УДК 004.056.57

ПРОАКТИВНАЯ ЗАЩИТА ОТ ПРОГРАММ-ВЫМОГАТЕЛЕЙ В ОС СЕМЕЙСТВА LINUX

И. Б. Бережной[1)], А. В. Садовский[2)]

[1)] *НИИ прикладных проблем математики и информатики, Белорусский государственный университет, Минск, Беларусь, BerazhnyIB@bsu.by*
[2)] *Белорусский государственный университет, Минск, Беларусь*

В статье рассматривается вопрос защиты от программ-вымогателей (ransomware) в ОС семейства Linux. Представлена общая схема функционирования таких программ и рассмотрена роль источника случайности в этой схеме. Предложен способ проактивной защиты от программ-вымогателей на основе контроля системных генераторов псевдослучайных чисел.

Ключевые слова: защита информации; программы-вымогатели; псевдослучайные числа; модификация ядра ОС Linux.

PROACTIVE PROTECTION AGAINST RANSOMWARE IN LINUX OS

I. B. Berezhnoy[a)], A. V. Sadovski[b)]

[a)] *Research Institute for Applied Problems of Mathematics and Informatics, Belarusian State University, Minsk, Belarus, BerazhnyIB@bsu.by*
[b)] *Belarusian State University, Minsk, Belarus*

The article considers the issue of protection against ransomware in Linux family OS. The general scheme of functioning of such programs is presented and the role of the source of randomness in this scheme is considered. A method to proactive protection against ransomware based on control of system random number generators is proposed.

Keywords: information security; ransomware; pseudo-random numbers; Linux kernel modification.

1. Введение

Программа-вымогатель (англ. ransomware, от слов ransom — выкуп и software — программное обеспечение) — тип вредоносного программного обеспечения (ВПО), который используется в целях вымогательства путем зашифровывания данных, ценных для пользователя компьютерной

системы, и последующего требования выкупа за их восстановление. Концепция программы-вымогателя, шифрующей пользовательские файлы, была представлена еще в 1996 году на конференции IEEE Security & Privacy conference [1], однако наибольшее распространение данный тип ВПО получил после 2013 года в связи с развитием криптовалют как трудноотслеживаемых способов платежей [2]. По данным компании Chainalysis суммарный ежегодный объем платежей на криптовалютные кошельки вымогателей с 2020 года превышает 500 миллионов долларов США, а за 2024 год превысил 800 миллионов долларов США [3].

Программы-вымогатели разных семейств отличаются способами проникновения, особенностями горизонтального распространения и закрепления в системе, некоторым дополнительным функционалом. Для выявления работы программ-вымогателей на конкретном компьютере системы класса MDR (Managed Detection and Response) используют ряд правил детектирования, к наиболее распространенным из которых относятся [4]:

активная работа с реестром;

наличие в оперативной памяти вредоносного кода;

запуск подозрительных сервисов и программ, особенно удаленным пользователем;

доступ к подозрительным узлам и URL;

получение дампов памяти, связанных с сервисом безопасности LSASS;

добавление пользователей и повышение их привилегий;

работа с сетевыми устройствами.

Следует отметить, что применение таких правил требует предварительной настройки на конкретный образец ВПО, ориентируется на вспомогательный функционал, который может отсутствовать, и мало эффективно в случае обфускации кода ВПО и использования в ВПО уязвимостей нулевого дня.

Основные рекомендации для борьбы с программами-вымогателями, предлагаемые на текущий момент, представляют собой только общие рекомендации по борьбе с ВПО произвольного типа с повышенным акцентом на необходимость регулярного резервного копирования информации [5]. Другими словами, в качестве общей стратегии вместо способов защиты предлагается рассматривать способы минимизации последствий.

2. Схема криптографического ядра ransomware

В качестве основной компоненты всех программ-вымогателей выступает блок шифрования пользовательских данных на псевдослучайных ключах шифрования, генерируемых в ВПО. Источником таких псевдослучайных значений могут выступать как самостоятельно разработанные

хакерами генераторы псевдослучайных чисел (ГПСЧ), так и ГПСЧ, реализованные в операционной системе.

В случае самостоятельной реализации ГПСЧ в программе-вымогателе исходный код ВПО часто содержит ошибки, приводящие к понижению энтропии начальных значений ГПСЧ и, как результат, к возможности восстановления зашифрованных данных [6].

