Pointer Flow Integrity

Формальная спецификация на языке Z

Сессия отладки ядра Windows 11 ARM64 (build 26100)

2026-07-25

1 Назначение и статус

PFI формализует, как *визор* (DBI/секвенсор) наблюдает и изолирует управление и поток данных задачи и как он обнаруживает нелегитимный перехват со стороны эмуляторов антивирусов (AVM) и виртуализационных прослоек (VMP / VM-call). Документ содержит:

1. абстрактную модель секвенции против машинного потока, графа потока указателей (PFG) и инварианта целостности данных (DFI);
2. перенаправление IDP (программный анклав), предикат обнаружения AVM и подмену адреса возврата S-route для атомарных шлюзов;
3. конкретную инстанциацию на верифицированной диспетчерской таблице ядра с теоремами.

Замечание о гигиене Z: для адресной арифметики используется функция Shift — *не* символ $\oplus$, поскольку в Z $\oplus$ обозначает overriding функций/отношений.

2 Базовые множества и определения

$$[ADDR,\ INSTR,\ REF,\ REG,\ VAL,\ SSN,\ EC,\ Process]$$

$$ATOM ::= syscall \mid sysenter \mid indirectCall \mid vmcall \mid gate$$
$$Mode ::= User \mid Kernel$$
$$Access ::= Rd \mid Wr \mid Ex$$
$$Prot ::= ReadOnly \mid RW$$

$$FlowElem == INSTR \cup ATOM$$
$$RetGate == \{\, syscall,\ sysenter \,\}$$
$$GateAtom == ATOM$$

$$EA == ADDR$$
$$Ptr == ADDR$$

$$Bool ::= true \mid false$$
$$PACKey == ADDR \times ADDR$$
$$SignedAddr == ADDR$$

$$EVENT ::= Enter\langle ATOM \times ADDR \rangle \ \mid \ Exit\langle ATOM \times ADDR \rangle$$

Глобальные сигнатуры (фиксируются семантикой ISA):

$Accesses : FlowElem \leftrightarrow ADDR$ — отношение «элемент обращается по адресу»
$Select : FlowElem \to \mathbb{N}$ — размер выборки (байты)
$Shift : ADDR \times \mathbb{Z} \to ADDR$ — адресный сдвиг
PageOf : $ADDR \to ADDR$ — база страницы (здесь: large page / PDE)
Protect : $ADDR \to Prot$
$WriteFault : \mathbb{P}\, ADDR$ — адреса, запись в которые вызывает отказ
$Exec : ADDR \to Bool$ — страница исполняемая

3 Поток управления: секвенция против машинного потока

Визор производит *секвенированный* поток Cf (исполняется из буфера визора), процессор исполняет *машинный* поток Xf. Они расходятся на атомах, когда исполнение покидает буфер (смена адреса снимает хуки AV-эмулятора).

Sequencing
Cf : seq $FlowElem$
Xf : seq $FlowElem$
$Cf \neq \langle\rangle$

Diverges
$a? : ATOM$
$k : \mathbb{N}$
$Cf\, k = a?$
$Xf\, k \neq a? \;\lor\; \#Xf \neq \#Cf$

Атом $a?$ в позиции k потока Cf *расходится*, если машинный поток в этой позиции не равен $a?$ или длины потоков различны — формальная запись утверждения «атомы уходят из-под секвенсора».

4 Состояние задачи

TS
$regs : REG \rightharpoonup VAL$
$frame$: seq VAL
$prevMode : Mode$
$lr : ADDR$
$retSlot : ADDR$
$prevMode \in \{User,\ Kernel\}$

Поле $retSlot$ — адрес, по которому возобновляется исполнение после системного атома; на верифицированном ядре это пара lr/fp, восстанавливаемая в `KiSystemServiceCopyEnd+0x38` инструкцией `ldp fp,lr,[sp,#0x50]`.

5 Граф потока указателей (PFG)

PFG
$ts : TS$
$CS : \mathbb{P}\, Ptr$
$IS : FlowElem \to \mathbb{P}\, Ptr$
$enclave : \mathbb{P}\, ADDR$
$\delta : \mathbb{Z}$
$enclave \neq \varnothing$

CS — текущее множество живых указателей, IS — разрешённое входное множество на элемент потока, *enclave* — изолированная область (программный анклав), δ — смещение перенаправления IDP.

6 Инвариант целостности данных (DFI)

Центральное свойство: любое обращение в память, выполняемое секвенированным элементом, должно быть по адресу из текущего множества или из входного множества этого элемента.

