Академия Федеральной службы охраны Российской Федерации

Подход к реализации системы верифицированного исполнения программного кода

к. т. н. Козачок Александр Васильевич
Кочетков Евгений Викторович

1 декабря 2017

Структура доклада

1. Характеристика предметной области исследования
2. Формальный логический язык задания функциональных требований к программному коду
3. Аксиомы безопасного исполнения программного кода
4. Система безопасного исполнения программного кода (СБИПК)
   - Базовая модель СБИПК
   - Алгоритмическое представление СБИПК
5. Подход к практической реализации подсистемы мониторинга с использованием полносистемного эмулятора Qemu

Структура доклада

Эффективность обнаружения вредоносных программ антивирусными средствами

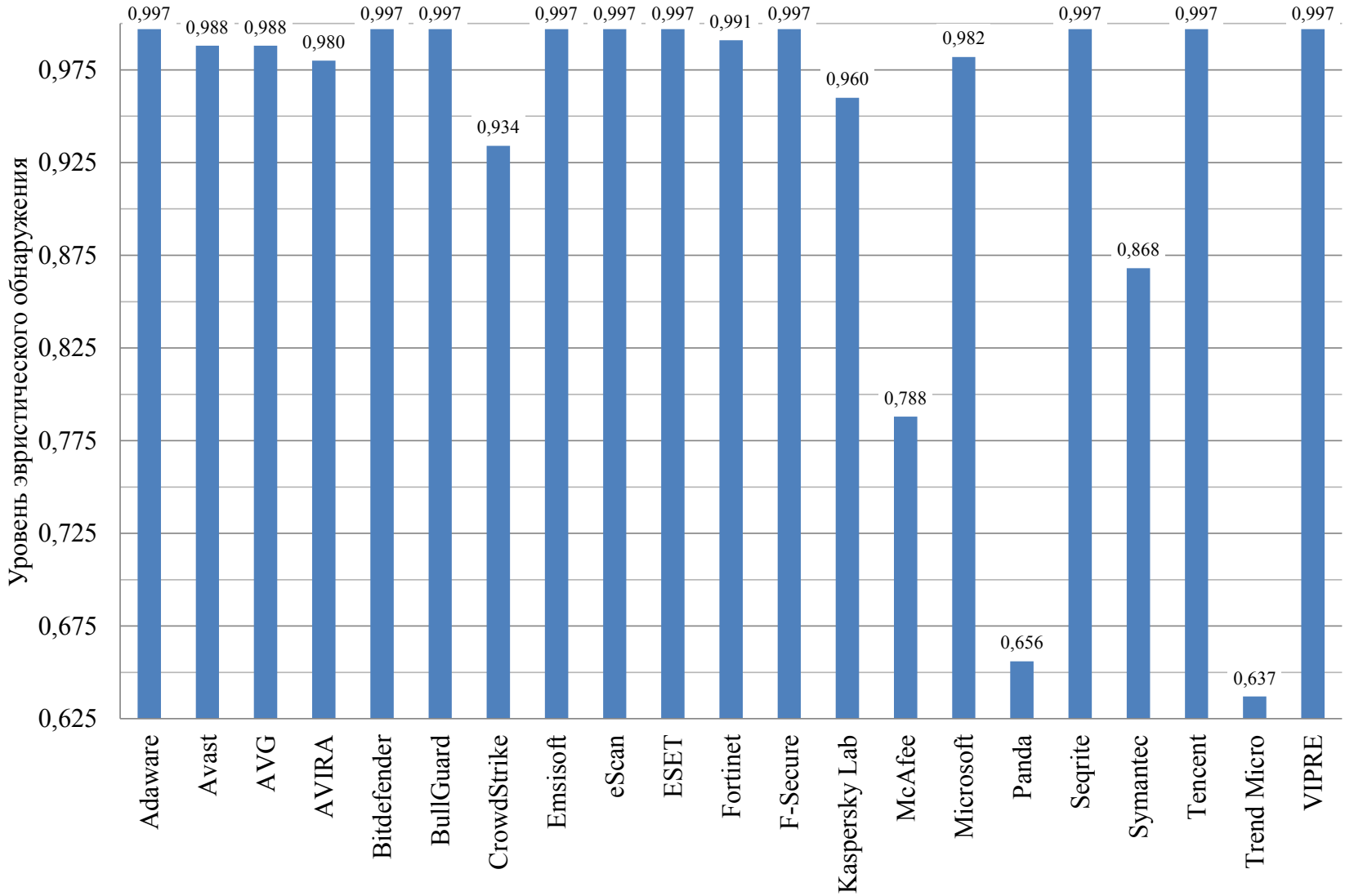

Рис. 1. Значение уровня обнаружения вредоносных программ антивирусными средствами без использования

Структура доклада

Формальный логический язык задания функциональных требований к программному коду I

Субъекты:

P^1 – системный процесс

P^2 – привилегированный процесс

P^3 – пользовательский процесс

Объекты:

M^1 – адресное пространство системного процесса

M^2 – адресное пространство другого процесса

