
    addq %rax,%r15                  // add partial secret to other parts of the secret

    movq %r12,%rax                  // restore original system call number
    call %r15                       // call ipmon enclave

glibc_custom_syscall_exit:
    nop
```

Listing 6: Aangepaste systeemaanroep-functie in glibc met ondersteuning voor grote geheimen

3.3.2 seccomp-BPF

De tweede component die we aanpassen om grotere geheimen mogelijk te maken in de nieuwe implementatie, is de seccomp-BPF filter die door IP-MON wordt ingesteld. De concrete implementatie van de pseudocode die beschreven staat in Sectie 3.1.1 is weergegeven in Listing 5. De pseudocode houdt geen rekening met beperkingen van een errno-waarde. Die praktische beperking, dat een errno-waarde in een seccomp-BPF filter maximaal 12 bits groot is, zorgt ervoor dat we meerdere errno-waardes moeten versturen om een groot geheim door te bekomen. Zoals besproken in Sectie3.3.1, roept glibc meerdere keren een systeemaanroep aan om de filter meerdere keren uit te laten voeren.

We zien in Listing 7 dat de filter enkele nieuwe statements en jumps bevat ten opzichte van de implementatie die is weergegeven in Listing 5. Die nieuwe statements en jumps staan in voor de overdracht van het geheim. Er wordt op voorhand een random waarde in invoke_key_exchange gestoken waarvan de waarde met zijn maximale incrementatie in glibc niet kan overlappen met andere random gegenereerde waarden bij het gebruik van verschillende adressen voor de mapping IP-MON. Die waarde wordt teruggestuurd als errno-waarde wanneer we in het tweetrapssysteem zien dat bij de eerste trap een systeemaanroep voorbijkomt die naar IP-MON mag gaan. In glibc wordt die waarde gebruikt om alle delen van het geheim op te halen, zoals ook in Sectie3.3.1 beschreven staat.

In de seccomp-BPF filter worden verschillende statements met bijhorende jumps voorzien om elk deel van het geheim door te sturen, afhankelijk van hoeveel keer de invoke_key_exchange variabele in glibc geïncrementeerd is.

```
/* [0] Load the instruction pointer from 'seccomp_data' buffer into accumulator */
BPF_STMT(BPF_LD | BPF_W | BPF_ABS,
    (offsetof(struct seccomp_data, instruction_pointer))),
/* [1][A] Jump forward to the next if instruction pointer does not match the ipmon
   checked syscall instruction address. If it matches, we need to trace (E-A-1) */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K,
    ((__u32*)&ipmon_checked_syscall_instr_ptr)[0], (unsigned char)(E-A-1), 0),
/* [2][B] Jump forward C-A-1 instructions if instruction pointer does not match the
   ipmon unchecked syscall instruction address. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K,
    ((__u32*)&ipmon_unchecked_syscall_instr_ptr)[0], 0, (unsigned char)(D-B-1)),
/* [3] Load system call number from 'seccomp_data' buffer into accumulator. */
BPF_STMT(BPF_LD | BPF_W | BPF_ABS,
    (offsetof(struct seccomp_data, nr))),
/* [4-5] Jump forward 0 instructions if system call number does not match '__NR_XXX'. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getpid, 2, 0),
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_sched_yield, 1, 0),
/* [6][C] Jump forward E-C-1 instructions if system call number does not match
   '__NR_XXX'. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_futex, 0, (unsigned char)(E-C-1)),
```

```
/* [7] Execute the system call */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
/* [8][D] Load system call number from 'seccomp_data' buffer into accumulator. */
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
/* [9] Jump forward 1 instructions if system call number does not match XXX. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, invoke_key_exchange, 0, 1),
/* [10] Return the ipmon syscall entry address bits 12 - 23 */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO |
    ((unsigned int)ipmon_enclave_entrypoint_ptr_bits_12_23 & SECCOMP_RET_DATA)),
/* [11] Jump forward 1 instructions if system call number does not match XXX. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, invoke_key_exchange + 1, 0, 1),
/* [12] Return the ipmon syscall entry address bits 24 - 35 */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO |
    ((unsigned int)ipmon_enclave_entrypoint_ptr_bits_24_35 & SECCOMP_RET_DATA)),
/* [13] Jump forward 1 instructions if system call number does not match XXX. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, invoke_key_exchange + 2, 0, 1),
/* [14] Return the ipmon syscall entry address bits 36 - 47 */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO |
    ((unsigned int)ipmon_enclave_entrypoint_ptr_bits_36_47 & SECCOMP_RET_DATA)),
/* [15] Jump forward 1 instructions if system call number does not match '__NR_XXX'. */
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_getpid, 0, 1),
/* [16] Return the ipmon syscall entry address */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO |
    (invoke_key_exchange & SECCOMP_RET_DATA)),
/* [17][E] Execute the system call in tracer */
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_TRACE),
```

Listing 7: Implementatie van de seccomp-BPF filter in IP-MON met ondersteuning voor grote geheimen, hier 48 bits waarvan 12 bits als startsignaal en 36 bits die het effectieve adres van de IP-MON systeemaanroepcomponent volledig kunnen beschrijven door een alignatie van die component op de 12 minst significante bits

4

Evaluatie van de veiligheid

In dit hoofdstuk staan de fundamentele verschillen tussen het originele en het nieuwe ontwerp en implementatie. Daarbij wordt vooral het veiligheidsaspect in acht genomen.

4.1 Geheimen in 12 bits

In de originele implementatie krijgt elke systeemaanroep die door IP-MON gemonitort zal worden een uniek geheim en het adres van de Replication Buffer mee in elk 64 bits. Dat gebeurt vanuit de kernel, waarop de kernel-patch is toegepast. In de nieuwe implementatie van IP-MON zijn er ook twee geheimen.

Het eerste geheim, het adres om naar IP-MON te kunnen springen, is uniek per variant en niet per systeemaanroep. Dat is minder veilig, maar langs de andere kant moet de systeemaanroep ook na het lekken van het geheim nog steeds door de seccomp-BPF filter geanalyseerd worden. Op die manier raken veiligheidsgevoelige systeemaanroepen nooit door het mechanisme naar IP-MON maar worden ze alsnog naar de cross-process monitor gestuurd.

Een drawback van seccomp-BPF is dat de errno-code, die we gebruiken om het geheim door te sturen, maar 16 bits groot is. Daarnaast wordt die 16 bits waarde eigenlijk afgetopt op 4095 (4096 waarden), wat neerkomt op 12 bits. Er moet ook nog rekening gehouden worden dat de eerste hexadecimaal 0x85 getallen gereserveerd worden voor errno's die gekend zijn door het besturingssysteem. Het komt er eigenlijk op neer dat we momenteel slechts 2325 verschillende adressen kunnen gebruiken om IP-MON in een variant te mappen bij een geheim van 12 bits. Bij geheimen van 48 en 76 bits, gebruiken we die waarde als sleutel om de overdracht van het adres van de IP-MON systeemaanroepcomponent te starten.

Het tweede geheim in de nieuwe implementatie is het adres van de Replication Buffer. In de originele implementatie is dat opnieuw een geheim dat bijgehouden wordt in de kernel. In de huidige implementatie is dat een thread_local variabele die we initialiseren door een vraag te stellen aan de cross-process monitor, die ook het adres van de Replication Buffer weet.

Dit is opnieuw een plaats waar nog verbetering mogelijk is in de nieuwe implementatie. Net zoals bij het eerste geheim, kunnen we gebruik maken van de seccomp-BPF filter die het geheim in meerdere stappen aan ons kan bezorgen.

De overhead kan hierdoor wel heel groot worden omdat we zowel hier als voor het eerste geheim meerdere keren langs de BPF-filter moeten passeren.

4.2 Onveilige overdracht grotere geheimen door beginsleutel

‘ In de huidige implementatie gebruiken we bij het gebruik van grotere geheimen een startsleutel om de overdracht van het adres van de IP-MON systeemaanroepcomponent te starten. Die sleutel wordt gebruikt in de seccomp-BPF filter als systeemaanroepnummer om aan te geven welke bits van het geheim doorgestuurd moeten worden door voorgedefinieerde waarden aan die sleutel toe te voegen. Wanneer een aanvaller de startsleutel te weten komt, en de bijhorende voorgedefinieerde waarden, kan hij zonder problemen het hele adres van de IP-MON systeemaanroepcomponent te weten komen.

Aangezien de filter statisch is, kunnen we dit probleem enkel potentieel oplossen door niet voor de hand liggende berekeningen te doen op de startsleutel of partiële delen van het resultaat om verschillende delen van het geheim te weten te komen. Die berekeningen moeten uniek zijn per uitvoering, en generatie van de bijhorende seccomp-BPF filter, van een programma in de MVUO ReMon.