DFI
PFG
$\forall e : FlowElem;\ ea : EA \mid$ $\quad (e \mapsto ea) \in Accesses \ \Rightarrow\ ea \in CS \cup IS(e)$

Обращение, чьего EA нет ни в одном из множеств, не может быть порождено санкционированным потоком — это сигнатура внедрённого перехвата (критерий фиксации AVM / OP-инжектов).

7 IDP — перенаправление EA в программный анклав

Redirect
$ea? : EA$ $ea! : EA$
$ea! = \text{Shift}(ea?,\ \delta)$ $ea! \in enclave$

При секвенции визор сдвигает эффективный адрес на δ, направляя выборку данных в изолированный анклав; эмулятор AB, читающий *исходный* адрес, не видит трафика.

8 Обнаружение AVM / эмулятора

AVMDetected
$e? : FlowElem$ $ea? : EA$
$(e? \mapsto ea?) \in Accesses \ \wedge\ ea? \notin CS \cup IS(e?)$

Свидетельство AVMDetected в ходе секвенции $\Rightarrow$ присутствует нелегитимная эмуляция/перехват. Это двойник инварианта DFI: DFI утверждает отрицание для санкционированного потока.

9 Атомы и S-route

$enabled : \mathbb{P}\, ATOM$ — множество наблюдаемых атомов

Наблюдение атомов (ядерный аналог — `KiTrackSystemCallEntry/Exit`, включается битом 0 по `PpmPolicyConfigTable+0xD00`):

ObserveAtom
$a? : ATOM$ $target? : ADDR$ $evt! : EVENT$
$a? \in enabled$ $evt! = Enter(a?,\ target?)$

S-route: для возвратного атома визор сохраняет исходный адрес возврата и подменяет его своей заглушкой, чтобы управление вернулось в секвенсор, а не в API эмулятора AB.

$STUB : ADDR$ — заглушка возобновления S-route $OK : EC$ $NtCode : \mathbb{P}\, ADDR$ — кодовая область обработчиков `nt!Nt*`

SRoute
ΔPFG $a? : ATOM$ $saved! : ADDR$
$a? \in RetGate$ $saved! = ts.retSlot$ $ts.retSlot' = STUB$ $CS' = CS;\ IS' = IS;\ enclave' = enclave;\ \delta' = \delta$

RestoreReturn
ΔPFG $saved? : ADDR$
$ts.retSlot' = saved?$

10 Конкретная инстанциация ядром (верифицировано)

Значения сняты непосредственно с живого ядра.

$KiServiceTableBase : ADDR$ $KiArgTable : ADDR$ $KiServiceLimitValue : \mathbb{N}$ $DescriptorAddr : ADDR$ $InvalidSvcEC : EC$
$KiServiceLimitValue = 489$

Функция декодирования реализует инструкцию `add x11,x11,x10,asr#4`:

$Decode : ADDR \times \mathbb{N} \to ADDR$
$\forall b : ADDR;\ e : \mathbb{N} \mid$ $\quad Decode(b,\ e) = \mathrm{Shift}(b,\ e \div 16)$

10.1 Дескриптор и таблица сервисов

ServiceTable
$base : ADDR$ $limit : \mathbb{N}$ $number : ADDR$ $table : SSN \rightharpoonup \mathbb{N}$
$base = KiServiceTableBase$ $limit = KiServiceLimitValue$ $number = KiArgTable$

10.2 Диспетчеризация (успешный путь)

Реализует `cmp x8,x10; bhs fail` и `ldrsw`/`add ... asr#4` из `KiSystemService`:

Dispatch
$ServiceTable$ $ssn? : SSN$ $handler! : ADDR$ $status! : EC$
$ssn? < limit$ $ssn? \in \text{dom } table$ $handler! = Decode(base,\ (table\, ssn?) \div 16)$ $status! = OK$

10.3 Выход за пределы (путь ошибки)

Реализует `mov x0,#0x1C; movk x0,#0xC000,lsl#0x10` в `KiSystemServiceCopyEnd+0x88`:

InvalidService
$ServiceTable$ $ssn? : SSN$ $status! : EC$
$ssn? \geq limit$ $status! = InvalidSvcEC$

10.4 Целостность таблицы (отображение только для чтения)

Проверено командой `!pte fffff8004bc96c28`: атрибуты PDE `-R-GA-K-LV` (large page, **без W**), тогда как обёртка-дескриптор по адресу `fffff8004cc01900`: `-W-GADK-LV` (доступна для записи):

TableIntegrity
$ServiceTable$
$\text{Protect}(\text{PageOf}(base)) = ReadOnly$ $\text{Protect}(\text{PageOf}(DescriptorAddr)) = RW$

Сама таблица смещений обработчиков отображена read-only на уровне PDE (large page); запись в неё из ядра вызывает data abort. Это ядерный аналог IDP-защиты визора.

