positive technologies

Как мы реализовали поддержку Astra Linux в PT Sandbox

Алексей Колесников
Младший специалист отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies (PT ESC)

Алексей Вишняков
Руководитель отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies (PT ESC)

ptsecurity.com

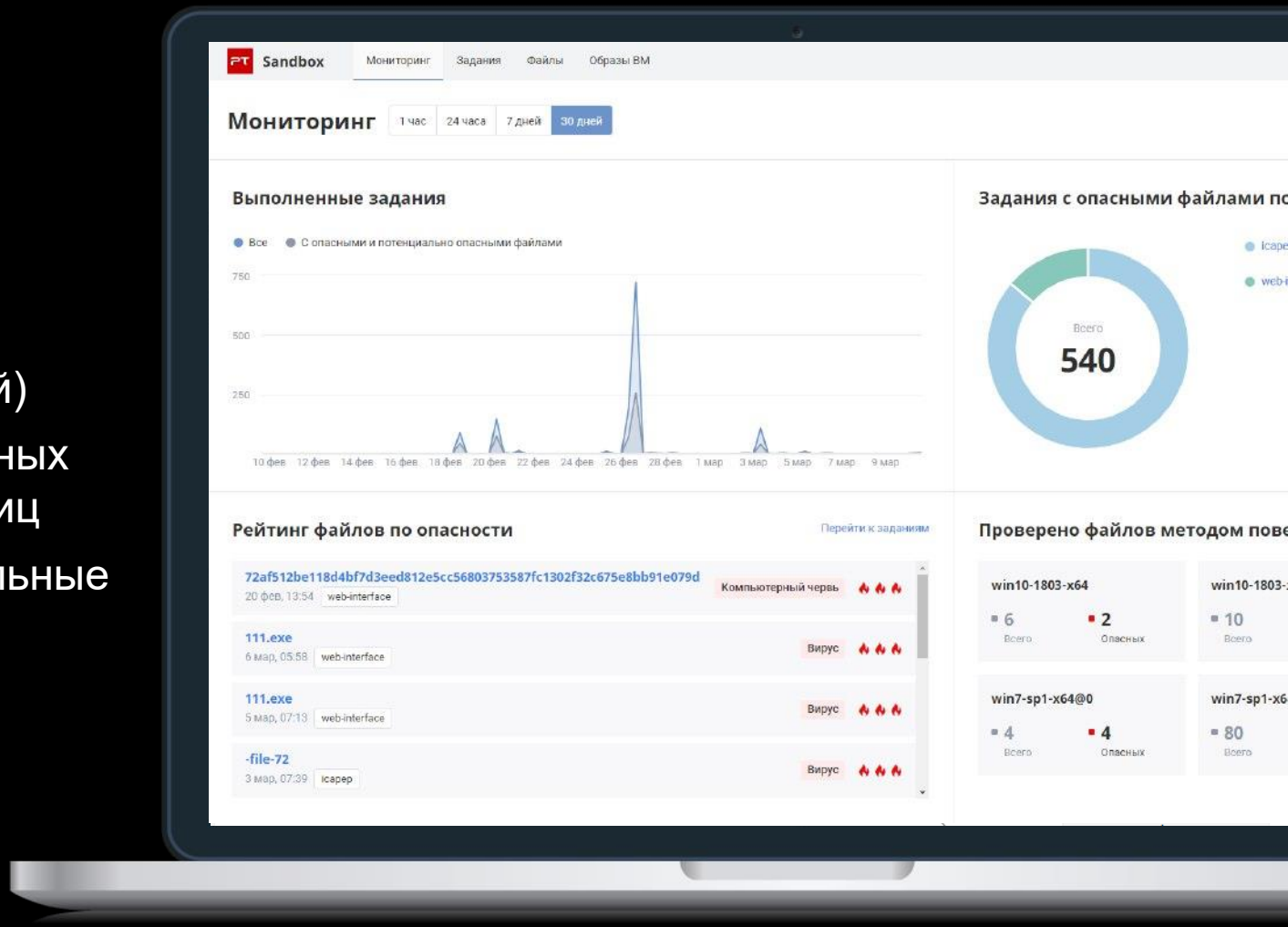
Содержание

- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

PT Sandbox

Песочница для защиты от целевых и массовых атак, в которых используются неизвестное вредоносное ПО и угрозы нулевого дня

- Обеспечивает комплексный анализ файлов и трафика (включая шифрованный)
- Поддерживает гибкую настройку виртуальных сред и защищена от техник обхода песочниц
- Использует уникальные и наиболее актуальные знания для выявления угроз



Содержание

- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

Proof of concept

Три компонента на старте

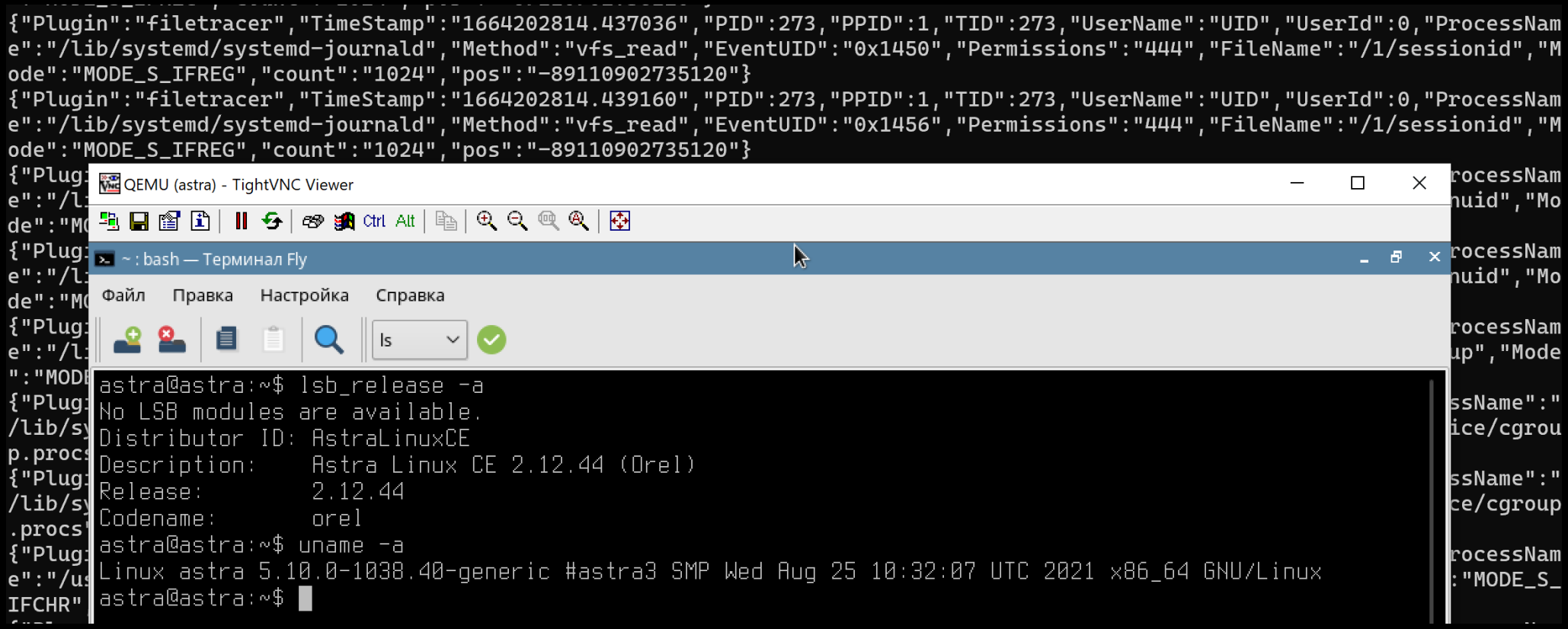
DRAKVUF

Proof of concept

DRAKVUF. Linux-плагины:

- syscalls — трассировка системных вызовов
- filetracer — операции с файловой системой
- …