M^3 – собственное адресное пространство процесса

E^1 – исполняемые файлы

E^2 – системные каталоги и конфигурация системы

E^3 – другие файлы и каталоги

E^4 – системные библиотеки

E^5 – собственные файлы и каталоги

D^1 – устройства вывода

D^2 – устройства ввода

D^3 – драйвера устройств

N^1 – сервисы узлов глобальной сети

N^2 – сервисы узлов локальной сети

N^3 – локальные сетевые сервисы

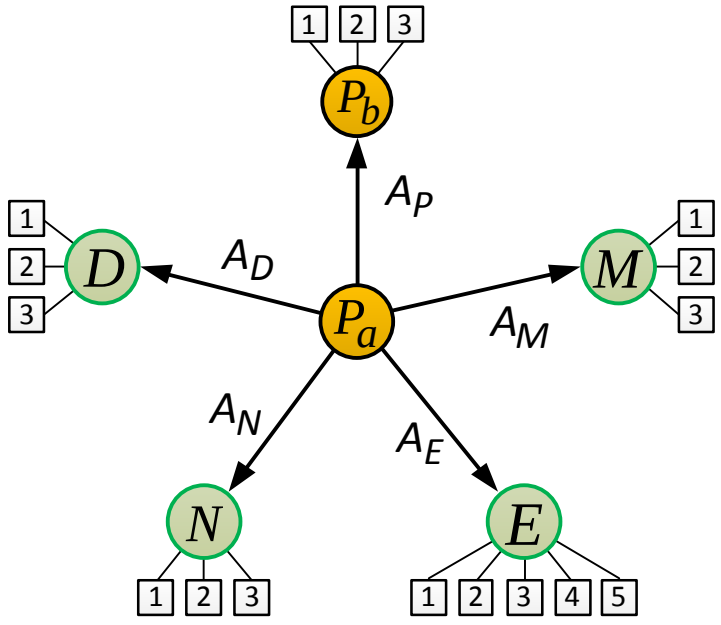

Рис. 2. Субъектно-объектная модель функционирования процесса в ОС

Действия:

c	Create	r	Read
o	Open		
d	Delete	w	Write

Формальный логический язык задания функциональных требований к программному коду II

Определение 1 (Описание функционирования процесса в ОС)

Под описанием функционирования процесса в ОС S_i будем понимать последовательность действий $\{s_i\}_{i\in\mathbb{T}}$ выполняемых субъектом $p_i^{k_p}$. Каждое действие описывается тройкой элементов:

$$s_i = p_i^{k_p} \xrightarrow{a_o} o_y^{k_o}, \tag{1}$$

где $p_i^{k_p}$ – субъект категории k_p с идентификатором i; $a_o \in A_o$ – некоторое действие субъекта по отношению к объекту; $o_y^{k_o}$ – некоторый объект категории k_o с идентификатором y.