4.3 Glibc

Ook met de aangepaste versie van glibc in combinatie met de seccomp-BPF filter moet zorgvuldig omgegaan worden. Wanneer een systeemaanroep voor de eerste keer in de filter komt, wordt die ofwel naar IP-MON omgeleid door een geheim terug te sturen of hij gaat rechtstreeks naar de cross-process monitor. Dat is een beslissing die enkel mag gebeuren als we er zeker van zijn dat de systeemaanroep eerst via de aangepaste, en in de variant ingeladen, versie van glibc passeert. Als we geen restricties opleggen wanneer het geheim mag teruggestuurd worden, kan het geheim lekken. Een mogelijkheid is dat er inline assembler in het programma geschreven staat, dat niet via glibc maar wel via de seccomp-BPF filter zal passeren om zo het geheim te weten te komen.

Om potentieel misbruik van dat geheim te voorkomen, is het de bedoeling om enkel systeemaanroepen die via onze aangepaste glibc passeren mogelijks om te leiden naar IP-MON door enkel dan het geheim mee te sturen. Als die niet via onze aangepaste versie van glibc passeert, wordt hij sowieso naar de cross-process monitor gestuurd.

4.4 Mapping van IP-MON

We weten al dat IP-MON in de adresruimte van de variant gemapt moet worden. Dat is iets dat de MVUO doet. Daarnaast wordt ook glibc in de adresruimte van elke variant gemapt door de MVUO.

Ook na het uitvoeren van een execve-functie in een variant in de originele implementatie, zal de MVUO IP-MON en glibc opnieuw in de adresruimte van de variant proberen mappen, weliswaar op een andere plaats dan voor de execve-aanroep. Door het gebruik van een seccomp-BPF filter in het nieuwe ontwerp moeten we dat gedrag aanpassen. Een seccomp-BPF filter is een constante filter, die wordt meegenomen over verschillende execve's heen waardoor de adressen die we in compare-instructies gebruiken in de seccomp-BPF filter ook constant blijven. Dat zorgt ervoor dat de MVUO moet garanderen dat IP-MON en glibc tijdens een execve-aanroep in een variant opnieuw op hetzelfde adres moet gemapt worden als voordien.

Dat zorgt er natuurlijk ook voor dat de mapping minder random wordt dan in de originele implementatie. Enkel de eerste mapping van IP-MON en glibc in de variant gebeurt random. Dat moet wel nog genuanceerd worden aangezien die mapping enkel in de MVUO gekend is, waar de variant zelf niet aankan, en als geheim in de seccomp-BPF filter zit.

5

Benchmarks

In dit hoofdstuk staan de opstellingen en de testresultaten beschreven die we uitvoeren om te meten hoe het nieuwe ontwerp en de implementatie ervan presteert ten opzichte van andere versies van de MVUO ReMon. Door die metingen te doen, krijgen we meer inzichten in de overhead die een seccomp-BPF filter in IP-MON introduceert.

Aangezien ReMon ontworpen is om te werken op linux, worden de testen ook op een computer met linux uitgevoerd. Meer specifiek op een computer met specificaties die in Tabel5.1 staan.

5.1 Microbenchmarks

De eerste meting is een microbenchmark. Daarmee bedoelen we dat we een klein aspect, zeg maar een taak, van een applicatie testen door enkel de uitvoering, en meer specifiek de uitvoeringstijd, van die taak te meten. Na het opstellen van de testomgeving, kunnen de metingen op verschillende versies uitgevoerd worden. Daarna worden de resultaten geëvalueerd.

5.1.1 Opstelling

Voor het meten van een taak waarbij zo weinig mogelijk andere overhead gecreëerd kan worden, kiezen we voor een eenvoudige systeemaanroep waarbij de implementatie geen andere systeemaanroepen doet. Een systeemaanroep die aan die voorwaarde voldoet is getpid.

Tabel 5.1: Specificaties computer waarop testen worden gedaan

Kenmerk	Waarde
Besturingssysteem	Ubuntu 20.04.4 LTS
CPU	6-core Intel® Core I5-10400
GPU	Intel® UHD Graphics 630 iGPU
RAM	32GB

Om de uitvoeringstijd zo nauwkeurig mogelijk te berekenen, schrijven we een programma dat de getpid systeemaanroep repetitief zal uitvoeren. De code van de gebruikte microbenchmark is terug te vinden in onderstaand codevoorbeeld Listing8. De microbenchmark meet de uitvoertijd van 10 000 000 getpid systeemaanroepen. Het hoge aantal uitvoeringen zorgt ervoor dat de ruis zal uitmiddelen. Die uitvoertijd wordt terug uitgemiddeld zodat we uitvoeringstijd van één getpid systeemaanroep weten. Die test wordt tien keer uitgevoerd.

Naast het aangepaste programma voor de microbenchmarks, moet ook het systeem voorbereid worden op het uitvoeren van de microbenchmarks om de metingen zo ruisvrij mogelijk te maken. De achtergrondtaken op het besturingssysteem van de host worden naar een minimum herleid. Daarnaast worden de turbo boost en hyperthreading functies van de CPU uitgeschakeld om meer stabiliteit in de metingen te brengen.

```
const int GETPID_TEST_COUNT     10000000
const int LOOP_TIMES            10
for (int size_i = 0; size_i < LOOP_TIMES; size_i++)
{
  auto start = std::chrono::high_resolution_clock::now();
  for (int cnt_i = 0; cnt_i < GETPID_TEST_COUNT; cnt_i++)
    getpid();
  auto end = std::chrono::high_resolution_clock::now();
  float result = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count();
  printf("\t> %u: %f ns\n", size_i, result / GETPID_TEST_COUNT);
}
```

Listing 8: Code van de microbenchmark met getpid systeemaanroepen

5.1.2 Versies

Om een vergelijking te kunnen maken, voeren we de metingen uit met verschillende versies van de MVUO ReMon. Onderstaande opsomming geeft weer op welke manieren, en verschillende versies, we de testen uitvoeren:

Native

Een native uitvoering van de microbenchmark, zonder gebruik te maken van een MVUO.

Default ReMon

Een MVUO, in dit geval ReMon, zal de microbenchmark evalueren door alle systeemaanroepen via zijn tracer-proces te monitoren.

Original ReMon with IP-MON with random 64 bit secret with getpid traced/allowed

De originele ReMon versie voert de microbenchmark uit. In deze versie wordt gebruik gemaakt van IP-MON en zal de getpid systeemaanroep niet of wel door IP-MON gemonitord worden, afhankelijk van de configuratie van IP-MON.

Voor deze versie wordt de originele kernelpatch toegepast op een linux kernel 5.4.0, waarbij IP-MON een random 64 bit getal per systeemaanroep zal genereren.

Original ReMon with IP-MON with zero 64 bit secret with getpid traced/allowed

Ook dit is de originele versie van ReMon die de microbenchmark uitvoer. In deze versie wordt tevens gebruik gemaakt van IP-MON en zal getpid, afhankelijk van de configuratie van IP-MON, niet of wel doorgelaten worden. Voor deze versie wordt een aangepaste kernelpatch toegepast op een linux kernel 5.4.0, waarbij IP-MON geen random 64 bit getal per systeemaanroep zal genereren. In plaats daarvan wordt het getal 0x0 als "uniek"getal per systeemaanroep ingesteld, om de overhead van het genereren van een random getal uit de meting te halen.

seccomp-BPF ReMon with IP-MON with zero 12 bit secret with getpid traced/allowed

Een uitvoering van de microbenchmarks door de nieuwe implementatie van ReMon, met IP-MON, waarbij een geheim van 12 bits door de filter wordt doorgegeven. Dat komt neer op één uitvoering van de seccomp-BPF filter om het geheim door te geven. Bij deze versie wordt de getpid systeemaanroep niet en wel door IP-MON gemonitord. Hierdoor kunnen we ook de overhead meten die gecreëerd wordt door het uitvoeren van de seccomp-BPF filter, waarna de systeemaanroep toch door CP-MON wordt gemonitord, ten opzichte van de originele implementatie.

seccomp-BPF ReMon with IP-MON with zero 76 bit secret with getpid traced/allowed

Zelfde uitvoering als de hierboven beschreven uitvoering, met een verschil in de grootte van het doorgegeven geheim. In deze versie zal de seccomp-BPF filter zeven keer uitgevoerd worden om een geheim van in totaal 84 bits door te geven, waarvan er 76 gebruikt worden.