11 Дополнительная верификация: аппаратная и ядерная реализация целостности

11.1 Trap-фрейм

TrapFrame
$pc : ADDR$ $spsr : ADDR$ $regs : REG \rightharpoonup VAL$

11.2 PAC — аутентификация указателей (ARMv8.3)

Аппаратный аналог целостности указателей из pfi.pdf. Функции ядра подписывают адрес возврата в прологе инструкцией `pacibsp` (PAC, B-ключ, модификатор SP); `paciasp`/`pacibsp` одно-

временно служит landing pad для BTI.

$Sign : ADDR \times PACKey \to SignedAddr$ $Auth : SignedAddr \times PACKey \to ADDR$ $PACKeyOf : Process \rightharpoonup PACKey$ $EnIBOf : Process \to Bool$

PACSign
$lr? : ADDR$ $k? : PACKey$ $lr! : SignedAddr$
$lr! = Sign(lr?,\ k?)$

Свидетельство (дизассемблер): `nt!NtClose`, `nt!NtQueryInformationProcess`, `nt!KiTrackSystemCallEntry` начинаются с `d503237f pacibsp`.

KernelCtxt
$apdbKey : PACKey$ $enIB : Bool$

PACContextSwitch
$\Delta KernelCtxt$ $p? : Process$
$apdbKey' = PACKeyOf(p?)$ $enIB' = EnIBOf(p?)$

Свидетельство: в `nt!KiSystemServiceExit` — `mrs SCTLR_EL1`, переключение бита `EnIB` (bit 30) по флагу процесса, загрузка `APDBKeyLo/Hi` (`S3_0_C2_C1_2/3`), `isb`. Ключи PAC переключаются по процессам.

Аутентификация (полный цикл PAC): `nt!KiTrackSystemCallExit` завершается инструкцией `d50323ff autibsp` непосредственно перед `ret` — пара подпись (`pacibsp`) на входе → аутентификация (`autibsp`) на возврате подтверждена.

11.3 Возврат в R3: ELR_EL1 — точка подмены S-route

SystemCallReturn
$tf : TrapFrame$ $elr' : ADDR$ $spsr' : ADDR$
$elr' = tf.pc$ $spsr' = tf.spsr$

Свидетельство: `msr ELR_EL1,x2`, где $x2 \leftarrow [sp, \#0x140]$ (trap-фрейм), затем очистка регистров/SIMD и `eret` по адресу `fffff8004c236068`. Именно `tf.pc` → `ELR_EL1` есть целевой адрес возврата в R3 — объект подмены S-route. Перед возвратом вызывается `nt!KiPreflightReturnToUserMode` — preflight проверки целостности.

11.4 Теневая и фильтрующая таблицы

ShadowServiceTable
$ntBase : ADDR;\ ntLimit : \mathbb{N}$ $guiBase : ADDR;\ guiLimit : \mathbb{N}$
$ntBase = KiServiceTableBase$ $ntLimit = 489$ $guiLimit = 1493$

Свидетельство: `KeServiceDescriptorTableShadow` — дескриптор[0] = nt (489), дескриптор[1] = `win32k.sys` (`Limit`=0x5D5=1493); `KeServiceDescriptorTableFilter` — отдельный base для win32k (sandbox). Выбор per-thread: `tst x10,#0x80`; `tst x10,#0x200000` в `KiSystemService`.

11.5 Наблюдение атомов: KiTrackSystemCallEntry

Схему `ObserveAtom` уточняем верифицированными деталями: чтение `PreviousMode` (`ldrsb w0,[x8,#0x252]` — поле `_KTHREAD+0x252`); фильтр `nt!KeIsTraceCallbackAllowed`; поиск по ключу-обработчику в `nt!KiSystemServiceTraceTable`; атомарный счётчик (`ldaddal`).

11.6 Граница user/kernel и Probe (DFI)

$MMUserProbe : ADDR$

Свидетельство: `nt!ProbeForRead` — проверка выравнивания (`tst x8,x0`) и диапазона против константы `0x7FFFFFFF0000` (`mov x8,#0x7FFFFFFF0000`; `cmp x9,x8`; `ccmpls`), затем принудительный fault по пробуемому адресу (`ldr w8,[x8]`); `nt!ProbeForWrite` — read/write-touch (`ldrsb`/`strb`) через границы страниц. Гейт по `PreviousMode` находится в вызывающей Nt-функции, а не в самой `Probe*`: в `nt!NtQueryInformationProcess` — `ldr x8,[xpr,#0x988]`; `ldrsb w21,[x8,#0x252]` (чтение `PreviousMode`); `cbnz w21,...` (ветвление в user-путь с пробингом через `0x7FFFFFFF0000`).