Proof of concept



Пример работы плагина filetracer

Содержание

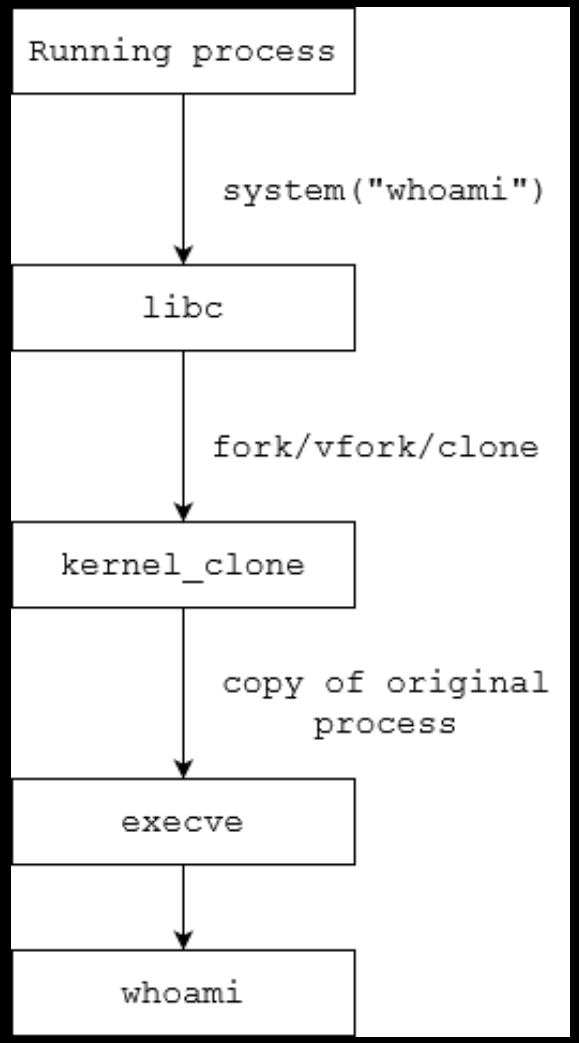
- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

Procmon. Теория

- Как ядро порождает процесс
- Где хранится информация о процессах

Как ядро порождает процесс

- Обращение к libc
- Системный вызов fork (vfork, clone)
- Подмена данных с помощью execve
- Выполнение команды whoami



Как ядро порождает процесс

С версии ядра 5.10 на замену fork (vfork) пришел clone (kernel_clone)

```
2511  #ifdef __ARCH_WANT_SYS_FORK
2512  SYSCALL_DEFINE0(fork)
2513  {
2514  #ifdef CONFIG_MMU
2515          struct kernel_clone_args args = {
2516                  .exit_signal = SIGCHLD,
2517          };
2518
2519          return kernel_clone(&args);
2520  #else
2521          /* can not support in nommu mode */
2522          return -EINVAL;
2523  #endif
2524  }
2525  #endif
2526
2527  #ifdef __ARCH_WANT_SYS_VFORK
2528  SYSCALL_DEFINE0(vfork)
2529  {
2530          struct kernel_clone_args args = {
2531                  .flags          = CLONE_VFORK | CLONE_VM,
2532                  .exit_signal    = SIGCHLD,
2533          };
2534
2535          return kernel_clone(&args);
2536  }
2537  #endif
```

```
2557  SYSCALL_DEFINE5(clone, unsigned long, clone_flags, unsigned long, newsp,
2558                  int __user *, parent_tidptr,
2559                  int __user *, child_tidptr,
2560                  unsigned long, tls)
2561  #endif
2562  {
2563          struct kernel_clone_args args = {
2564                  .flags          = (lower_32_bits(clone_flags) & ~CSIGNAL),
2565                  .pidfd          = parent_tidptr,
2566                  .child_tid      = child_tidptr,
2567                  .parent_tid     = parent_tidptr,
2568                  .exit_signal    = (lower_32_bits(clone_flags) & CSIGNAL),
2569                  .stack          = newsp,
2570                  .tls            = tls,
2571          };
2572
2573          return kernel_clone(&args);
2574  }
2575  #endif
```

Где хранится информация о процессах

Ядерная структура task_struct, описывающая процесс в Linux

/ include / linux / sched.h

```
643
644 struct task_struct {
645 #ifdef CONFIG_THREAD_INFO_IN_TASK
646         /*
647          * For reasons of header soup (see current_thread_info()), this
648          * must be the first element of task_struct.
649          */
650         struct thread_info              thread_info;
651 #endif
652         /* -1 unrunnable, 0 runnable, >0 stopped: */
653         volatile long                   state;
654
655         /*
656          * This begins the randomizable portion of task_struct. Only
657          * scheduling-critical items should be added above here.
658          */
659         randomized_struct_fields_start
660
661         void                            *stack;
662         refcount_t                      usage;
663         /* Per task flags (PF_*), defined further below: */
664         unsigned int                    flags;
665         unsigned int                    ptrace;
```

Где хранится информация о процессах

Ядро хранит короткое имя файла размером 16 байт (с учетом нулевого байта)

```
944         /*
945          * executable name, excluding path.
946          *
947          * - normally initialized setup_new_exec()
948          * - access it with [gs]et_task_comm()
949          * - lock it with task_lock()
950          */
951         char                            comm[TASK_COMM_LEN];
952
953         struct nameidata                *nameidata;
954
```

Procmon. PoC

Первая версия плагина:

- Поле ProcessName не содержит абсолютного пути исполняемого файла
- Нет информации о командной строке
- Отсутствуют переменные окружения


```
{
    Plugin: "procmon_alpha",
    TimeStamp: "1648735724.297456",
    PID: 1451,
    PPID: 1340,
    TID: 1451,
    UserName: "UID",
    UserId: 1000,
    ProcessName: "superlongnamefo",
    Method: "do_execveat_common.isra.0",
    EventUID: "0x6"
}
```

Абсолютный путь исполняемого файла

Чтение абсолютного пути не реализовано в коде

```
char* linux_get_process_name(drakvuf_t drakvuf, addr_t process_base, bool fullpath)
{
    UNUSED(fullpath);
    ACCESS_CONTEXT(ctx,
        .translate_mechanism = VMI_TM_PROCESS_PID,
        .pid = 0,
        .addr = process_base + drakvuf->offsets[TASK_STRUCT_COMM]
    );

    return vmi_read_str(drakvuf->vmi, &ctx);
}
```

Short process name

Абсолютный путь исполняемого файла

Абсолютный путь исполняемого файла

task_struct->mm_struct->exe_file->f_path

```
#define PATH_MAX 4096
char* p = NULL, *pathname;
struct mm_struct* mm = current->mm;
if (mm)
{
    down_read(&mm->mmap_sem);
    if (mm->exe_file)
    {
        pathname = kmalloc(PATH_MAX, GFP_ATOMIC);
        if (pathname)
            p = d_path(&mm->exe_file->f_path, pathname, PATH_MAX);
    }
    up_read(&mm->mmap_sem);
}
```

Абсолютный путь исполняемого файла

- Функция d_path состоит из нескольких больших функций
- Путь читается в обратном порядке
- Есть множество условий, которые нужно учитывать
- Сложно реализуется

```
char *d_path(const struct path *path, char *buf, int buflen)
{
        char *res = buf + buflen;
        struct path root;
        int error;

        /*
         * We have various synthetic filesystems that never get mounted.  On
         * these filesystems dentries are never used for lookup purposes, and
         * thus don't need to be hashed.  They also don't need a name until a
         * user wants to identify the object in /proc/pid/fd/.  The little hack
         * below allows us to generate a name for these objects on demand:
         *
         * Some pseudo inodes are mountable.  When they are mounted
         * path->dentry == path->mnt->mnt_root.  In that case don't call d_dname
         * and instead have d_path return the mounted path.
         */
        if (path->dentry->d_op && path->dentry->d_op->d_dname &&
            (!IS_ROOT(path->dentry) || path->dentry != path->mnt->mnt_root))
                return path->dentry->d_op->d_dname(path->dentry, buf, buflen);

        rcu_read_lock();
        get_fs_root_rcu(current->fs, &root);
        error = path_with_deleted(path, &root, &res, &buflen);
        rcu_read_unlock();

        if (error < 0)
                res = ERR_PTR(error);
        return res;
}
EXPORT_SYMBOL(d_path);
```