Множество всех действий процесса в ОС обозначим как:

$$\Omega = \bigcup_{i\in\mathbb{T}} s_i. \tag{2}$$

Формальный логический язык задания функциональных требований к программному коду III

Определение 2 (Аксиома безопасного исполнения программного кода)

Под аксиомой безопасного исполнения программного кода AX будем понимать описание параметров разрешенных действий $\{s_k\}_{k\in\mathbb{S}}$ для процесса $p_i^{k_p}$ во время его функционирования S_i, позволяющее исключить потенциальную возможность выполнения программой вредоносных действий:

$$AX_j = \bigcup_k s_k, k \in \mathbb{S}_j, \tag{3}$$

где $\mathbb{S}_j$ – множество индексов безопасных действий для процесса $p_i^{k_p}$ при заданных функциональных требованиях FR_j.

Пример аксиомы:

"Запрет на создание пользовательским процессом привилегированного или системного процесса"

$$AX_j = \neg EF[p^3 \xrightarrow{c} p^2] \vee \neg EF[p^3 \xrightarrow{c} p^1] \tag{4}$$

Формальный логический язык задания функциональных требований к программному коду IV

Определение 3 (Модель безопасного исполнения программного кода)

Моделью безопасного исполнения программного кода SM будем называть совокупность аксиом безопасного исполнения программного кода AX_j для процесса $p_i^{k_p}$:

$$SM = \bigcup_{j \in M} AX_j, \tag{5}$$

где M – множество индексов аксиом безопасного исполнения программного кода.

Определение 4 (Свойство безопасности исполнения программного кода)

Процесс p обладает свойством безопасности Λ с учетом заданных функциональных требований FR при выполнении следующего условия:

$$p \notin \Lambda \iff (\exists i : (S_i \in \Psi) \notin SM), \tag{6}$$

где Ψ – множество всех описаний функционирования процесса p.

$$|\Psi| = |\Omega|! = 157! = 1{,}17 \cdot 10^{278}. \tag{7}$$

Формальный логический язык задания функциональных требований к программному коду V

Теорема (О безопасности процессов)

Процесс обладает свойством безопасности Λ тогда и только тогда, когда для всех последовательностей действий процесса S_i из Ψ, заданных как $s_1, s_2, \ldots, s_l$, можно доказать Γ:

$$p \in \Lambda \iff \forall S_i \in \Psi : s_1, s_2, \ldots, s_l \vdash \Gamma. \tag{8}$$

$\Gamma = FR \wedge RS \wedge AZ$, FR – множество функциональных требований, RS – множество запрещенных действий, AZ – множество разрешенных действий в рамках модели безопасного исполнения SM.

$$FR = (f_1 \to f_2) \wedge \ldots \wedge (f_{n-1} \to f_n) = (\neg f_1 \vee f_2) \wedge \ldots \wedge (\neg f_{n-1} \vee f_n) \tag{9}$$

$$RS = \neg r_1 \wedge \neg r_2 \wedge \neg r_3 \wedge \ldots \wedge \neg r_k \tag{10}$$

$$AZ = (a_0 \vee a_1 \vee a_2 \vee a_3 \vee \ldots \vee a_m) \tag{11}$$

n – число функциональных ограничений, m – множество действий, разрешенных в рамках модели безопасного исполнения кода SM, k – множество действий, запрещенных в рамках модели безопасного исполнения кода SM.

Структура доклада

Базовые аксиомы безопасного исполнения программного кода I

Аксиома 1 (Аксиома безопасного выполнения действия "Create")

$\neg(p^* \xrightarrow{c} p^*)$ – *запрет на создание процессов*

$p^* \xrightarrow{c} m^3$ – *выделение памяти только в своем адресном пространстве процесса*

$p^* \xrightarrow{c} e^5$ – *доступ только к собственным файлам и каталогам*

$\neg(p^* \xrightarrow{c} n^*)$ – *запрет на создание соединений*

$\neg(p^* \xrightarrow{c} d^*)$ – *запрет на создание устройств*

Аксиома 2 (Аксиома безопасного выполнения действия "Open")

$\neg(p^* \xrightarrow{o} p^*)$ – *запрет на открытие других процессов*

$p^* \xrightarrow{o} e^{4,5}$ – *доступ только к системным бибилотекам и собственным файлам и каталогам*

$\neg(p^* \xrightarrow{o} d^*)$ – *запрет на открытие устройства*

Базовые аксиомы безопасного исполнения программного кода II

Аксиома 3 (Аксиома безопасного выполнения действия "Read")