Поэтому у самых эффективных семейств программ-вымогателей криптографические ядра, непосредственно отвечающие за шифрование файлов, соответствуют следующей схеме [7]:

1) злоумышленник генерирует пару открытого и личного ключей для асимметричного шифрования и загружает в ВПО (криптер) открытый ключ, после чего криптер некоторым образом доставляется на компьютер-жертву;

2) при запуске криптер осуществляет поиск файлов, предположительно представляющих ценность для владельца компьютера, по определенным признакам, чаще всего — по расширению имен файлов;

3) для каждого найденного файла с помощью криптографически сильного ГПСЧ, реализованного в операционной системе, генерируется ключ симметричного шифрования, затем с помощью этого ключа файл зашифровывается (либо полностью, либо частично — в случае большого объема файла), после чего к нему дописывается блок с использованным ключом, зашифрованный на открытом ключе злоумышленника;

4) жертва после оплаты выкупа получает программное обеспечение для расшифрования файлов, содержащее личный ключ злоумышленника (декриптер);

5) декриптер для каждого файла расшифровывает на личном ключе злоумышленника дописанный блок, извлекает из него ключ симметричного шифрования и с его помощью расшифровывает исходный файл.

Указанная схема обладает следующими преимуществами для злоумышленника:

1) ни при каких обстоятельствах закрытый ключ злоумышленника не передается жертве до оплаты выкупа;

2) злоумышленнику не требуется от жертвы никаких дополнительных данных, кроме идентификатора использованной ключевой пары;

3) жертве от злоумышленника требуется один файл фиксированного размера вне зависимости от размера зашифрованных данных;

4) при соблюдении известных требований на размер ключа восстановление личного ключа по открытому для жертвы невозможно;

5) при симметричном шифровании разных файлов использованные ключи не могут совпадать; как следствие, существенно затруднен криптоанализ алгоритма шифрования файлов либо процесс восстановления личного ключа.

Основное тонкое место описанной схемы действий криптографического ядра программы-вымогателя — получение данных от системного ГПСЧ. Во время работы ransomware случайный ключ размером не менее 16 байтов требуется для каждого подходящего файла. А так как количество таких файлов, подходящих по шаблоны поиска, обычно исчисляется тысячами, то, соответственно, за короткий промежуток времени с ГПСЧ потребуется большое количество псевдослучайных данных. При этом конкретный поток криптера не сможет продолжить свою работу до получения псевдослучайного ключа шифрования.

Таким образом, если имеется возможность повлиять на работу системного ГПСЧ и затруднить/приостановить его работу, то функционирование потоков процесса ransomware будет заморожено до момента шифрования пользовательских данных. В случае, если дальнейшие действия позволят остановить процесс программы-вымогателя, то такой подход может позволить полностью предотвратить шифрование пользовательских файлов (или потерять только первых *N* файлов, зашифрованных к моменту срабатывания блокировки), то есть осуществить проактивную защиту.

3. Функционирование ГПСЧ в ОС семейства Linux

ГПСЧ в ОС Linux использует аппаратные события, обнаруженные ядром Linux, в качестве источников шума для подачи данных на детерминированный генератор псевдослучайных чисел. Начиная с версии 5.18.1 ядра Linux ГПСЧ функционирует согласно схеме, представленной на рис. 1 [8].