seccomp-BPF ReMon with IP-MON with zero 48 bit secret with getpid allowed

Opnieuw dezelfde versie als de hierboven beschreven uitvoering, van de nieuwe implementatie van ReMon met IP-MON. Deze versie voert de filter vier keer uit om een geheim van 48 bits door te geven.

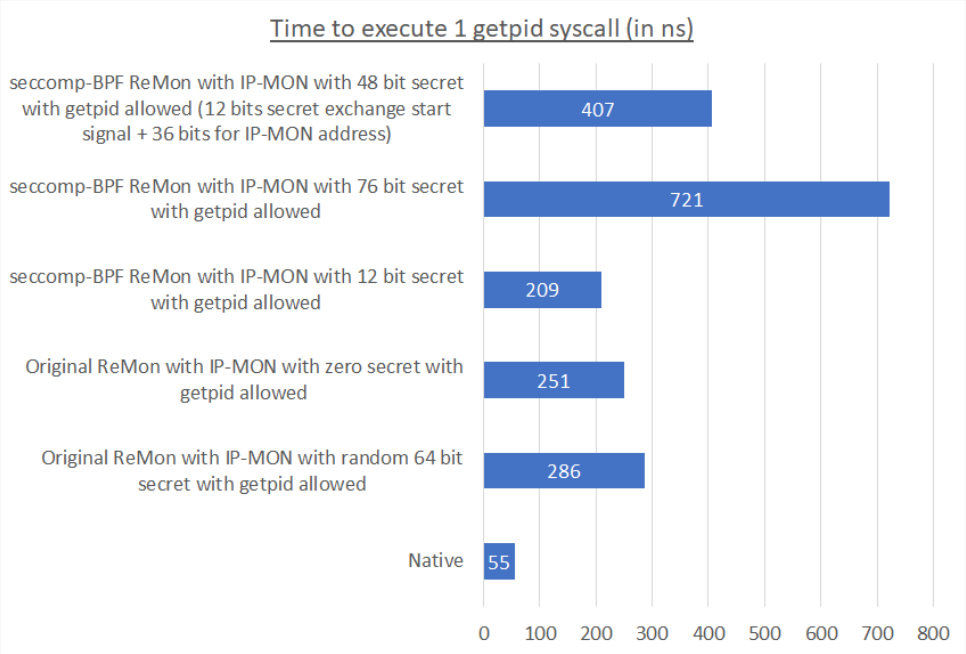
5.1.3 Resultaten

Uit de resultaten van de microbenchmark kunnen we twee aspecten evalueren. Een eerste aspect is de overhead van grote geheimen. Een tweede aspect is de snelheid ten opzichte van andere versies van ReMon, met en zonder IP-MON.

Overhead van grote geheimen

Figuur5.1 toont de resultaten van de uitvoering van de microbenchmark voor verschillende versies, waaruit we de overhead van grote geheimen kunnen meten. Die overhead wordt geïntroduceerd door het meermaals triggeren en uitvoeren van de seccomp-BPF filter.

We zien dat de originele versie van ReMon, waarbij getpid door IP-MON gemonitord wordt, een gemiddelde uitvoeringstijd heeft van 251 en 286 nanoseconden per getpid systeemaanroep, afhankelijk van het respectievelijk niet en wel genereren van een random 64 bit getal als geheim. Het genereren van dat geheim bedraagt om en bij de 35 nanoseconden.



Figuur 5.1: Meetresultaten van verschillende versies van ReMon, met IP-MON, om de overhead van grote geheimen in de nieuwe implementatie te meten en te vergelijken met de originele implementatie.

Wanneer we de gemiddelde uitvoeringstijd van een getpid systeemaanroep bekijken bij de uitvoering van die systeemaanroep in onze nieuwe implementatie van ReMon met IP-MON, zien we een verschil in snelheden. Dat verschil wordt geïntroduceerd door het meermaals uitvoeren van een seccomp-BPF filter. Hoe groter het geheim, hoe meer de seccomp-BPF filter uitgevoerd moet worden.

De eerste versie is "seccomp-BPF ReMon with IP-MON with zero 12 bit secret with getpid allowed". Die versie voert de seccomp-BPF filter eenmalig uit. Er kan vastgesteld worden dat deze versie sneller is dan de originele implementatie, maar boedt wel in op vlak van veiligheid zoals beschreven in Sectie4.1.

De tweede en derde versie zijn “seccomp-BPF ReMon with IP-MON with zero 48 bit secret with getpid allowed”en “seccomp-BPF ReMon with IP-MON with zero 76 bit secret with getpid allowed”. Bij de 48 bits versie zien we een goed evenwicht tussen veiligheid en snelheid. Er is een slechts een kleine, beperkte overhead door het meermaals uitvoeren van de seccomp-BPF filter ten opzichte van de originele implementatie. De 76 bits versie biedt een hogere veiligheid, maar is ook gevoelig trager.

We stellen vast dat de 48 bits versie van de nieuwe implementatie een goede oplossing biedt die snel en veilig genoeg is om de originele implementatie te vervangen.

Snelheid ten opzichte van andere versies

Aansluitend bij de overhead die geïntroduceerd wordt door het uitvoeren van de seccomp-BPF filter door het doorgeven van een geheim, is ook de evaluatiesnelheid, voor het kiezen tussen de veiligheidsgevoeligheid van een systeemaanroep, onder-

hevig aan het uitvoeren van een seccomp-BPF filter. In deze subsectie zetten we de meetresultaten van de uitvoering van de microbenchmark door verschillende versies naast elkaar om de uitvoeringssnelheden te vergelijken met alle mogelijke configuraties van IP-MON.

Figuur5.2 zet de meetresultaten op de verschillende versies met verschillende configuraties uit. De schaal van uitzetting is logaritmisch met basis 10, om de versies duidelijk met elkaar te kunnen vergelijken.

We zien dat bij een uitvoering waarbij getpid getraced wordt, m.a.w. doorgestuurd wordt naar CP-MON voor monitoring, er een verschil is tussen de originele en de nieuwe seccomp-BPF implementatie. Ook de "Default ReMon"mag mee in beschouwing genomen worden, aangezien die alle systeemaanroepen door CP-MON laat monitoren.

De originele ReMon implementatie, waarbij de in-kernel borker de systeemaanroep naar CP-MON doorstuurt, heeft een snelheid van 6617 tot 6680 nanoseconden om de getpid systeemaanroep uit te voeren. Dat is een fractie trager dan de "Default ReMon"versie, die daar 6726 nanoseconden over doet. Dat verschil is te wijten aan de evaluatie die de in-kernel broker moet doen om de systeemaanroep naar de juiste monitor door te sturen.