Complex functions

Абсолютный путь исполняемого файла

prepend_path — основная функция, создающая строку с абсолютным путем

Нельзя обратиться к полю структуры напрямую, нужно прочитать и посчитать смещение каждого поля

```
static int prepend_path(const struct path *path,
                        const struct path *root,
                        char **buffer, int *buflen)
{
        struct dentry *dentry;
        struct vfsmount *vfsmnt;
        struct mount *mnt;
        int error = 0;
        unsigned seq, m_seq = 0;
        char *bptr;
        int blen;

        rcu_read_lock();
restart_mnt:
        read_seqbegin_or_lock(&mount_lock, &m_seq);
        seq = 0;
        rcu_read_lock();
restart:
        bptr = *buffer;
        blen = *buflen;
        error = 0;
        dentry = path->dentry;
        vfsmnt = path->mnt;
        mnt = real_mount(vfsmnt);
        read_seqbegin_or_lock(&rename_lock, &seq);
        while (dentry != root->dentry || vfsmnt != root->mnt) {
                struct dentry * parent;

                if (dentry == vfsmnt->mnt_root || IS_ROOT(dentry)) {
                        struct mount *parent = READ_ONCE(mnt->mnt_parent);
                        struct mnt_namespace *mnt_ns;
```

Абсолютный путь исполняемого файла

Когда хотел написать простой плагин для DRAKVUF

Абсолютный путь исполняемого файла

Часть функции prepend_path в DRAKVUF

```
static void prepend_path(drakvuf_t drakvuf, addr_t path, addr_t root, GString* b)
{
    ACCESS_CONTEXT(ctx,
        .translate_mechanism = VMI_TM_PROCESS_DTB,
        .dtb = drakvuf->kpgd,
        .addr = path + drakvuf->offsets[PATH_DENTRY]);
    addr_t dentry;
    if (VMI_FAILURE == vmi_read_addr(drakvuf->vmi, &ctx, &dentry))
    {
        PRINT_DEBUG("Can't get path->dentry from struct path\n");
        return;
    }

    ctx.addr = path + drakvuf->offsets[PATH_MNT];
    addr_t vfsmnt;
    if (VMI_FAILURE == vmi_read_addr(drakvuf->vmi, &ctx, &vfsmnt))
    {
        PRINT_DEBUG("Can't get path->mnt from struct path\n");
        return;
    }

    addr_t root_dentry;
    ctx.addr = root + drakvuf->offsets[PATH_DENTRY];
    if (VMI_FAILURE == vmi_read_addr(drakvuf->vmi, &ctx, &root_dentry))
    {
        PRINT_DEBUG("Can't get root->dentry from root\n");
        return;
    }
```

Just prepare to main logic

Абсолютный путь исполняемого файла

Полный путь до исполняемого файла в поле ProcessName


```
{
    Plugin: "procmon",
    TimeStamp: "1664226785.979647",
    PID: 1620,
    PPID: 1,
    TID: 1620,
    UserName: "UID",
    UserId: 0,
    ProcessName: "/usr/lib/NetworkManager/nm-dispatcher",
    Method: "do_execveat_common"
}
```

В поисках родителя

Особенности procmon:

- PID — родитель
- NewPid — порождаемый процесс

Особенности перехвата execve:

- PID уже определен
- Находимся в дочернем процессе

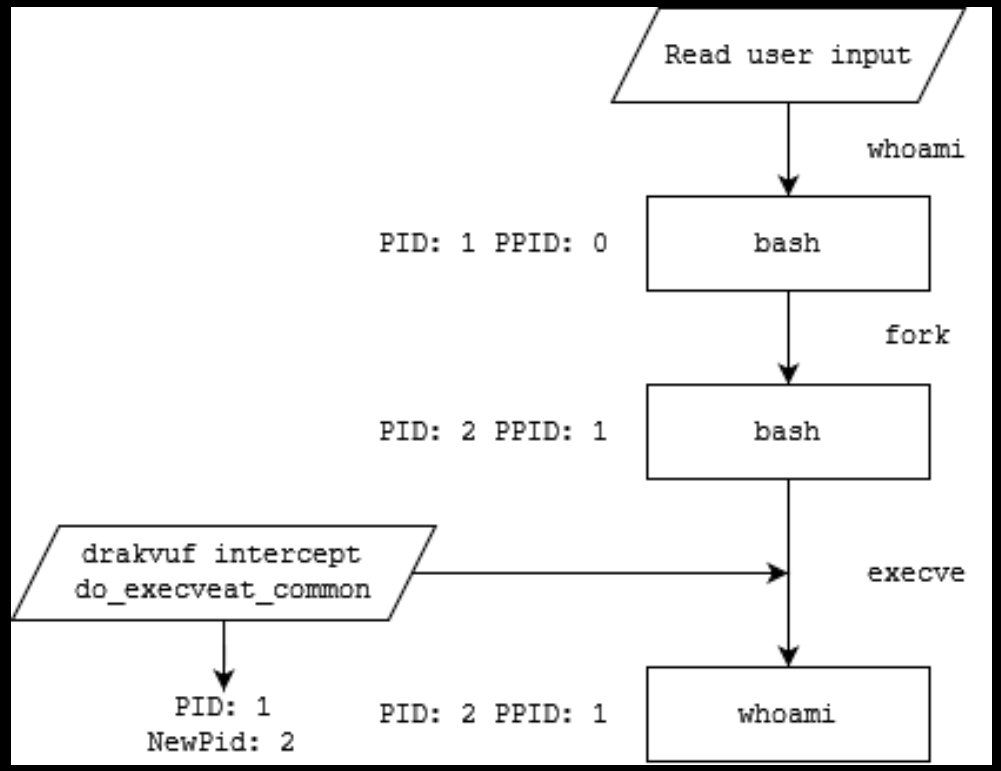


Схема создания процесса

В поисках родителя

Как получить информацию о процессе по PID

```
bool drakvuf_get_process_by_pid(drakvuf_t drakvuf, vmi_pid_t pid, addr_t* process, addr_t* dtb)
{
    pass_context_t pctx =
    {
        .pid = &pid,
        .process = process,
        .dtb = dtb
    };
    // Get process by pid
    drakvuf_enumerate_processes(drakvuf, process_visitor, (void*)(&pctx));

    if (!*process || !*dtb)
        return false;
    return true;
}
```

Main function

В поисках родителя

Windows

```
drakvuf->osi.is_wow64 = win_is_wow64;
drakvuf->osi.get_function_argument = win_get_function_argument;
drakvuf->osi.get_function_return_address = win_get_function_return_
drakvuf->osi.enumerate_processes = win_enumerate_processes;
drakvuf->osi.enumerate_processes_with_module = win_enumerate_proces
drakvuf->osi.enumerate_drivers = win_enumerate_drivers;
drakvuf->osi.enumerate_process_modules = win_enumerate_process_modu
drakvuf->osi.is_crashreporter = win_is_crashreporter;
drakvuf->osi.find_mmvad = win_find_mmvad;
drakvuf->osi.traverse_mmvad = win_traverse_mmvad;
drakvuf->osi.is_mmvad_commited = win_is_mmvad_commited;
```

Linux

```
drakvuf->osi.get_current_thread = linux_get_current_thread;
drakvuf->osi.get_current_process = linux_get_current_process;
drakvuf->osi.get_process_name = linux_get_process_name;
drakvuf->osi.get_current_process_name = linux_get_current_process_name;
drakvuf->osi.get_process_userid = linux_get_process_userid;
drakvuf->osi.get_current_process_userid = linux_get_current_process_userid;
drakvuf->osi.get_current_thread_id = linux_get_current_thread_id;
drakvuf->osi.get_process_pid = linux_get_process_pid;
drakvuf->osi.get_process_tid = linux_get_process_tid;
drakvuf->osi.get_process_ppid = linux_get_process_ppid;
drakvuf->osi.get_process_data = linux_get_process_data;
drakvuf->osi.exportsym_to_va = linux_eprocess_sym2va;
drakvuf->osi.export_lib_address = get_lib_address;
drakvuf->osi.get_function_argument = linux_get_function_argument;
drakvuf->osi.get_function_return_address = linux_get_function_return_address;
drakvuf->osi.check_return_context = linux_check_return_context;
```

