Final Report

Oliver Wilén, owilen@kth.se

Countered vulnerabilities

Direct Memory Access (DMA) is a process that allows a device, such as a hard drive or network card, to directly access system memory to read and write data. This allows devices to transfer data to and from system memory without going through the CPU, which can improve system performance. However, DMA also poses security risks because it allows devices to bypass security measures that are in place to protect the system and allow devices to access memory that they would normally not have access to.

Our project has focused on the vulnerabilities posed by a compromised driver of a network card. Specifically, we have worked with the virtio_net driver.

When instructing the network card to do a memory access, the virtio_net driver first creates a DMA descriptor, which is a data structure that contains information about the memory region that the network card should access. The DMA descriptor among other things includes the start address of the memory region and the length of the region in bytes. The driver then passes the DMA descriptor to the network card which can then use the information in the DMA descriptor to access the specified memory region and read or write the data at it. The network card does not have any ability to check if a received address is valid or should be prohibited and will thus blindly read and write to the address. If the virtio_net driver is compromised, it can therefore be used to instruct the network card to read and write to any address.

By compromising the driver and sending well-crafted packets to the network card we can therefore overwrite parts of the memory with the packet data. This could allow us to potentially overwrite security keys, password or crash the system among other things.

Similarly, a compromised driver can be used to leak data. The compromised driver can instruct the NIC to read from any address and the network card will then read from that address and send it out on the network. An adversary who has managed to compromise the driver and is able to capture the traffic coming from the NIC can therefore leak data from memory. Since the NIC collects the packet headers from the address as well, simply giving it a random address in memory will result in “invalid” packets being sent out. However, an adversary could first use the overwrite vulnerability discussed earlier, to overwrite parts of memory with valid headers which could result in leaked packets that can be sent directly to an adversary.

Design choices

In order to address the vulnerabilities outlined in the previous section, we propose implementing a monitor system. The primary role of this monitor will be to evaluate and verify the validity of addresses that are provided to the NIC by the driver and ensure that they are safe to read from or write to.

To accomplish this, the monitor will function as follows: when the driver wants to instruct the NIC to read or write to a particular address, instead of sending it directly to the NIC it will instead forward the DMA descriptor to our monitor. The monitor will then examine the information contained within the descriptor, including the address and size of the requested operation, and determine whether or not the request should be allowed. Originally the purpose of the monitor was to verify the validity of the addresses and also to stop compromised packages from being used. However, our current implementation only fulfills the first of those two criteria. Because of this, our monitor is only a proof of concept of the existence of such a monitor. Currently our monitor will issue a warning if an invalid address is passed to the network card. The monitor will also notify when a valid packet is processed, this is a debug print to show the monitor working and would be altered in a usable implementation.

This way the monitor can prevent the driver from giving the NIC invalid addresses but since the driver is assumed to be compromised, we also need to prevent the driver from simply bypassing the monitor and queuing the DMA descriptor directly as before. This can be done by revoking the driver’s access to the PCI bus, making it unable to communicate directly with the NIC. This does however prevent all communication between the driver and NIC and will require some functions which currently reside in the driver to be moved to the monitor.

Implementation details

Monitor

The monitor is designed to run as a server process within MINIX, and it will automatically start up when the operating system boots. Once the monitor server has initialized, it will continuously listen for IPC calls from other processes. When the monitor receives an IPC call, it will examine the message type and depending on the type of message received, the monitor will process the request and generate a response, which it will then send back to the calling process. This process of listening for and responding to IPC calls will continue indefinitely as long as the monitor is running.

The general function of the monitor is to receive DMA descriptors from the driver, evaluate whether the DMA descriptors contain valid memory access. We have created a wrapper library which handles creating the IPC calls and sending them to the monitor. The IPC calls use an added message layout for sending the data required by the monitor, currently, the payload does not exceed the 56-byte limit so we don’t have to use grants, but the message layout could easily be changed to instead send grants. The same message layout is used for all monitor functions, but unique layouts could be added for each.

Address Validation

When a process wants to send out a packet, the virtio_net driver will first copy the packet data into its own memory and then instruct the NIC to read from there. Similarly, when a packet is received the driver will give the NIC an address in its own memory to write the packet to before copying it to the destination processes memory. This means that the addresses that should be allowed to be passed along to the NIC should all reside inside the driver's memory.

The monitor can then determine whether a received DMA descriptor should be allowed by evaluating the address and size information contained within the DMA descriptor and checking whether the memory region defined by these values is contained within the memory allocated to the driver. This allows the monitor to validate the legitimacy of the request and ensure that the driver is not attempting to access memory that it is not authorized to use.

To do this the monitor will need to know what memory regions are allocated to the driver, which is known by the VM server. The monitor, therefore, queries the VM server and receives back all physical memory regions allocated to the driver. The VM already had a system call VM_INFO, and using the option VMIW_REGION will return the virtual regions of a given process. We, therefore, added another option to VM_INFO, VMIW_PHYS_REGION which instead returns all the physical regions of a given process.

Once the monitor knows the physical regions allocated to the driver, it can check if the address and the size in the DMA request are fully contained inside one of these regions and if it is the request is valid.

Virtio Library

For our implementation we added a call to our monitor in the virtio.c library instead of the virtio_net driver. This is because we cannot trust the driver to be uncompromised. However, the virtio_to_queue function that is altered is used by the Virtio block driver as well. This could create some issues. However, our monitor keeps track of the process that has called the virtio_to_queue function, and therefore will monitor if the block driver accesses any memory beyond its privileges. This would add some overhead to the block drivers functionality, but not break anything.

Explain why software counter the selected attacks

Our solution aims to address the vulnerabilities discussed in the previous section by introducing a monitor to act as an intermediary between the driver and the NIC. This monitor will evaluate all memory addresses being sent between the two and ensure that they are valid. We have added functionality to the VM server to fetch all physical address regions allocated to the driver, and the monitor will check if the address being sent belongs to these regions. By doing so, we can ensure that the address belongs to the driver and prevent it from accessing or altering data beyond its own memory.

Previously, the driver had the ability to instruct the NIC to read from and write to any address, which allowed the two vulnerabilities discussed earlier. This is a proof of concept, that the monitor is able to detect bad addresses and therefore it is possible to further build capabilities which will stop these attacks. As for now a notification of bad address access is our way of showing that an attack is happening.

For future work we would have to remove the driver's PCI access and therefore move some functionality from the driver to the monitor. This way, the driver has no other option than to communicate with the NIC through the monitor and can no longer bypass the monitor. So, while a compromised driver could still potentially disrupt the system by not sending or receiving data, it would no longer have the ability to access or alter data beyond its own memory - which was the primary goal of our project.

Individual Work

During the first weeks of the project, I had issues with compiling minix that took a while to solve. During this time, I was actively participating when the group was planning the project and writing the project specification. During the first implementations I was assisting my peers while they were coding. One could describe this as a form of pair programming. During this time, since I wasn't able to participate with the code, I became the contact point with Roberto by writing emails with the questions that we were having. When I managed to get my minix build working we had already come about halfway through our implementation. And therefore I asked the group for me to be more involved in the code. This was around the time where we started working with the grants. So my code can primarily be found in monitor.c and virtio_net.c. Other than writing code, I was debugging a lot to figure out why our grants weren't working. I created the slides for the design of the monitor in the presentation before new years. Since we never got the grants working, a lot of the work that I put in was never used. During the current implementation of our project, I worked with Sebastian on the new implementation of the monitor.