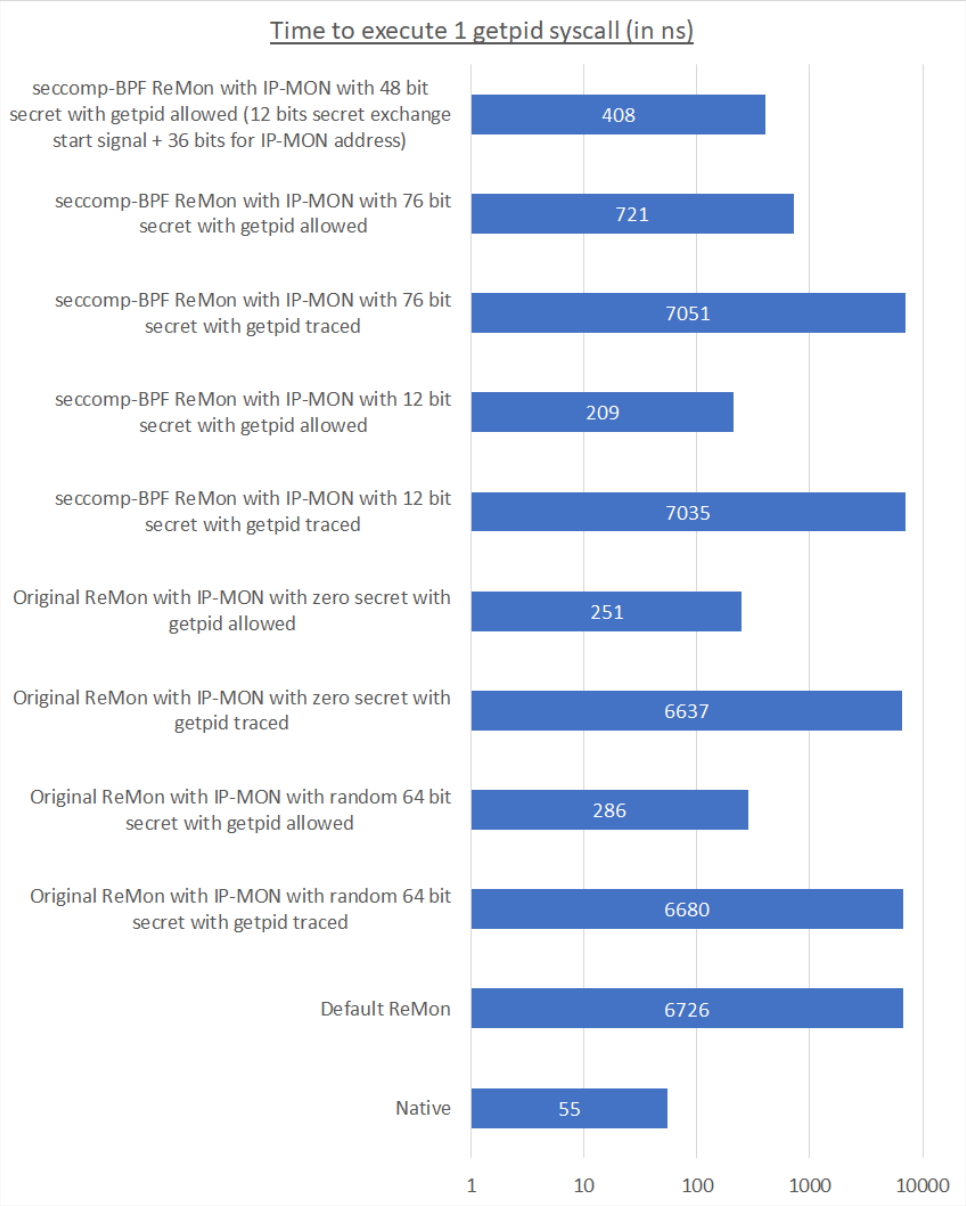
Wanneer we de nieuwe implementatie bekijken, zien we dat het daar zo'n 7035 nanoseconden duurt om de systeemaanroep getpid af te handelen wanneer die naar CP-MON wordt doorgestuurd. Dat is ongeveer 300 tot 400 nanoseconden trager dan een uitvoering door de "Default ReMon"MVUO of de originele implementatie van ReMon met IP-MON door gebruik te maken van een kernel-patch.

Die overhead, en dus lagere snelheid, die we bij de nieuwe implementatie zien komt opnieuw door de seccomp-BPF filter. Om dat te nuanceren kunnen we wel zeggen dat die overhead niet meer zal groeien omdat de filter slechts eenmalig wordt uitgevoerd om de systeemaanroep naar CP-MON door te sturen, in het geval dat er geen anomalieën optreden. Wanneer het systeemaanroepnummer of argumenten zouden wijzigen waardoor een niet veiligheidsgevoelige systeemaanroep naar een veiligheidsgevoelige systeemaanroep evalueert bij het aanroepen van de syscall-instructie in IP-MON, zal de filter slechts na het uitwisselen van het geheim evalueren naar een actie waarbij de systeemaanroep toch door CP-MON gemonitord moet worden. Aangezien dit geval slechts beperkt of zelden voorkomt, zal die geïntroduceerde overhead over de volledige uitvoering van een applicatie minimaal zijn.

Daarnaast is de langere uitvoeringstijd van een systeemaanroep in CP-MON bij de nieuwe implementatie ook minder erg aangezien er een veel snellere uitvoering van niet veiligheidsgevoelige systeemaanroepen kan gedaan worden.

5.2 nginx benchmarks

Een tweede benchmark die gedaan is, is de nginx benchmark. Met deze benchmark meten we de latency en de network throughput bij de uitvoering van een nginx server in een MVUO ReMON.



Figuur 5.2: Meetresultaten van verschillende versies van ReMon die de microbenchmark uitvoeren op een logaritmische schaal met basis 10.

5.2.1 Opstelling

Bij de uitvoering van een nginx server in een MVUO ReMon maken we gebruik van twee computers. Eén computer voert nginx uit in de MVUO ReMon, de tweede computer dient als client en zal aanvragen naar de server sturen via het wrk commando. Om die opstelling te realiseren, verbinden we de twee computers met een gigabit ethernet kabel en configureren de netwerkadapters zodat ze met elkaar kunnen communiceren.

5.2.2 Versies

Volgende versies worden gebruikt voor de benchmark van nginx:

Native

Een native uitvoering van de microbenchmark, zonder gebruik te maken van een MVUO.

Default ReMon

Een MVUO, in dit geval ReMon, zal de microbenchmark evalueren door alle systeemaanroepen via zijn tracer-proces te monitoren.

Original ReMon with IP-MON with random 64 bit secret with full allowed set

De originele ReMon versie voert de nginx benchmark uit. In deze versie wordt gebruik gemaakt van IP-MON en zal de complete set systeemaanroepen die IP-MON ondersteunt door IP-MON gemonitord worden. Voor deze versie wordt de originele kernelpatch toegepast op een linux kernel 5.4.0, waarbij IP-MON een random 64 bit getal per systeemaanroep zal genereren.

Original ReMon with IP-MON with random 64 bit secret with same allowed set as seccomp-BPF version

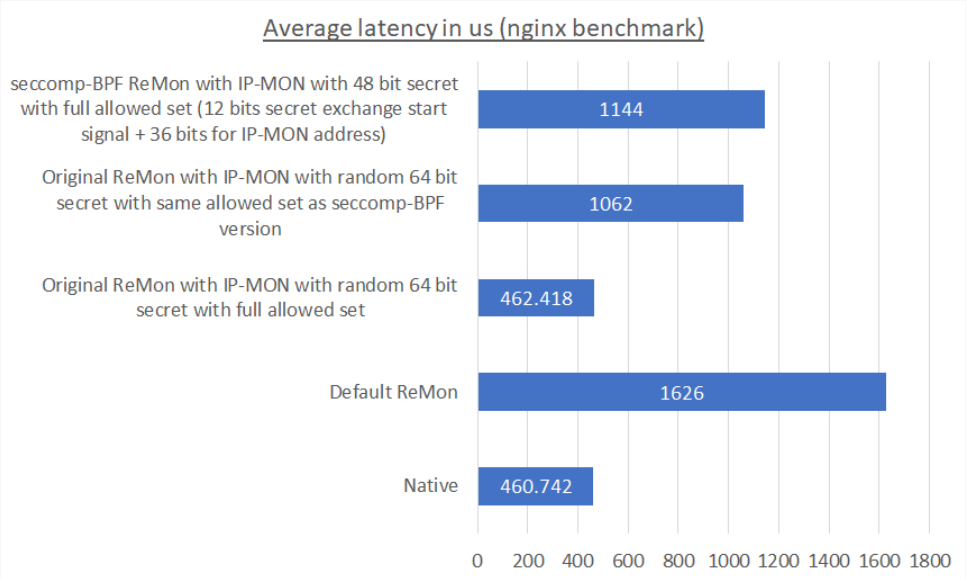
Deze originele ReMon versie voert de nginx benchmark uit. In deze versie wordt gebruik gemaakt van IP-MON en zal de overeenkomstige set systeemaanroepen die de nieuwe implementatie van IP-MON, met seccomp-BPF, ondersteunt door IP-MON gemonitord worden. Voor deze versie wordt de originele kernelpatch toegepast op een linux kernel 5.4.0, waarbij IP-MON een random 64 bit getal per systeemaanroep zal genereren.

seccomp-BPF ReMon with IP-MON with 48 bit secret with allowed set

Een uitvoering van de microbenchmarks door de nieuwe implementatie van ReMon, met IP-MON, waarbij een geheim van 48 bits door de filter meermaals uit te voeren, wordt doorgegeven. Dat komt neer op vier uitvoeringen van de seccomp-BPF filter om het geheim door te geven. Bij deze versie wordt de set systeemaanroepen die de nieuwe implementatie ondersteunt door IP-MON gemonitord.

5.2.3 Resultaten

Figuur5.3 toont geeft de gemiddelde latency weer per versie die de nginx benchmark uitvoert. We zien dat originele implementatie van ReMon met IP-MON een uitvoeringssnelheid heeft die vergelijkbaar is met een native uitvoering.