Интерфейс функций ОС в DRAKVUF

В поисках родителя

В поисках родителя

init_task — нулевой процесс

task_struct — linked list (можно получить доступ ко всем процессам в системе)

```
struct sched_info          sched_info;
                           Pointer to another task
struct list_head           tasks;
```

```
60  /*
61   * Set up the first task table, touch at your own risk!. Base=0,
62   * limit=0x1fffff (=2MB)
63   */
64  struct task_struct init_task
65  #ifdef CONFIG_ARCH_TASK_STRUCT_ON_STACK
66          __init_task_data
67  #endif
68          __aligned(L1_CACHE_BYTES)
69  = {
70  #ifdef CONFIG_THREAD_INFO_IN_TASK
71          .thread_info    = INIT_THREAD_INFO(init_task),
72          .stack_refcount = REFCOUNT_INIT(1),
73  #endif
74          .state          = 0,
75          .stack          = init_stack,
76          .usage          = REFCOUNT_INIT(2),
77          .flags          = PF_KTHREAD,
78          .prio           = MAX_PRIO - 20,
79          .static_prio    = MAX_PRIO - 20,
80          .normal_prio    = MAX_PRIO - 20,
81          .policy         = SCHED_NORMAL,
82          .cpus_ptr       = &init_task.cpus_mask,
83          .cpus_mask      = CPU_MASK_ALL,
84          .nr_cpus_allowed= NR_CPUS,
85          .mm             = NULL,
86          .active_mm      = &init_mm,
```

В поисках родителя

Нулевой процесс имеет имя **swapper**

```
#define INIT_TASK_COMM "swapper"

.ptraced        = LIST_HEAD_INIT(init_task.ptraced),
.ptrace_entry   = LIST_HEAD_INIT(init_task.ptrace_entry),
.real_parent    = &init_task,
.parent         = &init_task,
.children       = LIST_HEAD_INIT(init_task.children),
.sibling        = LIST_HEAD_INIT(init_task.sibling),
.group_leader   = &init_task,
RCU_POINTER_INITIALIZER(real_cred, &init_cred),
RCU_POINTER_INITIALIZER(cred, &init_cred),
.comm           = INIT_TASK_COMM,
.thread         = INIT_THREAD,
.fs             = &init_fs,
.files          = &init_files,
```

В поисках родителя

```
bool linux_enumerate_processes(drakvuf_t drakvuf, void (*visitor_func)(drakvuf_t drakvuf, addr_t eprocess, void* visitor_ctx), void* visitor_ctx)
{
    addr_t list_head;
    if (!linux_find_process_list(drakvuf, &list_head))
        return false;

    addr_t current_list_entry = list_head;
    addr_t next_list_entry;

    do
    {
        addr_t process_base = linux_process_list_entry_to_process(drakvuf, current_list_entry);

        visitor_func(drakvuf, process_base, visitor_ctx);

        if (!linux_find_next_process_list_entry(drakvuf, current_list_entry, &next_list_entry))
        {
            PRINT_DEBUG("Failed to find next task!\n");
            return false;
        }
        current_list_entry = next_list_entry;
    } while (list_head != next_list_entry);

    return true;
}
```

gather init_task addr

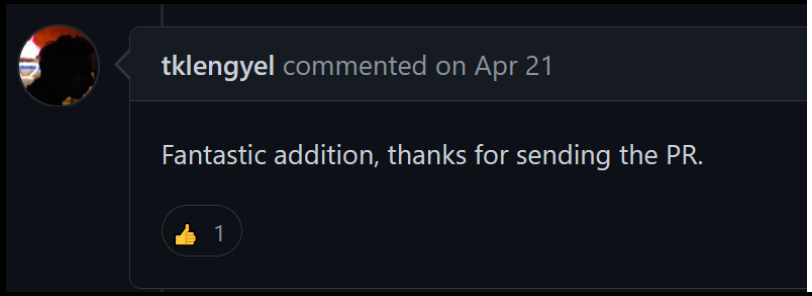
compare process data with context data (pid, dtb, etc)

gather next process from task_struct

Имплементация функции linux_enumerate_process

Procmon. Итоги

- Появилась Linux-версия плагина
- DRAKVUF значительно доработан под Linux
- Теперь писать плагины стало удобнее и быстрее




```
{
    Plugin: "procmon",
    TimeStamp: "1664244505.483220",
    PID: 1212,
    PPID: 599,
    TID: 1212,
    UserName: "UID",
    UserId: 1000,
    ProcessName: "/bin/bash",
    Method: "do_execveat_common",
    EventUID: "0xc",
    ThreadName: "bash",
    NewPid: 3296,
    NewTid: 3296,
    CommandLine: "ls --color=auto -la",
    ImagePathName: "/bin/ls"
}
```

MITRE ATT&CK. Первичная экспертиза

Ключевые тактики:

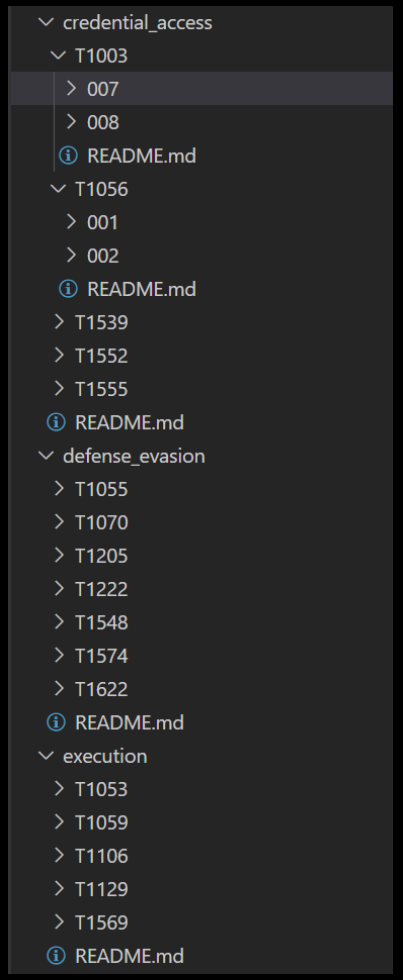
- Defense Evasion
- Credential Access
- Execution

Анализ плагинов:

- procmon
- filetracer

MITRE ATT&CK. Первичная экспертиза

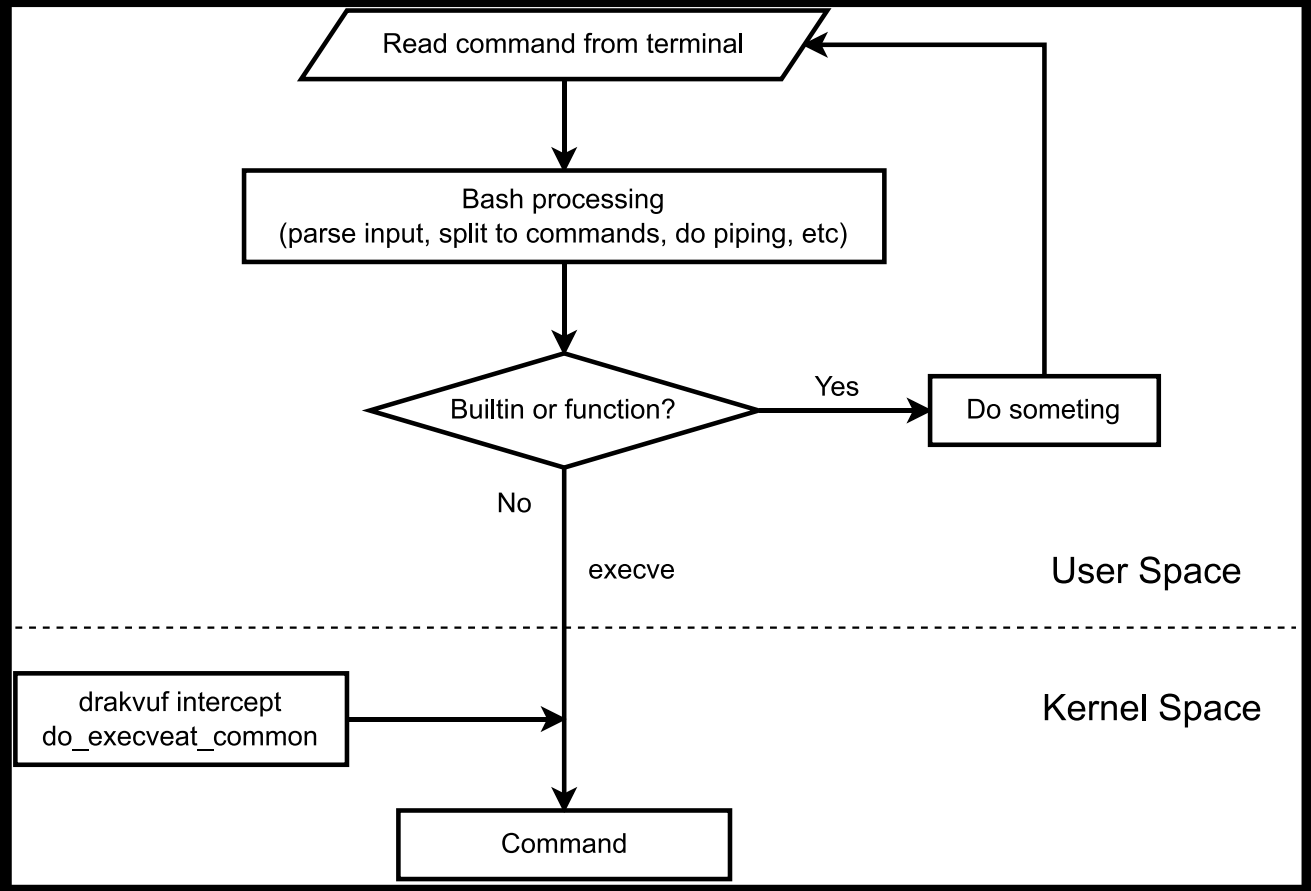
- procmon — вывод LD_PRELOAD
- Перехват Bash-команд в ядре
- socketmon — перехват сетевой активности
- ptracemon — перехват вызова ptrace
- ebpfmon — перехват вызова bpf
- Переработка интерфейса для написания плагинов