$\neg(p^* \xrightarrow{r} p^*)$ – *запрет на чтение информации о других процессах*

$p^* \xrightarrow{r} m^3$ – *чтение памяти в рамках своего адресного пространства процесса*

$p_i^* \xrightarrow{o} e_j^4 \wedge p^* \xrightarrow{r} e^4$ – *чтение файлов системных бибилотек после их открытия*

$p_i^* \xrightarrow{c \vee o} e_j^5 \wedge p_i^* \xrightarrow{r} e_j^5$ – *чтение собственных файлов и каталогов, созданных или открытых процессом*

$\neg(p^* \xrightarrow{r} n^*)$ – *запрет на работу с сетевой подсистемой*

$\neg(p^* \xrightarrow{r} d^*)$ – *запрет на работу устройствами*

Базовые аксиомы безопасного исполнения программного кода III

Аксиома 4 (Аксиома безопасного выполнения действия "Write")

$p^* \xrightarrow{w} m^3$ – *запись в память в рамках своего адресного пространства процесса*

$p_i^* \xrightarrow{c \vee o} e_j^5 \wedge p^* \xrightarrow{w} e_j^5$ – *запись в файл, который был открыт или создан данным процессом*

$\neg(p^* \xrightarrow{w} n^*)$ – *запрет на работу с сетевой подсистемой*

$\neg(p^* \xrightarrow{w} d^*)$ – *запрет на работу с устройствами*

Аксиома 5 (Аксиома безопасного выполнения действия "Delete")

$p_i^* \xrightarrow{d} p_i^*$ – *процесс может завершить свою работу*

$p_i^* \xrightarrow{c} e_j^5 \wedge p_i^* \xrightarrow{d} e_j^5$ – *процесс может закрывать (завершать) объекты, им порожденные* $(\forall o \in O' = \{O \backslash M\})$

Структура доклада

1 Характеристика предметной области исследования

2 Формальный логический язык задания функциональных требований к программному коду

3 Аксиомы безопасного исполнения программного кода

4 Система безопасного исполнения программного кода (СБИПК)
- Базовая модель СБИПК
- Алгоритмическое представление СБИПК

5 Подход к практической реализации подсистемы мониторинга с использованием полносистемного эмулятора Qemu