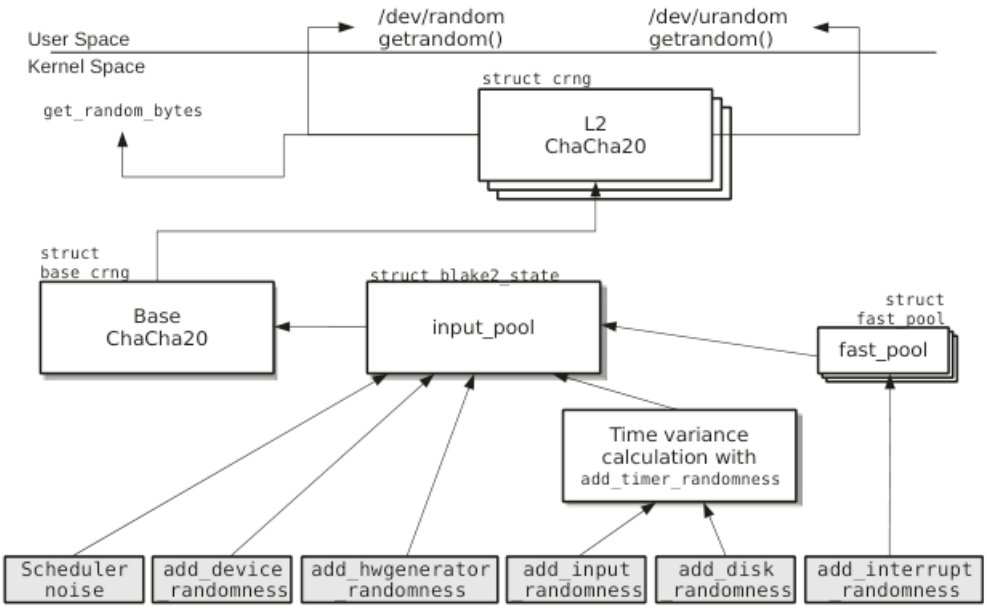

Рис. 1. Архитектура ГПСЧ в ОС Linux

ГПСЧ использует один пул энтропии — «входной пул» (англ. «input pool»). Его цель — сбор и накопление энтропии, предоставляемой различными источниками шума.

«Базовый» ChaCha20-генератор инициализируется на основе значения из «входного пула».

Набор вторичных ChaCha20-генераторов, по одному на каждый процессор, инициализируется «базовым» и предоставляет псевдослучайные данные через интерфейсы как режима пользователя (/dev/urandom, /dev/random, системный вызов getrandom()), так и режима ядра (функция get_random_bytes()).

Интерфейсы можно разделить на 2 класса:

1) блокирующие: /dev/random, getrandom();

2) неблокирующие: /dev/urandom, getrandom(GRND_INSECURE), get_random_bytes().

При выполнении запроса к блокирующему интерфейсу при недостаточной оценке энтропии input_pool интерфейс переходит в режим ожидания до накопления необходимой энтропии. Извлечение данных сопровождается снижением оценки энтропии input_pool.

Неблокирующие интерфейсы формируют запрошенное количество псевдослучайных данных независимо от того, достаточно ли энтропии у input_pool или ChaCha20-генератора. Они работают за счет периодической инициализации симметричного потокового алгоритма ChaCha20 данными из input_pool.

Вышеуказанные интерфейсы реализованы в файле random.c, являющемся частью ядра ОС Linux [9]. Соответственно, для влияния на данные интерфейсы требуется возможность внесения изменений в ядро ОС Linux.

4. Возможности модификации ядра Linux

ОС Linux – операционная система с открытым исходным кодом, а значит, предоставляет разработчикам полный контроль над системой, включая возможность модификации и пересборки любых компонентов ядра, в том числе и файла random.c.

Однако у данного подхода имеются следующие недостатки:

1) версия ядра должна соответствовать версии дистрибутива ОС;

2) трудоемкая процедура конфигурации и компиляции;

3) при внесении даже минимальных изменений в конкретном модуле требуется пересборка всего ядра.

Еще одним способом является использование *Linux Security API* — специального интерфейса, предназначенного для интеграции сторонних модулей безопасности, таких как SELinux, AppArmor и другие [10]. *Linux*

Security API позволяет модулям безопасности установить свои обработчики для системных вызовов, изучать контекст операции и централизованно принимать решение о её разрешении или запрете, работая при этом параллельно с системой разграничения доступа ядра ОС.

Однако, у *Linux Security API* есть пара важных ограничений:

1) модули безопасности не могут быть загружены динамически, являются частью ядра и требуют его пересборки;

2) в большинстве случаев одновременно может функционировать только один модуль безопасности.

При этом запрет на динамическую загрузку принципиальный: модуль безопасности должен быть частью ядра, чтобы обеспечивать безопасность постоянно, с момента загрузки.

Альтернативой без необходимости сборки собственного ядра является использование инструментов для отладки и анализа производительности функций ядра: FTrace и KProbe [11].