Содержание

- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

Перехват команд Bash в ядре



Упрощенная схема обработки команд

Перехват команд Bash в ядре

Зачем изобретать велосипед?

Перехват команд Bash в ядре

```c
/*
 * Function wrapper - execve()
 */
__attribute__((visibility("default"))) int execve (const char *filename, char *const argv[], char *const envp[])
{
    static int (*func)(const char *, char * const *, char * const *);

    FN(func, int, "execve", (const char *, char * const *, char * const *));

    snoopy_entrypoint_execve_wrapper_init(filename, argv, envp);
    snoopy_action_log_syscall_exec();
    snoopy_entrypoint_execve_wrapper_exit();

    return (*func) (filename, argv, envp);
}
```

execve from libc

Log action

```c
#if defined(RTLD_NEXT)
#  define REAL_LIBC RTLD_NEXT
#else
#  define REAL_LIBC ((void *) -1L)
#endif

#define FN(ptr, type, name, args)   ptr = (type (*)args)dlsym (REAL_LIBC, name)
```

Search symbol

github.com/a2o/snoopy

Перехват команд Bash в ядре

Что предлагает интернет?

```
alias iso8601="date +%Y-%m-%dT%H:%M:%S"

justlog(){
  name=$(basename "$1")
  log=~/logs/${name}-$(iso8601)
  "$@" > "$log" 2>&1
}

timelog(){
  name=$(basename "$1")
  log=~/logs/${name}-$(iso8601)
  # https://github.com/wwalker/ilts
  # You could replace ilts with ts
  # /usr/bin/ts %FT%H:%M:%.S
  "$@" |& ilts -S -E > "$log" 2>&1
}
```

`set -x` is fine, but if you do something like:

```
set -x;
command;
set +x;
```

```
trap 'echo "# $BASH_COMMAND"' DEBUG
```

Препарируем Bash

Препарируем Bash

```
#define __NR_drakvuf 1337
void call_drakvuf(char* line)
{
    int ret;
    asm(
        "syscall"
        : "=a" (ret)
        : "0" (__NR_drakvuf), "D" (line)
        : "rcx", "r11", "memory"
    );
}
```

Dirty hack — inline assembly в исходном коде

Препарируем Bash

```
void
bash_syslog_history (line)
      const char *line;
{
  char trunc[SYSLOG_MAXLEN];
  static int first = 1;

  if (first)
    {
      openlog (shell_name, OPENLOG_OPTS, SYSLOG_FACILITY);
      first = 0;
    }

  if (strlen(line) < SYSLOG_MAXLEN)
    syslog (SYSLOG_FACILITY|SYSLOG_LEVEL, "HISTORY: PID=%d UID=%d %s", getpid(), current_user.uid, line);
  else
    {
      strncpy (trunc, line, SYSLOG_MAXLEN);
      trunc[SYSLOG_MAXLEN - 1] = '\0';
      syslog (SYSLOG_FACILITY|SYSLOG_LEVEL, "HISTORY (TRUNCATED): PID=%d UID=%d %s", getpid(), current_user.uid, trunc);
    }
}
```

Data will be sent to the syslog daemon

Функция, отвечающая за отправку данных в демона

Препарируем Bash

- История команд включена принудительно
- Bash модифицируется по необходимости
- Логирование не зависит от конфигурации оболочки
- Программы из user space не могут повлиять на данные

```
/* Define if you want each line saved to the history list in bashhist.c:
   bash_add_history() to be sent to syslog(). */
/* #define SYSLOG_HISTORY */
#if defined (SYSLOG_HISTORY)
#  define SYSLOG_FACILITY LOG_USER
#  define SYSLOG_LEVEL LOG_INFO
#  define OPENLOG_OPTS LOG_PID
#endif
```

Define a variable to force syslog

Анализ syslog

- syslog - функция из libc
- syscall syslog не используется
- Данные отправляются в демона с помощью сокетов

```
/* Get connected, output the message to the local logger.  */
if (!connected)
  openlog_internal (NULL, LogStat | LOG_NDELAY, LogFacility);

/* If we have a SOCK_STREAM connection, also send ASCII NUL as a record
   terminator.  */
if (LogType == SOCK_STREAM)
  ++bufsize;

if (!connected || __send (LogFile, buf, bufsize, MSG_NOSIGNAL) < 0)
  {
    if (connected)
      {
        /* Try to reopen the syslog connection.  Maybe it went down.  */
        closelog_internal ();
        openlog_internal (NULL, LogStat | LOG_NDELAY, LogFacility);
      }

    if (!connected || __send (LogFile, buf, bufsize, MSG_NOSIGNAL) < 0)
      {
```

send data through unix sockets to daemon

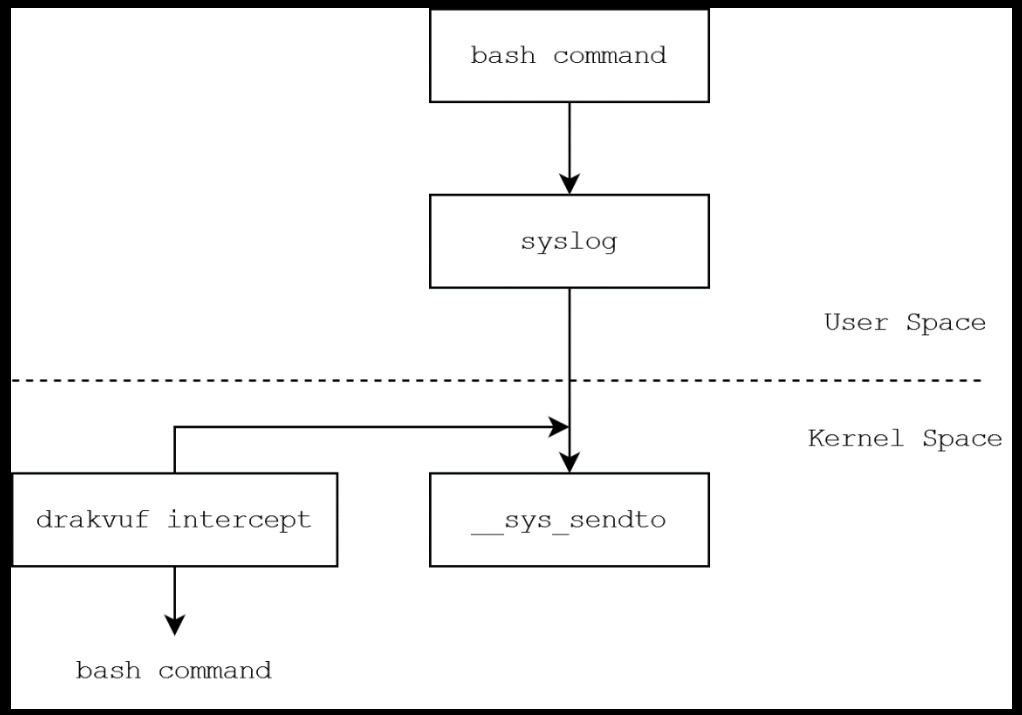
syslog → unixsocketmon

Процесс получения команды Bash из user space в kernel space

```
SYSCALL_DEFINE6(sendto, int, fd, void __user *, buff, size_t, len,
                unsigned int, flags, struct sockaddr __user *, addr,
                int, addr_len)
{
        return __sys_sendto(fd, buff, len, flags, addr, addr_len);
}

/*
 *      Send a datagram down a socket.
 */

SYSCALL_DEFINE4(send, int, fd, void __user *, buff, size_t, len,
                unsigned int, flags)
{
        return __sys_sendto(fd, buff, len, flags, NULL, 0);
}
```