Базовая модель системы безопасного исполнения программного кода

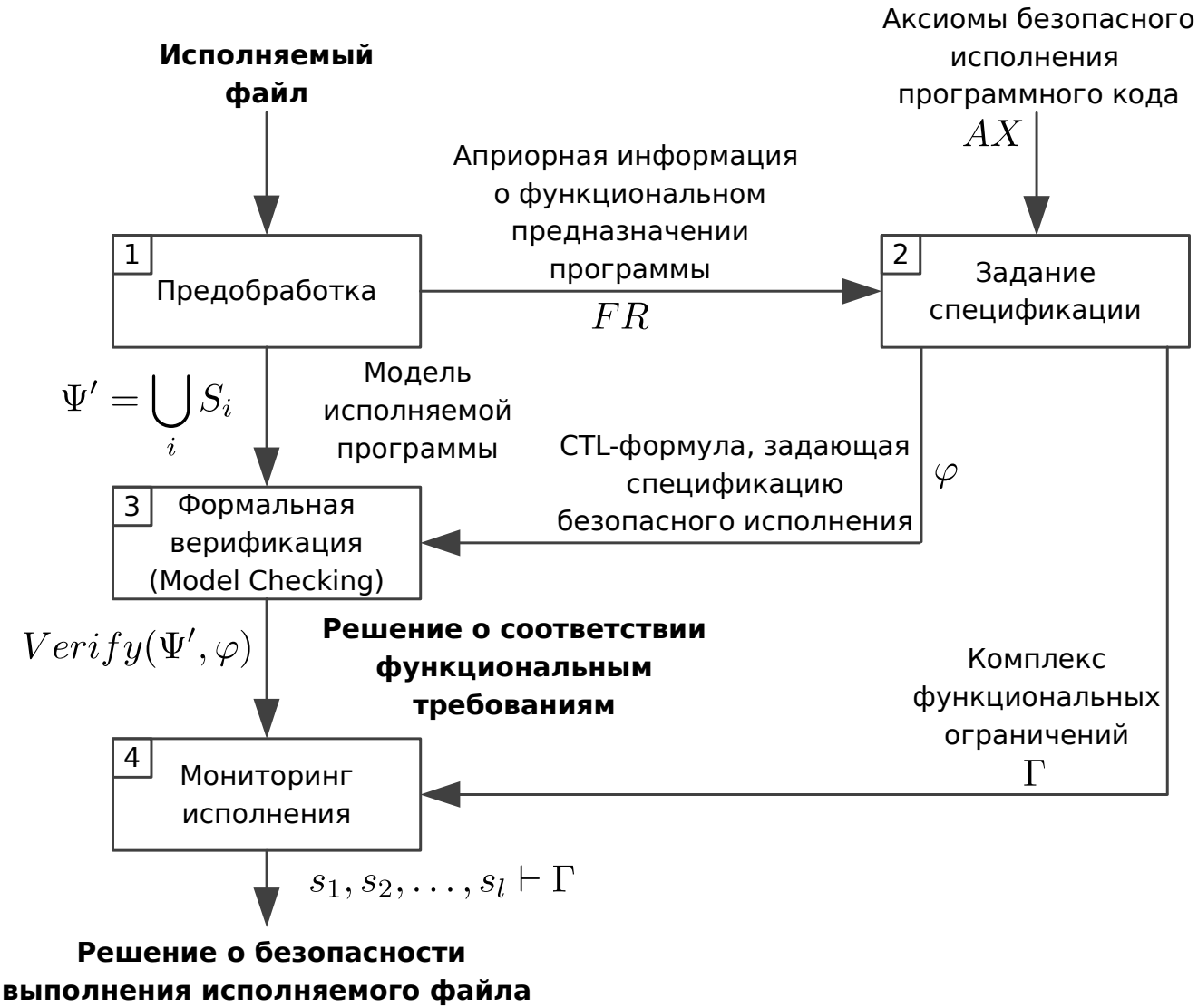

Рис. 3. Структурная модель системы безопасного исполнения программного кода

Алгоритмическое представление системы безопасного исполнения программного кода

```
1  Function SecureCodeExec(B, FR, AX, FCTT)
       Data: B: бинарный исполняемый файл
       FR: функциональные требования, заданные на формальном языке
       AX: аксиомы безопасного исполнения кода
       FCTT: таблица соответствия функций и действий в ОС
       Result: Decision: решение о безопасности исполняемого файла B
       /* Проверка ограничений                                    */
2      if checkMinCriterias(B) ≠ true then
3          return false
4      end
       /* Построение модели исполняемого файла                    */
5      M ← buildModel(B,FCTT)
       /* Статический этап                                        */
6      φ ← buildSpec(FR)
7      if Verify(M,φ) ≠ true then
8          return false
9      end
       /* Динамический этап                                       */
10     if SecurityMonitor(B,AX,FR,M,FCTT) ≠ true then
11         return false
12     end
13 return true
```

Рис. 4. Алгоритм функционирования системы безопасного исполнения программного кода

Алгоритм предварительной обработки и построения модели исполняемого файла

```
1  Function buildModel(B, FCTT)
       Result: {S,R}
2      T ← ∅
3      R ← ∅
4      S ← {s_0}
5      D ← Dissasembler(B)
6      while Cover(D,T) ≤ C_min do
7          repeat
8              data ← Mutation(data,T)
9              trace ← Fuzzer(B,data)
10         until trace ∉ T
11         T ← T ∪ trace
12         C ← split(trace)
13         seq ← {s_0}
14         foreach c ∈ C do
15             sysCall ← ExtractSysCall(c)
16             if sysCall = ∅ then
17                 continue
18             end
19             s ← TranslateSysCall(sysCall, FCTT)
20             seq ← seq‖s
21             if s ∉ S then
22                 S ← S ∪ s
23             end
24         end
25         for i ← 0 to | seq | −2 do
26             s_1 ← seq[i]
27             s_2 ← seq[i + 1]
28             r ← (s_1,s_2)
29             if r ∉ R then
30                 R ← R ∪ r
31             end
32         end
33     end
34 return {S,R,L}
```