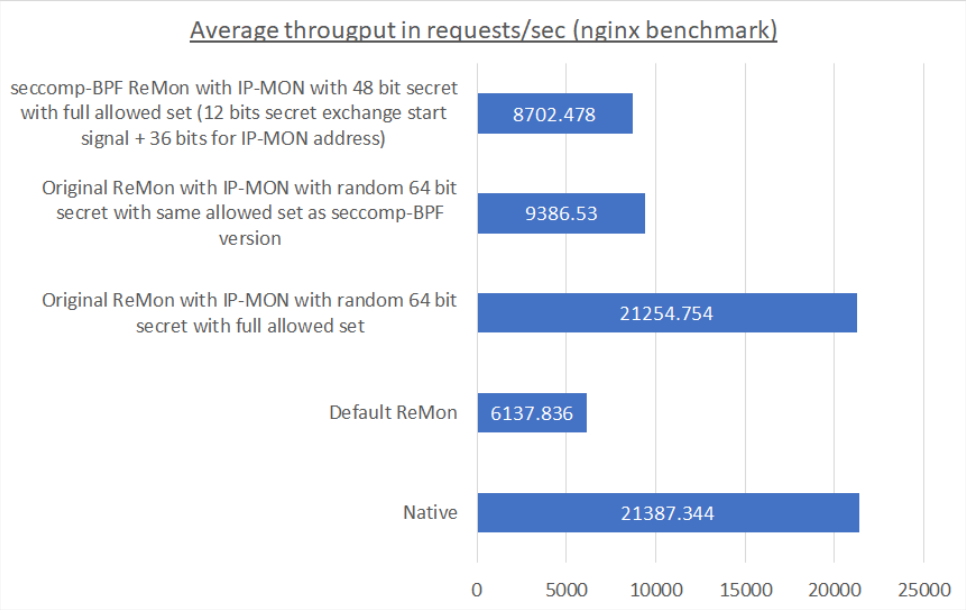


Figuur 5.3: Gemiddelde latency in microseconden bij de uitvoering van nginx in verschillende versies van de MVUO ReMon

Door de beperkte ondersteuning die de nieuwe implementatie biedt, kan niet dezelfde set systeemaanroepen in IP-MON gemonitord worden. We passen dezelfde set toe op de originele versie van ReMon met IP-MON.

Wanneer we de resultaten van de originele versie met een aangepaste policy vergelijken met de nieuwe versie die met seccomp-BPF werkt, zien we opnieuw gelijkaardige resultaten. Daaruit stellen we vast dat de nieuwe implementatie een vergelijkbare snelheid heeft als de originele implementatie, voor de reeds ondersteunde systeemaanroepen.

Figuur5.4 toont de resultaten van de gemiddelde throughput. Deze resultaten zijn omgekeerd evenredig met de latency en de resultaten kunnen dus op dezelfde wijze als hierboven geïnterpreteerd worden.



Figuur 5.4: Gemiddelde throughput in aanvragen per seconde bij de uitvoering van nginx in verschillende versies van de MVUO ReMon

6

Tekortkomingen

In dit hoofdstuk staan de tekortkomingen van de het nieuwe ontwerp en zijn implementatie in ReMon beschreven. In het laatste hoofdstuk is er een Sectie7.2 over verder onderzoek waarin meer uitleg staat over hoe de hierna beschreven tekortkomingen kunnen opgelost worden.

6.1 Beperkte set ondersteunde systeemaanroepen in IP-MON

De huidige implementatie van het nieuwe ontwerp biedt ondersteuning voor een beperkte subset van systeemaanroepen die door IP-MON afgehandeld kunnen worden. Deze set is kleiner dan de set waarvoor de originele implementatie ondersteuning biedt. De systeemaanroepen die niet meer ondersteund worden ten opzichte van de originele implementatie staan in Tabel6.1.

Tabel 6.1: Set van niet ondersteunde systeemaanroepen in de nieuwe implementatie van IP-MON teno opzichte van de originele implementatie

__NR_brk	__NR_close	__NR_fstat
__NR_mprotect	__NR_open	__NR_openat
__NR_pread64	__NR_preadv	__NR_pwrite64
__NR_read	__NR_readv	

De volledige lijst van extensief geteste systeemaanroepen die wel werken is terug te vinden in Bijlage 4.

Er moet mee in acht genomen worden dat de huidige implementatie slechts voor evaluatiedoeleinden dient om na te gaan

of het de tijd en moeite loont om de ondersteuning voor meer systeemaanroepen mogelijk te maken.

6.2 Combinatie van filters

6.2.1 Behoud van filters na clone- of execve-operaties

De seccomp-technologie stelt bij het kindproces na een clone-operatie exact dezelfde filter in als die van zijn ouderproces. Hetzelfde wordt gedaan na een execve-operatie. Die feature is belangrijk binnenin seccomp om de controle over een programma niet te verliezen.

Een tweede feature is dat meerdere filters ingesteld kunnen worden. De ingestelde filters worden in omgekeerde volgorde van toevoegen uitgevoerd. Dat wil zeggen dat de meest recent ingestelde filter eerst wordt uitgevoerd. Wanneer de filter geëvalueerd is naar een teruggeefactie, zal die de volgende filter uitvoeren. Dit zal gebeuren tot de eerst ingestelde filter ook is uitgevoerd. De teruggeefactie die werkelijk wordt teruggestuurd, is de eerstgeziene actie met de hoogste prioriteit. Onderstaande lijst geeft een overzicht van die prioriteit weer in afnemende prioriteit:

- SECCOMP_RET_KILL_PROCESS
- SECCOMP_RET_KILL_THREAD (of SECCOMP_RET_KILL)
- SECCOMP_RET_TRAP
- SECCOMP_RET_ERRNO
- SECCOMP_RET_USER_NOTIF
- SECCOMP_RET_TRACE
- SECCOMP_RET_LOG
- SECCOMP_RET_ALLOW

De twee voorgenoemde features beperken ons echter in de vrijheid de we hebben om filters toe te voegen met verschillende statische geheimen. Dat komt omdat we een verschil in prioriteit zouden creëren om beslissingen te nemen bij het toevoegen van verschillend evaluerende filters. De drie acties die in onze nieuwe implementatie worden gebruikt zijn: SECCOMP_RET_ALLOW, SECCOMP_RET_TRACE en SECCOMP_RET_ERRNO.

Wanneer we, zoals bij de originele implementatie, bij elke clone- of execve-operatie IP-MON op een ander adres zouden mappen, zouden we entries moeten toevoegen in de seccomp-BPF filter om systeemaanroepen die vanuit de systeemaanroepcomponent van de nieuw gemapte IP-MON komen, door IP-MON te laten monitoren. Onderstaande filter is een dubbele

filter van een kindproces, gevolgd door die van zijn ouderproces. Door het verschil in adres waarop IP-MON gemapt zit, zal elke filter naar een ander resultaat evalueren afhankelijk van in welke actie we zitten in het tweetrapssysteem dat we gebruiken om een systeemaanroep door IP-MON te laten monitoren.

Nemen we om dit te illustreren een getpid-instructie die het te monitoren programma zal uitvoeren. De eerste actie in het tweetrapssysteem zorgt ervoor dat de dubbele filter, die voorgesteld wordt in pseudocode in Listing9 als volgt zal evalueren. De systeemaanroep komt in de eerste filter terecht. Daar wordt gezien dat de systeemaanroep niet vanuit de systeemaanroepcomponent van IP-MON van het kindproces komt. Maar de systeemaanroep mag wel in IP-MON terechtkomen voor monitoring. Daardoor evalueert de eerste filter naar een SECCOMP_RET_ERRNO actie, die het adres van de IP-MON systeemaanroepcomponent zal doorsturen. De tweede filter wordt ook nog uitgevoerd. Daar gebeurt net hetzelfde. De filter ziet dat de systeemaanroep niet vanuit IP-MON van het ouderproces komt, maar de filter mag wel in IP-MON terechtkomen. Daardoor evalueert de filter naar dezelfde SECCOMP_RET_ERRNO actie. De waarde die daarbij hoort is een ander adres van IP-MON, meer specifiek het adres van IP-MON van het ouderproces. Maar aangezien beide acties hetzelfde zijn, op hun meegestuurde waarde na, is ook hun prioriteit even hoog. Daardoor zal de eerstgeziene actie met deze prioriteit teruggestuurd worden. Dat is die van de eerste filter. De correcte actie dus.

glibc zal naar IP-MON van het kindproces kunnen springen en voert daar de tweede trap van het systeem uit, de systeemaanroep vanuit de systeemaanroepcomponent van IP-MON. We zien dat bij het uitvoeren van een getpid-instructie vanuit IP-MON in ons kindproces de eerste filter zal evalueren naar een SECCOMP_RET_ALLOW actie. Daarna wordt ook de filter van het ouderproces uitgevoerd. Die filter zal evalueren naar een SECCOMP_RET_ERRNO omdat het adres van de systeemaanroepcomponent van het kindproces niet overeenkomt met dat van het ouderproces want IP-MON zit op twee verschillende adressen gemapt. Aangezien de eerstgeziene actie met de hoogste prioriteit wordt teruggestuurd, zal de SECCOMP_RET_ERRNO uit de tweede filter teruggestuurd worden, hoewel we eigenlijk de systeemaanroep mochten monitoren vanuit IP-MON. De systeemaanroep in IP-MON zal nu een onbekende errno-waarde terugkrijgen, die eigenlijk bedoeld is voor het uitvoeren van de tweede trap in het tweetrapssysteem waar we al beland zijn, en de uitvoering van het programma zal foutlopen aangezien we verwachten dat de systeemaanroep daar gewoon wordt uitgevoerd, of in het slechtste geval naar CP-MON wordt doorgestuurd.