FTrace (Function Tracker) – это встроенная инфраструктура ядра Linux, предназначенная для отслеживания вызовов функций и трассировки их потоков выполнения. Изначально созданная для профилирования, она позволяет добавлять собственные хуки для экспорта и мониторинга функций ядра, что дает возможность отслеживать частоту и длительность вызовов функций, отображать графы вызовов, фильтровать интересующие функции по шаблонам и т. д.

Технология FTrace работает за счет поиска указанной сигнатуры функции в /proc/kallsyms. Обязательным условием для ее нахождения является то, что она должна быть экспортирована в коде во время сборки, либо добавлена в /proc/kallsyms напрямую.

Таким образом, FTrace предоставляет удобный функционал для внесения своих изменений в системный ГПСЧ или логирования связанных событий, таких как заполнение пула энтропии или вызов интерфейса получения псевдослучайных данных из пространства пользователя.

Главный недостаток FTrace – необходимость поддержки соответствующих FTrace опций конфигурирования при сборке ядра ОС. Тем не менее, ядра, используемые популярными дистрибутивами Linux, все необходимые опции в себе содержат, так как они не влияют на производительность и полезны при отладке. Однако полная универсальность у технологии отсутствует.

Технология KProbe, существующая и улучшающаяся с 2002 года, представляет собой специализированное API, в первую очередь предназначенное для отладки и трассирования ядра. В отличие от FTrace, эта технология позволяет устанавливать пред- и постобработчики не только на вход и возврат из функции, но и для *любой инструкции* в ядре.

Обработчики получают доступ к регистрам и могут их изменять. Таким образом, возможно получить как мониторинг, так и возможность влиять на дальнейший ход работы целевых функций ядра.

При этом, так как KProbe основывается на прерываниях и управляет регистрами процессора, возможности собственных обработчиков ограничены (нельзя выделить много памяти, заниматься вводом/выводом), а написание собственных обработчиков может иметь высокую техническую сложность (например, для получения аргументов перехватываемой функции надо их извлекать из заранее известных регистров и стека).

5. Экспериментальная проверка

Была проведена экспериментальная проверка возможности внесений изменений в работу ГПСЧ ОС Debian 12 посредством хука FTrace.

В качестве вносимого изменения рассматривалась дополнительная задержка для каждого вызова функции get_random_bytes_user(), которая используется в интерфейсах обращения к /dev/random из пространства пользователя.

Эксперимент проводится в несколько шагов.

1. Создание теста работы /dev/random, который представляет собой 1000 обращений на чтение 8MB из /dev/random. После чего выводится график и статистика времени выполнения.

2. Создание собственного обработчика, который будет вводить искусственную задержку в 50 мс. В качестве инструмента задержки использован mdelay (активное ожидание процессора).

3. Проведение теста до включения собственного обработчика.

4. Внедрение обработчика в ядро посредством команды insmod.

5. Проведение теста с активным обработчиком.

6. Сравнение результатов теста до и после включения обработчика.

На рис. 2 и 3 представлены результаты теста до включения обработчика. На рис. 4 и 5 представлены результаты теста после включения обработчика.

```
=== Benchmark Results ===
Samples: 1000
Chunk size: 8 MB
Average time: 17352.8 µs
Minimum time: 15216 µs
Maximum time: 28153 µs
Average throughput: 483.416 MB/s
```