ProbeRead
$ptr? : ADDR$ $len? : \mathbb{N}$
$ptr? + len? \leq MMUserProbe$

11.7 Копирование аргументов: Input Set

$Number : SSN \to \mathbb{N}$
— число доп. аргументов на user-стеке

ArgCopy
$ssn? : SSN$ $n! : \mathbb{N}$
$n! = Number(ssn?)$

Свидетельство: счётчик из `Number`-таблицы управляет computed branch `adr x8,CopyEnd; sub x10,x8,x10,lsl#3; br x10` в лестницу `ldr/str` (`nt!KiSystemServiceCopyStart`) — копируется ровно столько qword с user-стека, сколько декларировано во входном множестве вызова.

11.8 Атом syscall на ARM64 — SVC; PAN/UAO

На ARM64 атом системного вызова (вместо x86 `sysenter`/`syscall`) — инструкция `SVC`: `SVC` → вектор исключений → `nt!KiSystemService`. Символ самого вектора в данной сессии не разрешён. **PAN/UAO напрямую не подтверждены**: копирование user-стека выполняется обычным `ldr` (не `ldtr`), переключений `msr pan`/ `msr uao` на пути syscall не обнаружено; разделение user/kernel обеспечивается явной границей `MM_USER_PROBE_ADDRESS` и таблицами страниц.

11.9 Контроль цели косвенного вызова (CFG/CET)

В `nt!KiTrackSystemCallExit` перед `blr x15` вызывается `nt!KscpCfgCheckUserCallTargetEs` — проверка цели вызова по конфигурации CFG (Control Flow Guard): косвенный вызов санкционируется отдельно, поверх PAC/BTI. Дополнительный механизм целостности потока управления.

11.10 W^X и очистка состояния

Свидетельство: кодовая страница `nt!NtClose` — PDE `-RXGA-K-LV` (исполняемая, без W $\Rightarrow$ W^X). На выходе из системного вызова ядро обнуляет SIMD v0–v31 и x1–x15 перед `eret` (DFI: отсутствие утечки данных ядра через регистры).

12 Теоремы

Теорема 1 (Корректность диспетчеризации)**.** *Ограниченный, присутствующий в таблице SSN разрешается в nt-обработчик.*

$$\vdash\ \forall\, ServiceTable;\ Dispatch \mathbin{@} handler! \in NtCode$$

Свидетельство: `KiServiceTable[0]` $\rightarrow$ `nt!NtAccessCheck`, [1] $\rightarrow$ `nt!NtWorkerFactoryWorkerReady`.

Теорема 2 (Отклонение вне диапазона)**.** *Любой SSN не меньше лимита даёт верифицированный код ошибки.*

$$\vdash\ \forall\, ServiceTable;\ InvalidService \mathbin{@} status! = InvalidSvcEC$$

Теорема 3 (Принудительная целостность)**.** *Отображение read-only вызывает отказ при любой записи на страницу таблицы.*

$$\vdash\ \forall\, ServiceTable;\ TableIntegrity \mathbin{@} \text{Protect}(\text{PageOf}(base)) = ReadOnly\ \Rightarrow\ \text{PageOf}(base) \in WriteFault$$

Теорема 4 (DFI исключает AVM)**.** *При выполнении DFI ни одна пара (элемент, адрес) не удовлетворяет предикату обнаружения.*

$$\vdash\ DFI\ \Rightarrow\ \neg\big(\exists\, e : FlowElem;\ ea : EA \mathbin{@} AVMDetected(e,\ ea)\big)$$

Доказательство (набросок). *Раскрывая определения: DFI даёт $(e \mapsto ea) \in Accesses \Rightarrow ea \in CS \cup IS(e)$, что прямо противоречит посылке $ea \notin CS \cup IS(e)$ из $AVMDetected$.* □

Теорема 5 (S-route удерживает управление секвенсора)**.** *После S-route над возвратным атомом слот возврата ядра не равен исходному, поэтому управление после атома не может уйти в API эмулятора.*

$$\vdash\ \forall\, SRoute \mathbin{@} ts.retSlot' \neq ts.retSlot$$

Доказательство (набросок). *$ts.retSlot' = STUB$, причём $STUB$ контролируется визором, откуда $STUB \neq saved! = ts.retSlot$.* □