Рис. 5. Преобразование исполняемого файла в модель, пригодную для верификации

Алгоритм преобразования системного вызова в действие в операционной системе

1 **Function** *TranslateSysCall*($sysCall, FCTT$)
  Data: $sysCall$: системный вызов
  $FCTT$: таблица соответствия функций и действий в ОС
  Result: k_p: категория субъекта
  a: действие в операционной системе
  o: тип объекта
  k_o: категория объекта
2 $k_p = getProcessCategory()$
3 **foreach** $record \in FCTT$ **do**
4 **if** $record.name = sysCall.name$ **then**
5 $a \leftarrow record.action$
6 $o \leftarrow record.object$
7 $k_o \leftarrow getObjInfo(sysCall)$
8 **return** (k_p, a, o, k_o)
9 **end**
10 **end**
11 **return** $\varnothing$

Рис. 6. Функция преобразования системного вызова в действие в операционной системе.

Структура доклада

Фрагмент API Call Translate Table (APICTT)

Объект	Действие	API
Процесс (P)	Create	CreateProcess, NumaCreateThread, CreateFiber, CreateRemoteThread
	Read	GetProcessInformation, GetThreadPriority, GetThreadInformation, GetProcessVersion
	Delete	TerminateProcess, TerminateThread, ExitProcess, ExitThread
Память (M)	Create	VirtualAllocEx, HeapAlloc, HeapCreate, CreateFileMapping, GlobalAlloc
	Open	GlobalHandle, LocalHandle, OpenFileMapping
	Read	ReadProcessMemory, GetProcessHeap, GlobalMemoryStatusEx, HeapWalk

Таблица 1. Фрагмент таблицы APICTT

Фрагмент Syscall Translate Table (SCTT)

Объект	Действие	System call
Внешняя память (E)	Create	NtCreateFile, NtCreateDirectoryObject, NtCreateSymbolicLinkObject
	Read	NtQueryAttributesFile, NtReadFile, NtQueryDirectoryFile
Устройства (D)	Create	NtAddDriverEntry, NtLoadDriver
	Read	NtEnumerateDriverEntries, NtQueryDriverEntryOrder
	Delete	NtDeleteDriverEntry, NtUnloadDriver
	Write	NtModifyDriverEntry, NtSetDriverEntryOrder

Таблица 2. Фрагмент таблицы SCTT

Реализация подсистемы мониторинга с использованием полносистемного эмулятора Qemu