Рис. 2. Метрики теста до встраивания обработчика

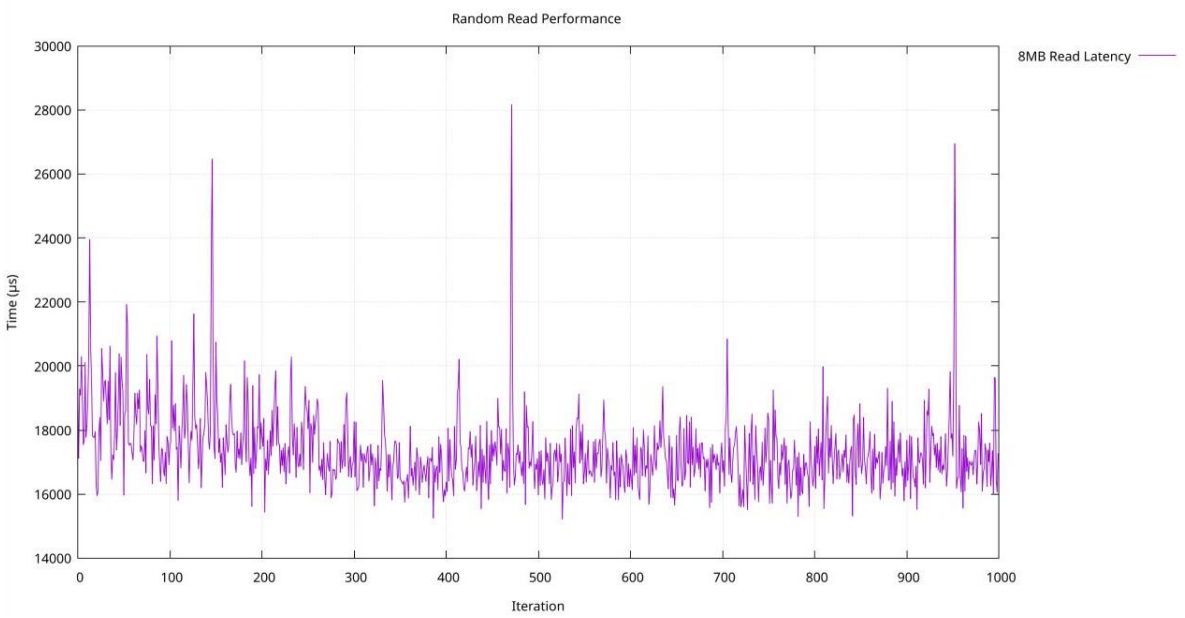

Рис. 3. Визуализация времени выполнения теста до встраивания обработчика

```
=== Benchmark Results ===
Samples: 1000
Chunk size: 8 MB
Average time: 65609.6 µs
Minimum time: 62418 µs
Maximum time: 73233 µs
Average throughput: 127.856 MB/s
```

Рис. 4. Метрики теста после встраивания обработчика

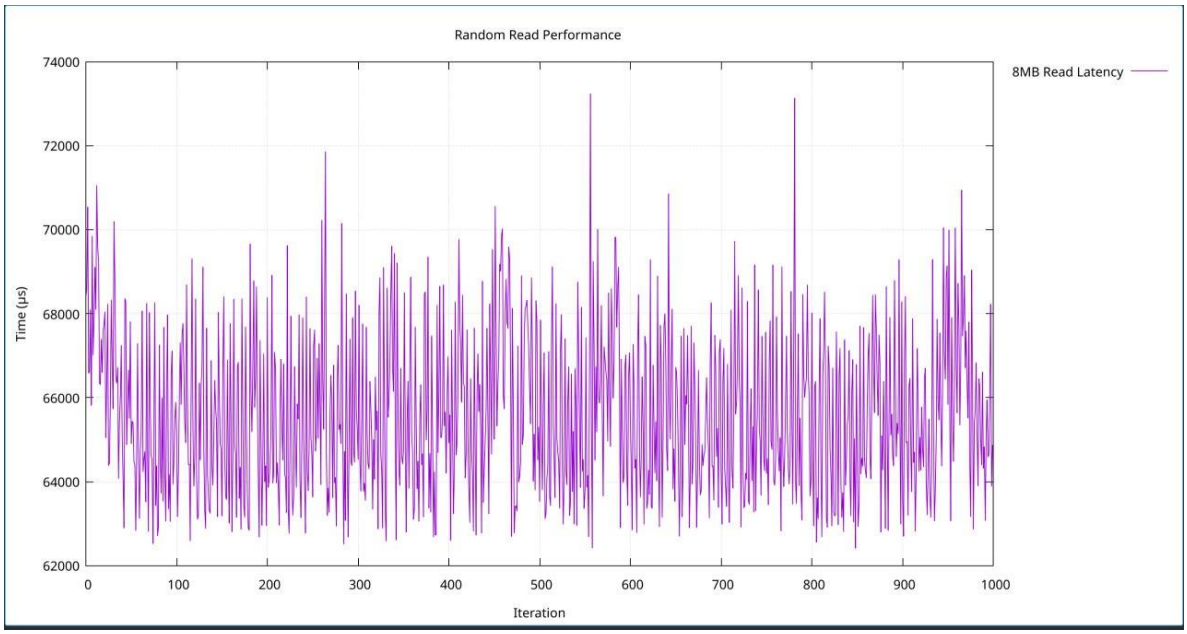

Рис. 5. Визуализация времени выполнения теста после встраивания обработчика

Среднее время выполнения чтения увеличилось с 17,353 мс до 65,610 мс, что примерно совпадает с ожидаемыми результатами (добавление 50 мс). Отличие может быть обусловлено тем, что mdelay использует примерную оценку прошедшего времени и, в зависимости от состояния аппаратуры, может выдавать ошибку в оценке.