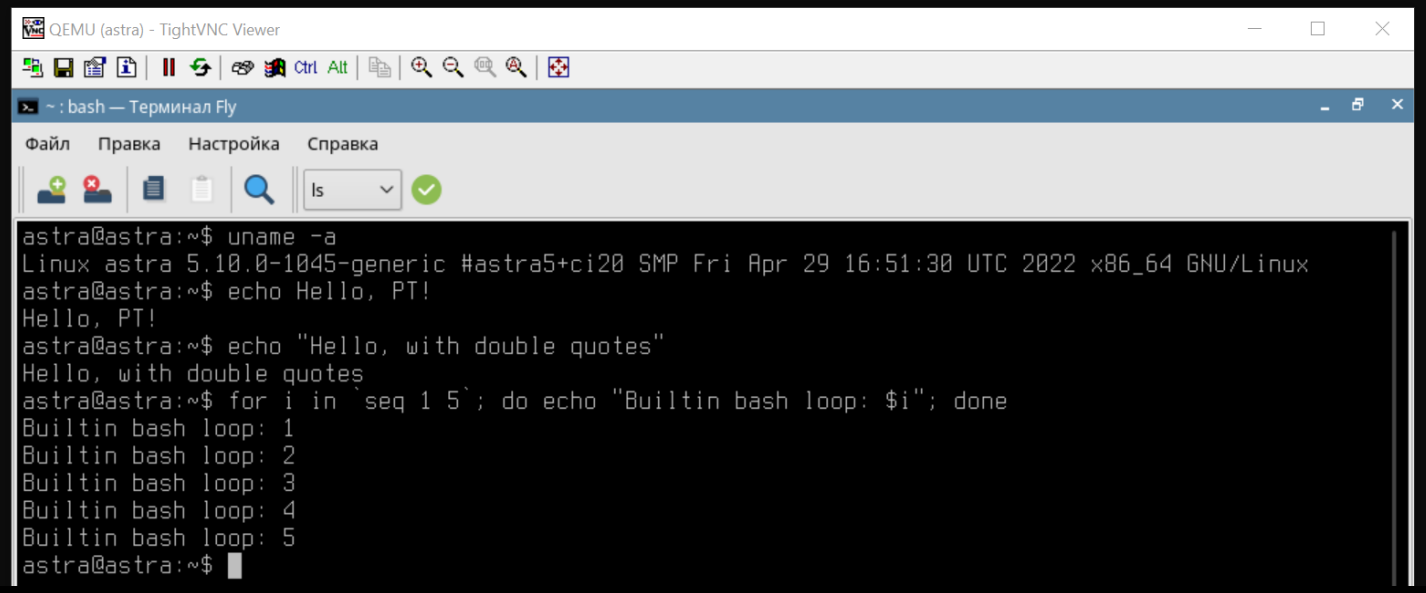
Different syscalls

Same function



Unixsocketmon


```
{"Plugin":"unixsocketmon","TimeStamp":"1663781111.154071","PID":1718,"PPID":1715,"TID":1718,"UserName":"UID","UserId":1000,"ProcessName":"/bin/b
ash","Method":"sock_sendmsg","EventUID":"0xfb","Type":"AF_LOCAL","Size":85,"Value":"<14>Sep 21 20:25:11 /bin/bash[1718]: HISTORY: PID=1718 UID=1
000 CommandLine: uname -a"}
{"Plugin":"unixsocketmon","TimeStamp":"1663781113.808604","PID":1718,"PPID":1715,"TID":1718,"UserName":"UID","UserId":1000,"ProcessName":"/bin/b
ash","Method":"sock_sendmsg","EventUID":"0x12a","Type":"AF_LOCAL","Size":92,"Value":"<14>Sep 21 20:25:13 /bin/bash[1718]: HISTORY: PID=1718 UID=
1000 CommandLine: echo Hello, PT!"}
{"Plugin":"unixsocketmon","TimeStamp":"1663781116.022816","PID":1718,"PPID":1715,"TID":1718,"UserName":"UID","UserId":1000,"ProcessName":"/bin/b
ash","Method":"sock_sendmsg","EventUID":"0x154","Type":"AF_LOCAL","Size":109,"Value":"<14>Sep 21 20:25:15 /bin/bash[1718]: HISTORY: PID=1718 UID
=1000 CommandLine: echo \"Hello, with double quotes\""}
{"Plugin":"unixsocketmon","TimeStamp":"1663781120.166077","PID":1718,"PPID":1715,"TID":1718,"UserName":"UID","UserId":1000,"ProcessName":"/bin/b
ash","Method":"sock_sendmsg","EventUID":"0x17e","Type":"AF_LOCAL","Size":134,"Value":"<14>Sep 21 20:25:20 /bin/bash[1718]: HISTORY: PID=1718 UID
=1000 CommandLine: for i in `seq 1 5`; do echo \"Builtin bash loop: $i\"; done"}
```

Пример работы плагина unixsocketmon

Содержание

- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

Ptracemon

Перехват одного системного вызова ptrace

```
SYSCALL_DEFINE4(ptrace, long, request, long, pid, unsigned long, addr,
                unsigned long, data)
{
        struct task_struct *child;
        long ret;

        if (request == PTRACE_TRACEME) {
                ret = ptrace_traceme();
                goto out;
        }

        child = find_get_task_by_vpid(pid);
        if (!child) {
                ret = -ESRCH;
                goto out;
        }
```

```
#define PTRACE_TRACEME          0
#define PTRACE_PEEKTEXT         1
#define PTRACE_PEEKDATA         2
#define PTRACE_PEEKUSR          3
#define PTRACE_POKETEXT         4
#define PTRACE_POKEDATA         5
#define PTRACE_POKEUSR          6
#define PTRACE_CONT             7
#define PTRACE_KILL             8
#define PTRACE_SINGLESTEP       9

#define PTRACE_ATTACH          16
#define PTRACE_DETACH          17

#define PTRACE_SYSCALL         24
```

Ptracemon


```
{"Plugin":"ptracemon","TimeStamp":"1664270646.444392","PID":6400,"PPID":6399,"TID":6400,"UserName":"UID","UserId":0,"ProcessNa
me":"/home/astra/linux-inject/inject","Method":"ptrace","EventUID":"0x1","Type":"PTRACE_ATTACH","TargetPID":6373,"TargetProces
sName":"/home/astra/linux-inject/sample-target"}
{"Plugin":"ptracemon","TimeStamp":"1664270646.462064","PID":6400,"PPID":6399,"TID":6400,"UserName":"UID","UserId":0,"ProcessNa
me":"/home/astra/linux-inject/inject","Method":"ptrace","EventUID":"0x2","Type":"PTRACE_GETREGS","TargetPID":6373,"TargetProce
ssName":"/home/astra/linux-inject/sample-target"}
```


QEMU (astra) - TightVNC Viewer

linux-inject : bash — Терминал Fly

Файл Правка Настройка Справка

```
astra@astra:~/linux-inject$ sudo ./inject -n sample-target ./sample-library.so
targeting process "sample-target" with pid 6373
ptrace(PTRACE_GETSIGINFO) failed
astra@astra:~/linux-inject$
```