```
set bpf_allowed_syscalls_child = [__NR_getpid, ..., __NR_gettid]
set bpf_allowed_syscalls_parent = [__NR_getpid, ..., __NR_gettid]

// start child filter
if ipmon_child_syscall_address_ptr == seccomp_data.instruction_pointer do
    if seccomp_data.nr in bpf_allowed_syscalls_child do
        return SECCOMP_RET_ALLOW
    end else do
        return SECCOMP_RET_TRACE
    end
end else do
    if seccomp_data.nr in bpf_allowed_syscalls_child do
```

```
        return SECCOMP_RET_ERRNO
    end else do
        return SECCOMP_RET_TRACE
    end
end
// end of child filter

// start parent filter
if ipmon_parent_syscall_address_ptr == seccomp_data.instruction_pointer do
    if seccomp_data.nr in bpf_allowed_syscalls_parent do
        return SECCOMP_RET_ALLOW
    end else do
        return SECCOMP_RET_TRACE
    end
end else do
    if seccomp_data.nr in bpf_allowed_syscalls_parent do
        return SECCOMP_RET_ERRNO
    end else do
        return SECCOMP_RET_TRACE
    end
end
// end of parent filter

return FIRST_OCCURED_HIGHEST_PRIORITY_RET_ACTION
```

Listing 9: Dubbele filter kind- en ouderproces met verschillende adressen van IP-MON

Op die manier zien we dat we IP-MON niet mogen mappen op verschillende adressen in eenzelfde variant of zijn kindprocessen na een clone- of execve-operatie. Met andere woorden wordt IP-MON op constante adressen gemapt na clone- of execve-operaties. Daardoor moet ook de filter slechts éénmaal ingesteld worden per proces i.e. variant.

6.2.2 Monitoren van programma's die zelf gebruik maken van seccomp-BPF filters

Zoals in de vorige sectie beschreven staat, vormt het combineren van filters in de nieuwe implementatie van ReMon met de seccomp-BPF technologie een bron voor problemen. Filters zullen niet meer evalueren naar de gewenste actie op een bepaald punt in het tweetrapssysteem. Het probleem uit de vorige sectie kunnen we deels oplossen door de mapping van IP-MON constant te houden en filters slechts eenmalig per proces in te stellen.

Een andere manier waarop meerdere filters naast onze filter van IP-MON worden toegevoegd, is dat het te monitoren pro-

gramma zelf gebruik maakt van seccomp-BPF filters en ze dus ook instelt. Dat is het geval bij hedendaagse webbrowsers zoals FireFox en Google Chrome, die door het gebruik van seccomp-BPF filters een sandboxing mechanisme proberen opzetten.

Wanneer we zulke programma's uitvoeren in de ReMon MVUO met de nieuwe implementatie van IP-MON, zal dat ervoor zorgen dat filters toegevoegd worden bovenop onze filter van IP-MON waardoor de evaluatie naar een actie onvoorspelbaar wordt. Ook het gedrag zal dus verschillen van wat we verwachten waardoor er fouten optreden.

Dit probleem kan enkel opgelost worden door het uitschakelen van het gebruik van de seccomp-BPF technologie in ofwel het te monitoren programma of de ReMon MVUO. Sommige programma's laten de gebruiker toe het gebruik van de seccomp-BPF technologie uit te schakelen. Op die manier kan ReMon met de nieuwe implementatie van IP-MON nog steeds gebruikt worden om zo'n programma te monitoren. In het geval dat het programma het niet toelaat het gebruik van de seccomp-BPF technologie uit te schakelen, kan het programma ook onderworpen worden aan de ReMon MVUO zonder gebruik te maken van IP-MON. Een default MVUO dus.

7

Conclusie en verder onderzoek

In het laatste hoofdstuk van deze masterproef wordt een conclusie geschreven. We gaan kijken of het resultaat in de lijn der verwachtingen lag en op welke vlakken er verbeteringen mogelijk zijn. Daarnaast staat in dit hoofdstuk ook een duurzaamheidsreflectie geschreven waarin gekeken wordt binnen welke SDGs deze masterproef kadert.

7.1 Conclusie

In deze thesis werd nagegaan of we het ontwerp en de implementatie van Monitoring Relaxation kunnen aanpassen zodat de functionaliteit van de kernelpatch, die nodig is om IP-MON in de ReMon MVUO te laten werken, vervangen kan worden door functionaliteit die gebruik maakt van nieuwe technologieën in de Linux kernel. Daarnaast is het belangrijk dat na die aanpassingen dezelfde uitvoeringssnelheden behaald kunnen worden als in de originele implementatie. Op die manier zou een kernelpatch overbodig worden.

Het nieuwe ontwerp en de nieuwe implementatie, die gebruik maakt van seccomp-BPF filtering, is beloftevol. De nieuwe implementatie behaalt snelheden die vergelijkbaar zijn met de originele implementatie, voor de systeemaanroepen die tot nog toe ondersteund worden. Naast het snelheidsaspect, werd ook ingezet op het behouden van een hoog veiligheidsaspect, waaraan ook voldaan wordt. Dat door het uitbreiden van de nieuwe implementatie zodat die ondersteuning biedt voor geheimen met een grotere grootte. Daarbij moet wel een overhead in snelheid in acht worden genomen, die lineair groeit naarmate de grootte van een geheim groeit.

De tot nog toe ondersteunde set van systeemaanroepen in de nieuwe implementatie is nog niet klaar voor omverblazende prestaties. Maar er zit wel potentieel in deze nieuwe versie, die vergelijkbaar in snelheid genoemd kan worden met de originele implementatie wanneer we enkel de tot nog toe ondersteunde systeemaanroepen van de nieuwe implementatie in vergelijking nemen.

De broncode van de implementatie is terug te vinden in de volgende twee projecten op GitHub:

ReMon

https://github.com/lennertfranssens/ReMon

ReMon-glibc

https://github.com/lennertfranssens/ReMon-glibc/

7.2 Verder onderzoek

De nieuwe implementatie is nog niet helemaal klaar. Er is slechts een beperkte set van ondersteunde systeemaanroepen die door IP-MON gemonitord kunnen worden. Om de set gelijk te maken aan de set van ondersteunde systeemaanroepen uit de originele implementatie, is verder onderzoek nodig.

Daarnaast kan ook verder gezocht worden naar veiligere manieren om grotere geheimen door te sturen vanuit IP-MON, niet noodzakelijk via de seccomp-BPF filter. Dat is nodig om ook het geheim van de Replication Buffer door te sturen, wat nu voor een grote overhead in snelheid zou zorgen.

Tot slot zou het probleem met de groeiende filters, dat beschreven staat in Sectie6.2, potentieel opgelost kunnen worden door meer onderzoek te doen naar strategieën waarbij de hoogste prioriteit van alle filters samen, de door ons bedoelde actie voorstelt.

7.3 Ethische en maatschappelijke reflectie

Het geleverde werk in deze masterproef kadert binnen één aspect in de SDGs. Dat aspect wordt in de volgende sectie besproken, samen met hoe deze masterproef binnen dat aspect kadert.

7.3.1 Industrie, innovatie en infrastructuur

SDG 9.1 beschrijft het ontwikkelen van kwalitatieve, betrouwbare en veerkrachtige infrastructuur[12]. In deze masterproef is het design van de ReMon-applicatie aangepast zodat Monitoring Relaxation breder beschikbaar gemaakt kan worden. Monitoring Relaxation versnelt de werking van een MVUO zodat de overhead beperkt wordt, en de keuze om deze toepassing te gebruiken verhoogt. We maken de beschikbaarheid en toegankelijkheid breder waardoor meer mensen gebruik kunnen maken van een applicatie die niet enkel sneller werkt dan zijn alternatieven, maar ook veiligheid nog steeds voorop zet.

Er wordt voldaan aan zowel het kwalitatieve, het betrouwbare, duurzame en veerkrachtige aspect dat in SDG 9.1 omschreven staat.