Также был проведен аналогичный эксперимент по проверке возможности внесения изменений в работу ГПСЧ ОС Debian 12 посредством пред-обработчика KProbe с тем же функционалом (задержкой на 50 мс вызова функции get_random_bytes_user()). Представленный на рис. 6 и 7 результат эксперимента также подтвердил ожидаемое поведение: среднее время выполнения чтения из /dev/random увеличилось на 50 мс.

```
=== Benchmark Results ===
Samples: 1000
Chunk size: 8 MB
Average time: 65978.7 µs
Minimum time: 62278 µs
Maximum time: 75158 µs
Average throughput: 127.141 MB/s
```

Рис. 6. Метрики теста после встраивания пред-обработчика KProbe

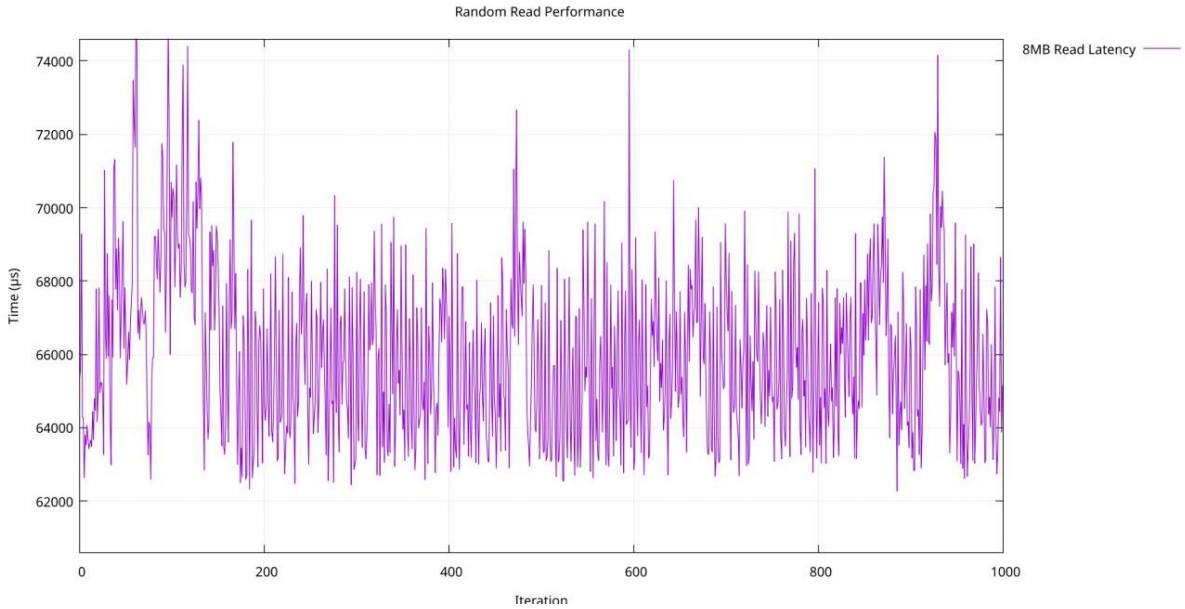

Рис. 7. Визуализация времени выполнения теста после встраивания пред-обработчика KProbe

Следует отметить, что как внедрение, так и выгрузка собственного обработчика из ядра системы требует прав суперпользователя, но при их

наличии легко автоматизируемые. Поэтому было реализовано консольное приложение с функциональностью диспетчера модулей с одновременной поддержкой обеих технологий FTrace и KProbe, который позволяет загружать и выгружать заранее подготовленные модули, тем самым управляя задержкой системного ГПСЧ (рис. 8).