Пример работы плагина ptracemon

Содержание

- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

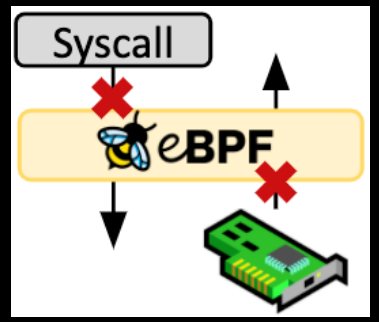
Анализируем eBPF в Astra Linux

Extended Berkeley Packet Filter

Анализируем eBPF в Astra Linux

Extended Berkeley Packet Filter

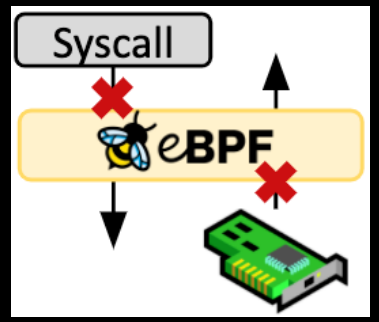
Security



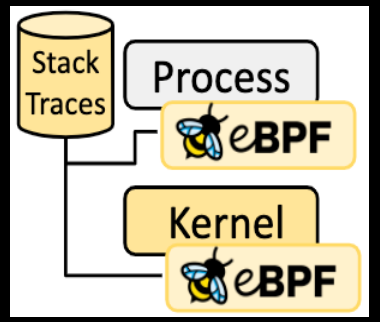
Анализируем eBPF в Astra Linux

Extended Berkeley Packet Filter

Security



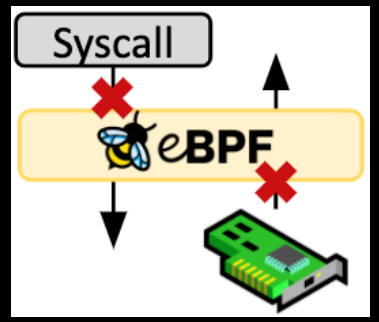
Tracing & profiling



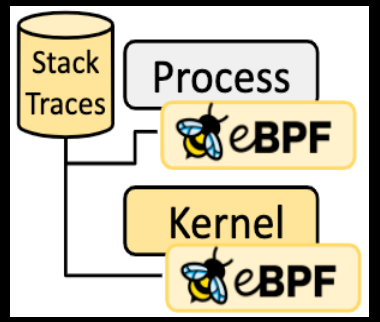
Анализируем eBPF в Astra Linux

Extended Berkeley Packet Filter

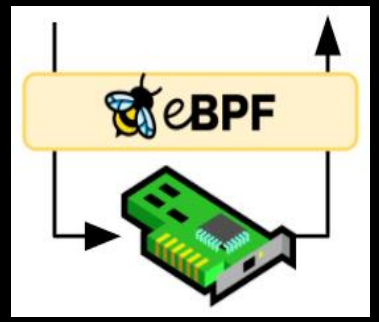
Security



Tracing & profiling



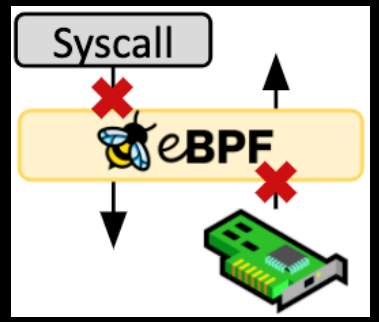
Networking



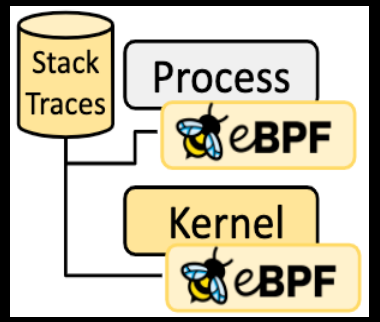
Анализируем eBPF в Astra Linux

Extended Berkeley Packet Filter

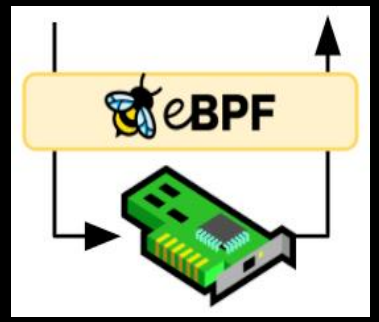
Security



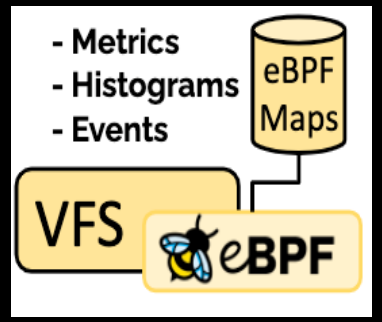
Tracing & profiling



Networking



Observability & monitoring



Анализируем eBPF в Astra Linux

Extended Berkeley Packet Filter

CO-RE
Compile Once — Run Everywhere

Анализируем eBPF в Astra Linux

Сравнение на примере tc (traffic control) из пакета iproute2

```
astra@astra:~$ ldd /sbin/tc
        linux-vdso.so.1 (0x00007fffa7fa4000)
        libelf.so.1 => /usr/lib/x86_64-linux-gnu/libelf.so.1 (0x00007d2df9111000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007d2df8e0d000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007d2df8c09000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007d2df886a000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007d2df8650000)
        /lib64/ld-linux-x86-64.so.2 (0x00007d2df9590000)
astra@astra:~$
```

```
ubuntu@laptop:~$ ldd /sbin/tc
        linux-vdso.so.1 (0x00007ffd0a3ea000)
        libbpf.so.0 => /lib/x86_64-linux-gnu/libbpf.so.0 (0x00007fc5d90af000)
        libelf.so.1 => /lib/x86_64-linux-gnu/libelf.so.1 (0x00007fc5d9091000)
        libmnl.so.0 => /lib/x86_64-linux-gnu/libmnl.so.0 (0x00007fc5d9089000)
        libbsd.so.0 => /lib/x86_64-linux-gnu/libbsd.so.0 (0x00007fc5d9071000)
        libcap.so.2 => /lib/x86_64-linux-gnu/libcap.so.2 (0x00007fc5d9066000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fc5d8f7f000)
        libxtables.so.12 => /lib/x86_64-linux-gnu/libxtables.so.12 (0x00007fc5d8f6a000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc5d8d42000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fc5d8d26000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc5d91b4000)
        libmd.so.0 => /lib/x86_64-linux-gnu/libmd.so.0 (0x00007fc5d8d19000)
ubuntu@laptop:~$
```