Referenties

[1] S. Volckaert, "Geavanceerde technieken voor de uitvoering van meerdere varianten," Ph.D. dissertation, Universiteit Gent, 2016.

[2] UN, "Take action for the sustainable development goals," https://www.un.org/sustainabledevelopment/sustainable-development-goals/, geraadpleegd op 30 maart 2022.

[3] "Seccomp bpf (secure computing with filters)," https://www.kernel.org/doc/html/v5.15/userspace-api/seccomp_filter.html, geraadpleegd op 2 november 2021.

[4] "Syscall user dispatch," https://www.kernel.org/doc/html/v5.15/admin-guide/syscall-user-dispatch.html, geraadpleegd op 7 november 2021.

[5] "ptrace(2) — linux manual page," https://man7.org/linux/man-pages/man2/ptrace.2.html, geraadpleegd op 12 november 2021.

[6] Team PaX, "Pax address space layout randomization (aslr)," 2003.

[7] S. Volckaert, B. D. Sutter, T. D. Baets, and K. D. Bosschere, "Ghumvee: efficient, effective, and flexible replication," in *International Symposium on Foundations and Practice of Security*. Springer, 2012, pp. 261–277.

[8] "seccomp(2) — linux manual page," https://man7.org/linux/man-pages/man2/seccomp.2.html, geraadpleegd op 17 oktober 2021.

[9] E. Garcia, "New year, new kernel: Collabora's contributions to linux 5.11," *Collabora News*, 2021. [Online]. Available: https://www.collabora.com/news-and-blog/news-and-events/new-year-new-kernel-collabora-contributions-linux-511.html

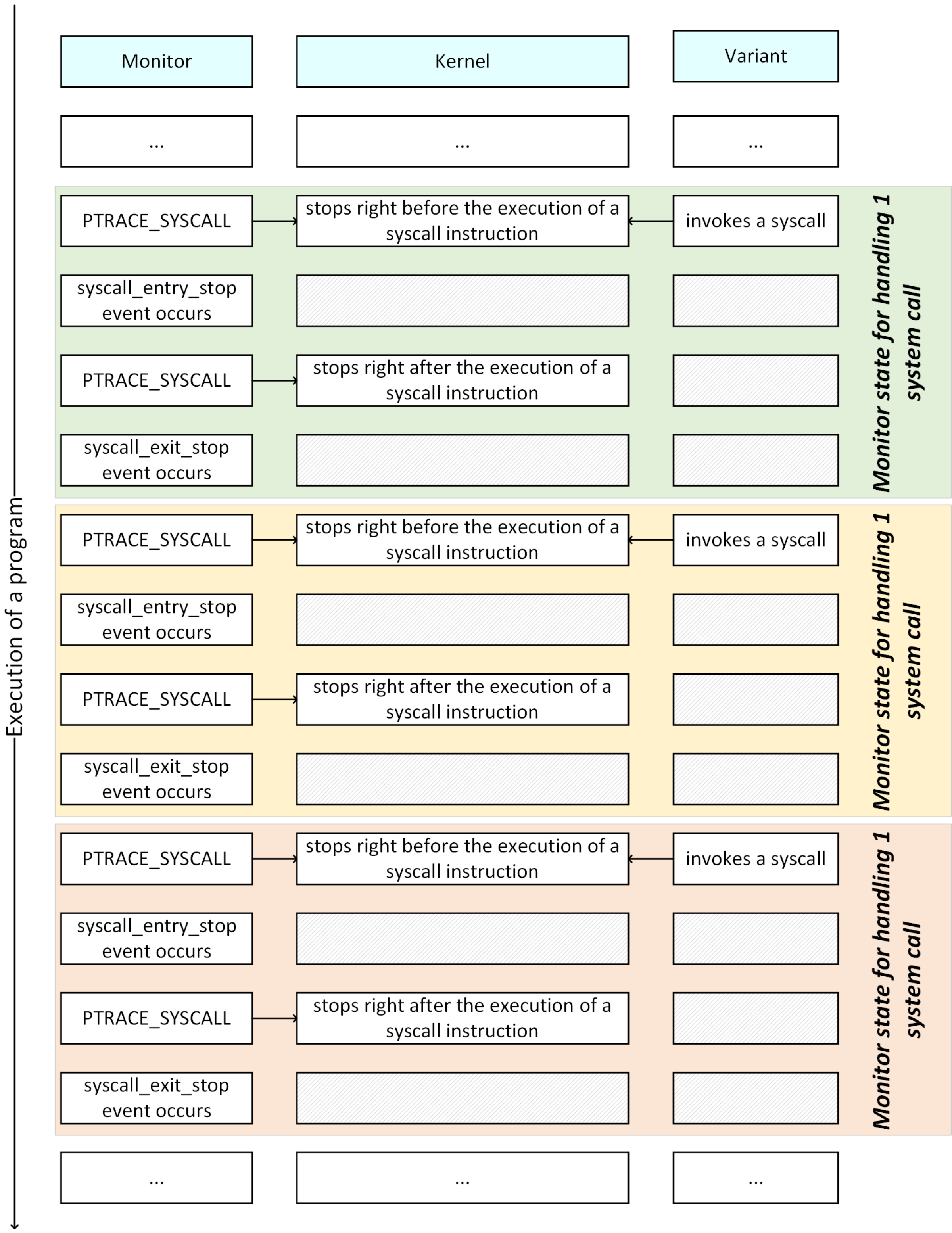
[10] "Remon-glibc," https://github.com/ReMon-MVEE/ReMon-glibc, geraadpleegd op 29 november 2021.

[11] "errno(3) — linux manual page," https://man7.org/linux/man-pages/man3/errno.3.html, geraadpleegd op 1 maart 2022.

[12] UN, "Goal 9: Build resilient infrastructure, promote sustainable industrialization and foster innovation," https://www.un.org/sustainabledevelopment/infrastructure-industrialization/, geraadpleegd op 30 maart 2022.

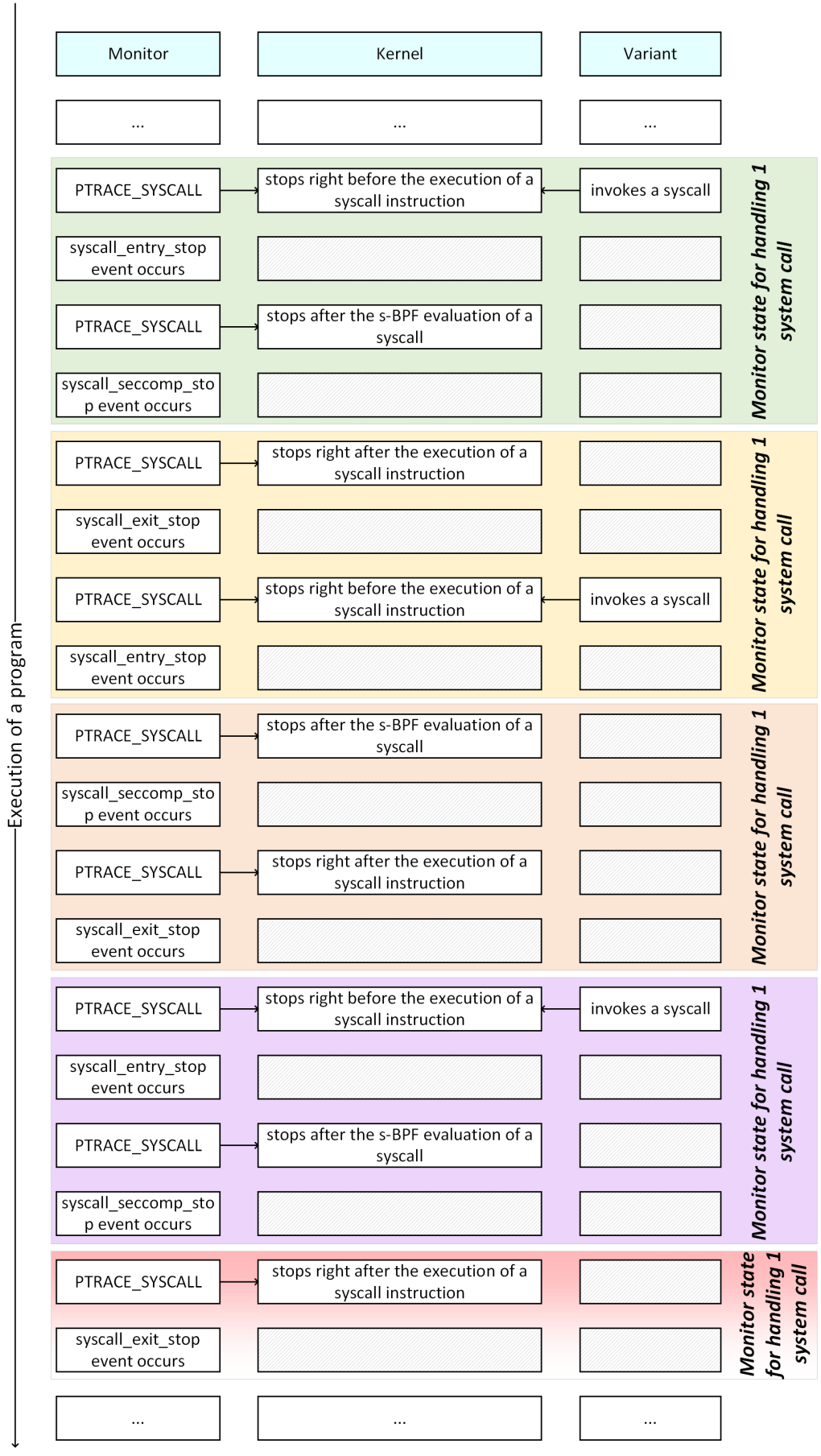
Bijlagen

Bijlage 1

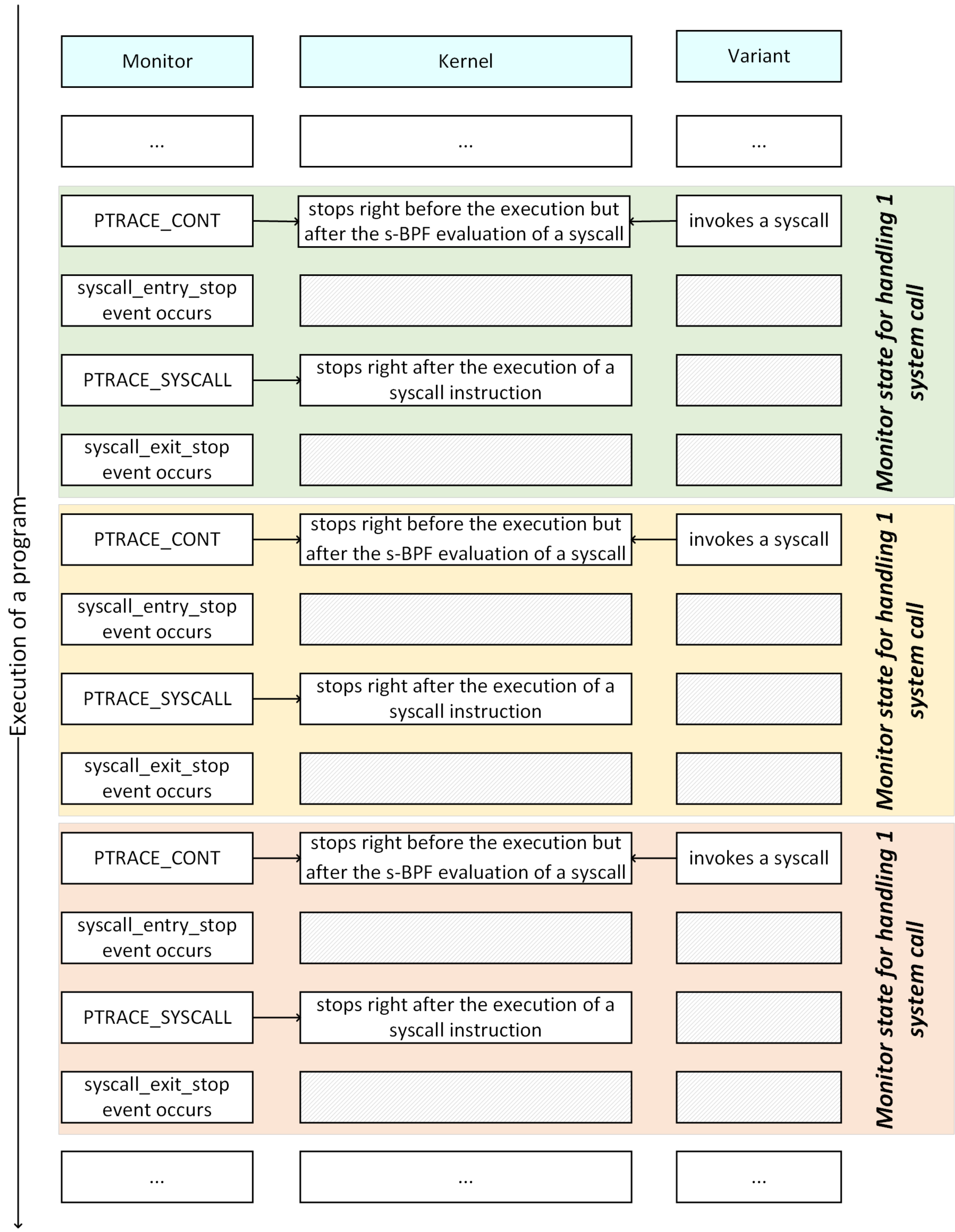


Execution of a program
Monitor
Kernel
Variant
...
...
...
PTRACE_SYSCALL
stops right before the execution of a syscall instruction
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
PTRACE_SYSCALL
stops right before the execution of a syscall instruction
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
PTRACE_SYSCALL
stops right before the execution of a syscall instruction
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
...
...
...

Bijlage 2


Execution of a program
Monitor
Kernel
Variant
...
...
...
PTRACE_SYSCALL
stops right before the execution of a syscall instruction
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops after the s-BPF evaluation of a syscall
syscall_seccomp_stop event occurs