Теорема 6 (Целостность возврата по PAC)**.** *Адрес возврата, подписанный ключом процесса, нельзя подменить на произвольный адрес без отказа аутентификации.*

$$\vdash\ \forall\, PACSign;\ p? : Process \mathbin{@} PACKeyOf(p?) = k?\ \Rightarrow\ \forall\, a \neq lr? \mathbin{@} Auth(lr!,\ k?) \neq a$$

Теорема 7 (W^X)**.** *Исполняемые страницы ядра недоступны для записи.*

$$\vdash\ \forall\, a : NtCode \mathbin{@}\ \text{Protect}(\text{PageOf}(a)) = ReadOnly\ \wedge\ Exec(\text{PageOf}(a))$$

Теорема 8 (Граница user/kernel)**.** *Любая попытка probes диапазоном, выходящим за `MM_USER_PROBE_ADDRES` отклоняется (fault).*

$$\vdash\ \forall\, ProbeRead \mathbin{@} (ptr? + len? > MMUserProbe)\ \Rightarrow\ (ptr? + len? \in WriteFault)$$

13 Таблица соответствия понятий

Понятие *pfi.pdf*	Схема / определение Z	Якорь в ядре (верифицировано)
Секвенция Cf против потока Xf	`Sequencing`, `Diverges`	буфер визора против `blr x11`
Атом	`ATOM` (свободный тип)	шлюз `syscall/sysenter` в `KiSystemService`
TS (состояние задачи)	`TS`	trap-фрейм, prevMode `_KTHREAD+0x252`
CS / IS / EA / P	компоненты `PFG`	входы декодирования `KiServiceTable`
Инвариант DFI	`DFI`	`ProbeForRead/Write` (prevMode=User)
IDP + анклав	`Redirect`, *enclave*, δ	со стороны визора; ядерный аналог — read-only таблица
Обнаружение AVM	`AVMDetected`	критерий рассогласования InputSet
S-route	`SRoute`, `RestoreReturn`, `STUB`	`[sp,#0x50]` retSlot, `KiSystemServiceCopyEnd+0x38`
Диспетчерская таблица	`ServiceTable`, `Dispatch`, `Decode`	`KiServiceTable`, Shift$(b, \cdot \div 16)$
Лимит сервисов	`ServiceTable.limit` $= 489$	`KiServiceLimit` $= 0 \times 1E9$
Неверный сервис	`InvalidService`	$0 \times C000001C$
Целостность таблицы	`TableIntegrity`, `\Protect`	PDE `-R-GA-K-LV` (read-only large page)
Наблюдение атомов	`ObserveAtom`, `enabled`	`KiTrackSystemCallEntry/Exit` (бит 0), `+0x252`
Целостность указателей (PAC)	`PACSign`, `PACContextSwitch`	`pacibsp`, `APDBKey`, `SCTLR_EL1.EnIB`
Возврат в R3 / S-route	`SystemCallReturn` (`ELR_EL1`)	`msr ELR_EL1,[sp,#0x140]`; `eret`
Preflight	`KiPreflightReturnToUserMode`	проверка перед возвратом в R3
Теневые/фильтр. таблицы	`ShadowServiceTable`	Shadow(win32k 1493)/Filter; `tst` per-thread
BTI / W^X	`Exec`, `\Protect`	PDE `-RXGA-K-LV`; `pacibsp` как landing pad
Очистка состояния	(гигиена / DFI no-leak)	v0–v31, x1–x15 $\to$ 0 перед `eret`
Граница user/kernel (DFI)	`ProbeRead`, `MMUserProbe`	`ProbeForRead/Write`, `0x7FFFFFFF0000`
Input Set (копирование арг.)	`ArgCopy`, `Number`	`br x10` в лестницу `ldr/str`, `CopyStart`
Атом syscall (ARM64)	`SVC` $\to$ `KiSystemService`	вместо x86 `sysenter`/`syscall`
PAN/UAO	— (не подтверждено)	граница via `MM_USER_PROBE_ADDRESS`
PAC: аутентификация	`Auth`	`autibsp` перед `ret` (`KiTrackSystemCallExit`)
DFI: гейт PreviousMode	(Nt-вызывающий)	`ldrsb [+0x252]`; `cbnz` в `NtQueryInformationProcess`
CFG (цель вызова)	—	`KscpCfgCheckUserCallTargetEs` перед `blr`

Конец спецификации.