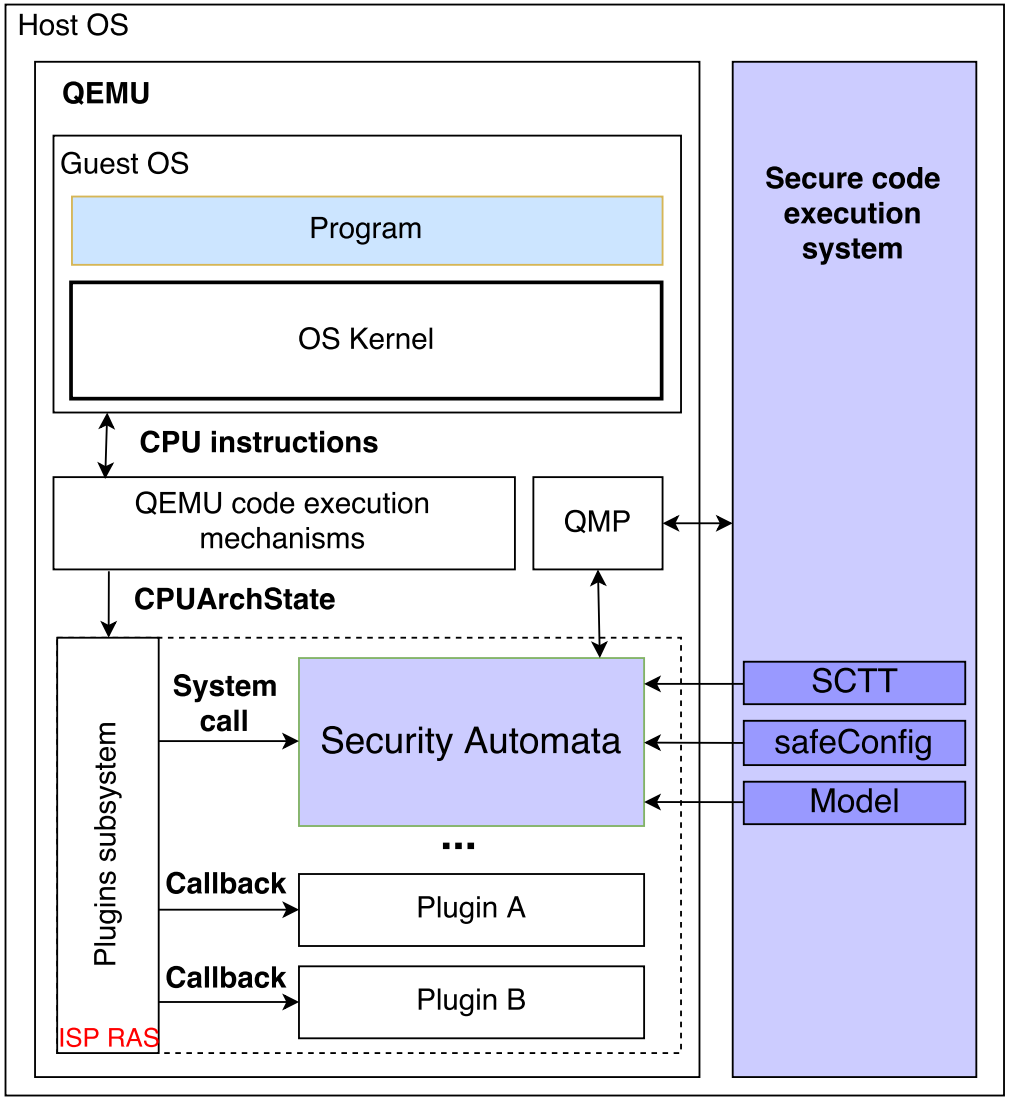

Рис. 7. Подсистема мониторинга исполнения программного кода с использованием плагинов, разработанных Институтом системного программирования РАН для эмулятора Qemu

Модельный пример работы SecurityMonitor

1. Исходные данные

AX

$FR = p_i^* \xrightarrow{r} e^3 \rightarrow (\neg p_i^* \xrightarrow{w} e^5) = (\neg p_i^* \xrightarrow{r} e^3) \vee (\neg p_i^* \xrightarrow{w} e^5)$

$\Gamma = FR \wedge RS \wedge AZ = ((\neg p_i^* \xrightarrow{r} e^3) \vee (\neg p_i^* \xrightarrow{w} e^5)) \wedge RS \wedge AZ$

2. Последовательность извлеченных системных вызовов

NtCreateFile(), NtReadFile(), NtCreateFile(), NtWriteFile(), ...

3. Последовательность действий процесса в ОС

$p_i^* \xrightarrow{o} e^3, p_i^* \xrightarrow{r} e^3, p_i^* \xrightarrow{o} e^5, p_i^* \xrightarrow{w} e^5, \ldots$

4. Оценка безопасности действий процесса в ОС

1. $p_i^* \xrightarrow{o} e^3 \vdash \Gamma$
2. $p_i^* \xrightarrow{o} e^3, p_i^* \xrightarrow{r} e^3 \vdash \Gamma$
3. $p_i^* \xrightarrow{o} e^3, p_i^* \xrightarrow{r} e^3, p_i^* \xrightarrow{o} e^5 \vdash \Gamma$
4. $p_i^* \xrightarrow{o} e^3, p_i^* \xrightarrow{r} e^3, p_i^* \xrightarrow{o} e^5, p_i^* \xrightarrow{w} e^5 \nvdash \Gamma$