Monitor state for handling 1 system call
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
PTRACE_SYSCALL
stops right before the execution of a syscall instruction
invokes a syscall
syscall_entry_stop event occurs
Monitor state for handling 1 system call
PTRACE_SYSCALL
stops after the s-BPF evaluation of a syscall
syscall_seccomp_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
PTRACE_SYSCALL
stops right before the execution of a syscall instruction
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops after the s-BPF evaluation of a syscall
syscall_seccomp_stop event occurs
Monitor state for handling 1 system call
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
...
...
...

Bijlage 3



Execution of a program
Monitor
Kernel
Variant
...
...
...
PTRACE_CONT
stops right before the execution but after the s-BPF evaluation of a syscall
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
PTRACE_CONT
stops right before the execution but after the s-BPF evaluation of a syscall
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
PTRACE_CONT
stops right before the execution but after the s-BPF evaluation of a syscall
invokes a syscall
syscall_entry_stop event occurs
PTRACE_SYSCALL
stops right after the execution of a syscall instruction
syscall_exit_stop event occurs
Monitor state for handling 1 system call
...
...
...

Bijlage 4

Deze bijlage bevat de volledige lijst extensief geteste systeemaanroepen die ondersteund worden in de nieuwe implementatie van IP-MON.

Tabel 1: Set van ondersteunde systeemaanroepen in de nieuwe implementatie van IP-MON

__NR_accept	__NR_accept4	__NR_access
__NR_alarm	__NR_bind	__NR_capget
__NR_chdir	__NR_clock_gettime	__NR_connect
__NR_dup	__NR_dup2	__NR_dup3
__NR_epoll_create	__NR_epoll_create1	__NR_epoll_ctl
__NR_epoll_wait	__NR_faccessat	__NR_fadvise64
__NR_fchdir	__NR_fcntl	__NR_fdatasync
__NR_fgetxattr	__NR_fsync	__NR_getcwd
__NR_getdents	__NR_getegid	__NR_geteuid
__NR_getgid	__NR_getitimer	__NR_getpeername
__NR_getpgrp	__NR_getpid	__NR_getppid
__NR_getpriority	__NR_getrusage	__NR_getsockname
__NR_getsockopt	__NR_gettid	__NR_gettimeofday
__NR_getuid	__NR_getxattr	__NR_inotify_init
__NR_inotify_init1	__NR_ioctl	__NR_lgetxattr
__NR_listen	__NR_lseek	__NR_lstat
__NR_madvise	__NR_mkdir	__NR_mremap
__NR_nanosleep	__NR_newfstatat	__NR_pipe
__NR_pipe2	__NR_poll	__NR_pwritev
__NR_readlink	__NR_readlinkat	__NR_recvmmsg
__NR_sched_yield	__NR_select	__NR_sendfile
__NR_sendmmsg	__NR_sendto	__NR_setitimer
__NR_setsockopt	__NR_shutdown	__NR_socket
__NR_socketpair	__NR_stat	__NR_sync
__NR_syncfs	__NR_sysinfo	__NR_time
__NR_timerfd_gettime	__NR_timerfd_settime	__NR_times
__NR_uname	__NR_write	__NR_writev