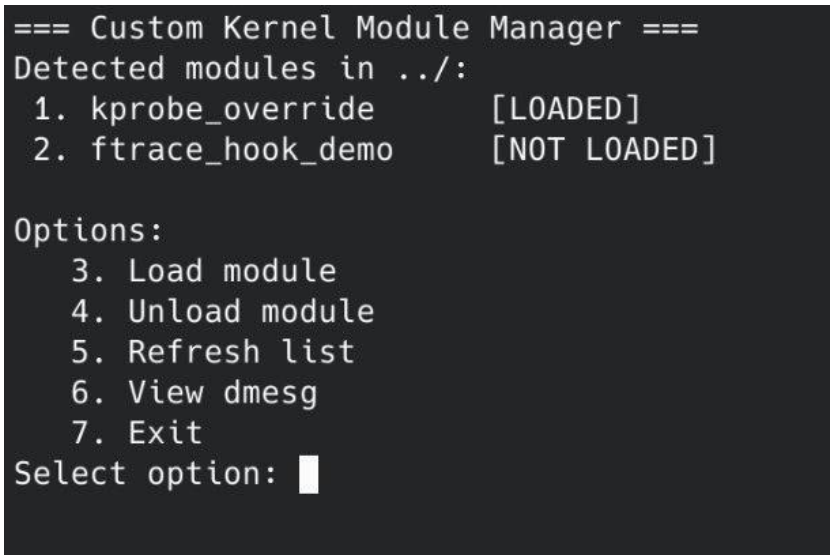

Рис. 8. Интерфейс разработанного приложения

6. Заключение

В статье рассмотрен вопрос проактивной защиты от программ-вымогателей путем контроля системного ГПСЧ ОС Linux. Приведен обзор возможных подходов к реализации такого контроля как на основе полной пересборки ядра, так и путем изменения поведения ядра в уже собранном виде с помощью технологий FTrace и KProbe. Разработаны и экспериментально подтверждены методы внедрения задержек в работу ГПСЧ, что может повлечь замораживание работы программ-вымогателей до начала этапа шифрования пользовательских данных.

Библиографические ссылки

1. *Young A., Yung M.* Cryptovirology: extortion-based security threats and countermeasures // Proceedings 1996 IEEE Symposium on Security and Privacy. 1996. P. 129–140

2. *Fruhlinger J.* Ransomware explained: How it works and how to remove it. // CSO Online. URL: https://www.csoonline.com/article/563507/what-is-ransomware-how-it-works-and-how-to-remove-it.html (date of access: 09.09.2025).

3. 35% Year-over-Year Decrease in Ransomware Payments, Less than Half of Recorded Incidents Resulted in Victim Payments // Chainalysis. URL: https://www.chainalysis.com/blog/crypto-crime-ransomware-victim-extortion-2025/ (date of access: 09.09.2025).

4. Managed detection and response in 2024 // Kaspersky Security Services. – URL: https://securelist.com/kaspersky-managed-detection-and-response-report-2024/115635/ (date of access: 09.09.2025).
5. Ransomware protection: How to keep your data safe in 2025 // Kaspersky Resource Center. URL: https://www.kaspersky.com/resource-center/threats/how-to-prevent-ransomware (date of access: 09.09.2025).
6. *Nugroho Y.* Decrypting Encrypted files from Akira Ransomware (Linux/ESXI variant 2024) using a bunch of GPUs // TinyHack.com – URL: https://tinyhack.com/2025/03/13/decrypting-encrypted-files-from-akira-ransomware-linux-esxi-variant-2024-using-a-bunch-of-gpus (date of access: 09.09.2025).
7. Full source of the Conti Ransomware // Github. – URL: https://github.com/gharty03/Conti-Ransomware (date of access: 09.09.2025).
8. *Müller S.* Documentation and Analysis of the Linux Random Number Generator. Technical report // Bundesamt für Sicherheit in der Informationstechnik. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN_V5_0.pdf (date of access: 09.09.2025).
9. Linux Kernel // Github. URL: https://github.com/torvalds/linux/blob/master/drivers/char/random.c (date of access: 09.09.2025).
10. Linux Security Modules: General Security Hooks for Linux // The kernel development community. URL: https://docs.kernel.org/security/lsm.html (date of access: 09.09.2025).
11. Перехват функций в ядре Linux с помощью ftrace // Хабр : сайт. URL: https://habr.com/ru/articles/413241/ (дата обращения: 09.09.2025).