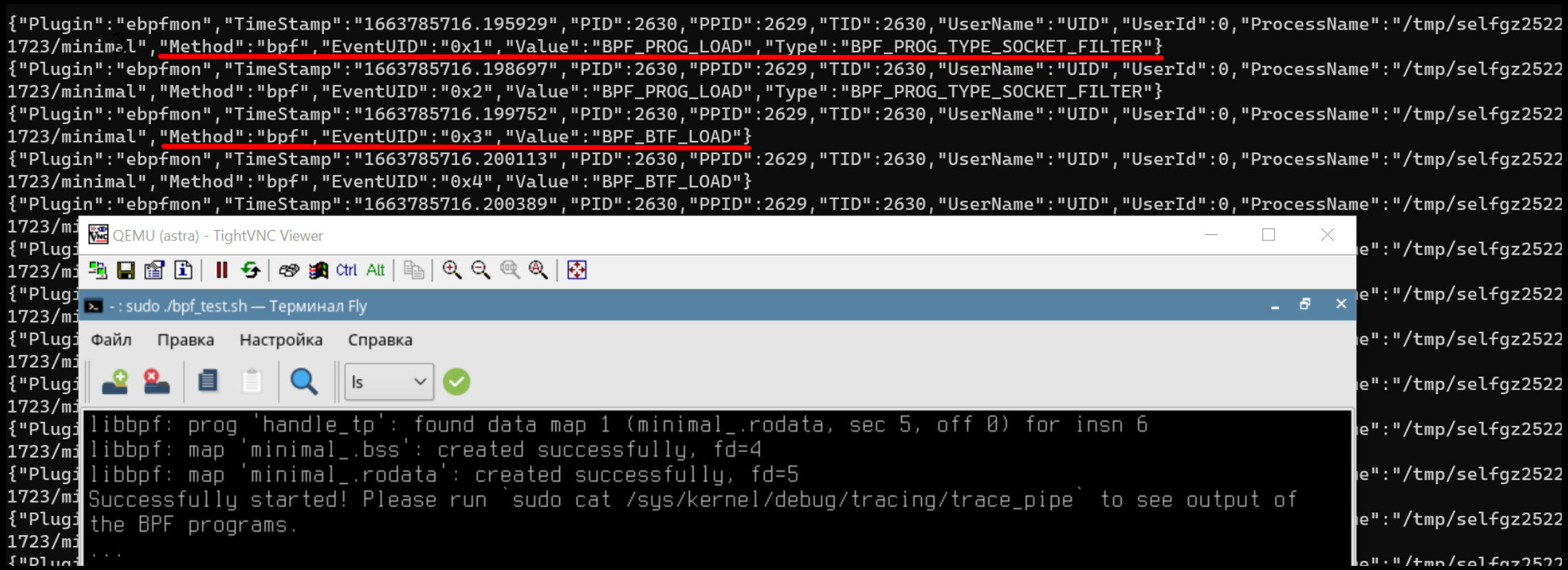
Ebpfmon

Макрос раскроется и создаст символ __do_sys_bpf

```
SYSCALL_DEFINE3(bpf, int, cmd, union bpf_attr __user *, uattr, unsigned int, size)
{
        union bpf_attr attr;
        int err;

        if (sysctl_unprivileged_bpf_disabled && !bpf_capable())
                return -EPERM;

        err = bpf_check_uarg_tail_zero(uattr, sizeof(attr), size);
        if (err)
                return err;
        size = min_t(u32, size, sizeof(attr));

        /* copy attributes from user space, may be less than sizeof(bpf_attr) */
        memset(&attr, 0, sizeof(attr));
        if (copy_from_user(&attr, uattr, size) != 0)
                return -EFAULT;

        err = security_bpf(cmd, &attr, size);
        if (err < 0)
                return err;

        switch (cmd) {
        case BPF_MAP_CREATE:
                err = map_create(&attr);
                break;
        case BPF_MAP_LOOKUP_ELEM:
                err = map_lookup_elem(&attr);
                break;
```

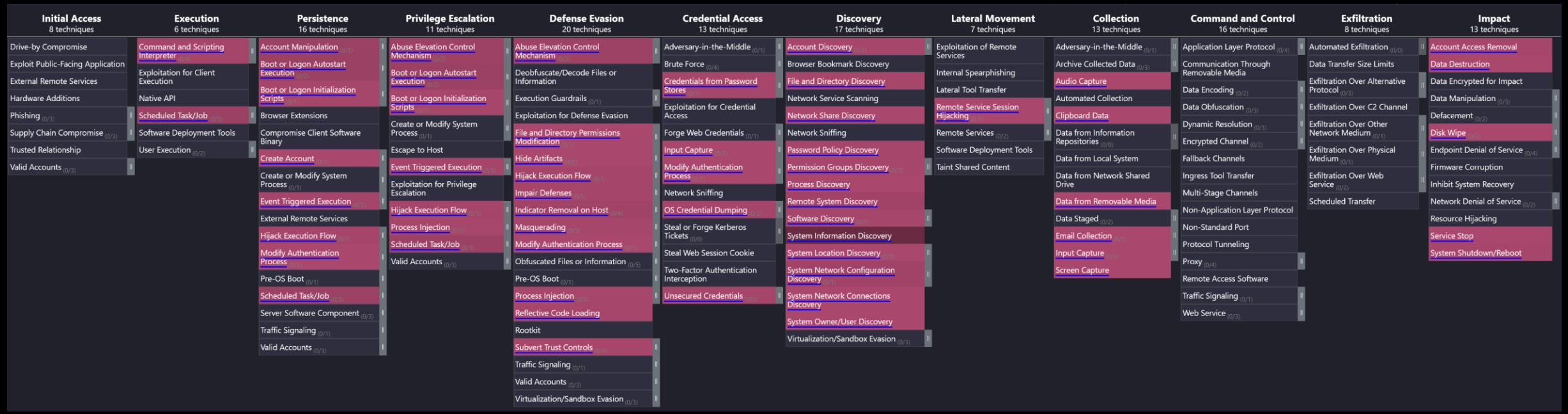
```
#define __SYSCALL_DEFINEx(x, name, ...)                                        \
        __diag_push();                                                          \
        __diag_ignore(GCC, 8, "-Wattribute-alias",                              \
                      "Type aliasing is used to sanitize syscall arguments");\
        asmlinkage long sys##name(__MAP(x,__SC_DECL,__VA_ARGS__))               \
                __attribute__((alias(__stringify(__se_sys##name))));            \
        ALLOW_ERROR_INJECTION(sys##name, ERRNO);                                \
        static inline long __do_sys##name(__MAP(x,__SC_DECL,__VA_ARGS__));\
        asmlinkage long __se_sys##name(__MAP(x,__SC_LONG,__VA_ARGS__));\
        asmlinkage long __se_sys##name(__MAP(x,__SC_LONG,__VA_ARGS__))\
        {                                                                       \
                long ret = __do_sys##name(__MAP(x,__SC_CAST,__VA_ARGS__));\
                __MAP(x,__SC_TEST,__VA_ARGS__);                                 \
                __PROTECT(x, ret,__MAP(x,__SC_ARGS,__VA_ARGS__));               \
                return ret;                                                     \
        }                                                                       \
        __diag_pop();                                                           \
        static inline long __do_sys##name(__MAP(x,__SC_DECL,__VA_ARGS__))
#endif /* __SYSCALL_DEFINEx */
```

Ebpfmon



Пример работы плагина ebpfmon

MITRE ATT&CK. Экспертиза



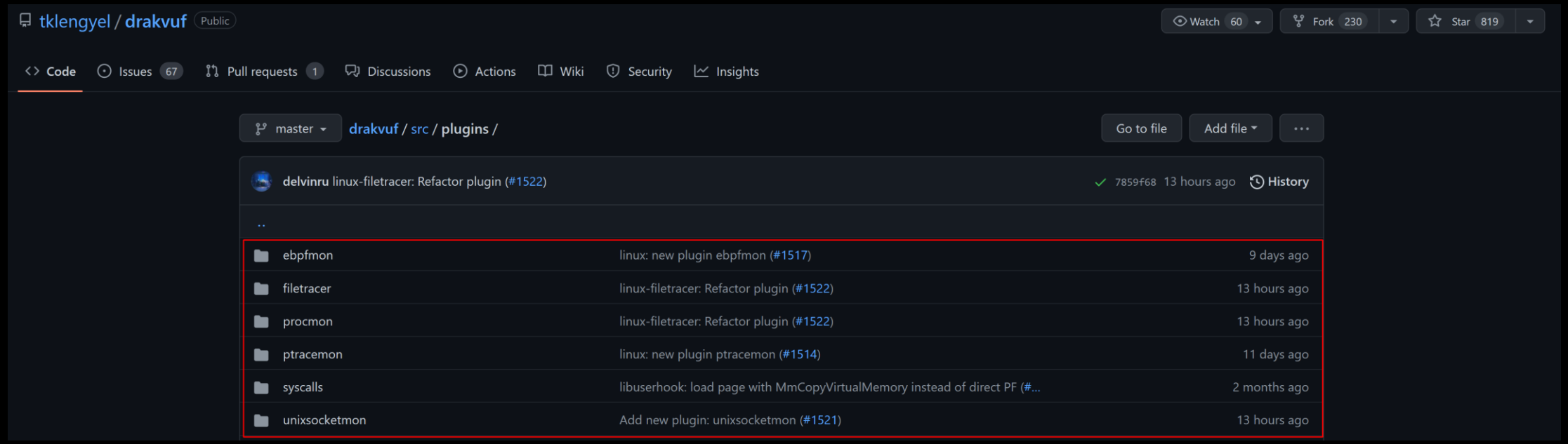
Покрываемые техники в PT Sandbox для Astra Linux

Содержание

- Коротко о PT Sandbox
- Proof of concept
- Procmon
- Unixsocketmon
- Ptracemon
- Анализ eBPF в Astra Linux
- Заключение

Заключение

- Появились новые Linux-плагины для DRAKVUF
- В PT Sandbox для Astra Linux добавлен поведенческий анализ
- Есть много возможностей для дальнейшего развития



github.com/tklengyel/drakvuf

Полезные ссылки

Телеграм-чат PT Sandbox
t.me/ptsandbox

PT Sandbox
ptsecurity.com/ru-ru/products/sandbox/

Блог PT ESC Threat Intelligence
ptsecurity.com/ru-ru/research/pt-esc-threat-intelligence/

Услуги PT ESC
ptsecurity.com/ru-ru/services/esc/

Вопросы
webinar@ptsecurity.com

Алексей Колесников
akolesnikov@ptsecurity.com

Алексей Вишняков
avishnyakov@ptsecurity.com
Twitter: @Vishnyak0v