Exploiting USB/IP in Linux

Ignat Korchagin

ignat@cloudflare.com

@secumod
Who am I?

- systems engineer at Cloudflare
- interests in security and crypto
- enjoy low-level programming
- more builder than a breaker
- ... but try to stay alert
Agenda

- What is USB/IP
- USB/IP implementation in Linux
- Overview of sharing a USB device
- Vulnerable USB/IP code
- Potential exploit impact
- Hardening USB/IP setups
But first....
But first... Am I vulnerable?
What is USB/IP?
What is USB/IP?

- a way to share your USB devices over the network
What is USB/IP?

- A way to share your USB devices over the network
- Driver/device agnostic
What is USB/IP?

- a way to share your USB devices over the network
- driver/device agnostic
- sends URBs over TCP connection
What is USB/IP?

- a way to share your USB devices over the network
- driver/device agnostic
- sends URBs over TCP connection
- implemented for Linux and Windows
USB/IP implementation in Linux
USB/IP Linux implementation

Client

- usbp
- vhci-hcd
- usbip-core

Server

- usbipd
- usbip-host
- usbip-core
- USB hcd

user

kernel
USB/IP Linux implementation

Client

- usbp
- vhci-hcd
- usbp-core

Server

- usbipd
- usbp-host
- usbp-core

USB device

USB hcd
USB/IP Linux implementation

Client

- usbp
- vhci-hcd
- usbp-core

Server

- usbipd
- usbp-host
- usbp-core
- driver
- USB hcd
- USB device
USB/IP Linux implementation

Client

- usbip
- vhci-hcd
- usbip-core

Server

- usbipd
- usbip-host
- usbip-core
- driver
- USB hcd
- USB device
USB/IP Linux implementation

Client
- usbip
- vhci-hcd
- usbip-core

Server
- usbipd
- usbip-host
- usbip-core
- driver
- USB hcd
- USB device
USB/IP Linux implementation

Client

- usbip
  - vhci-hcd
  - usbip-core

Server

- usbipd
  - usbip-host
    - usbip-core
      - driver
      - USB hcd
      - USB device
USB/IP Linux implementation

Client

- usbip
- vhci-hcd
- usbip-core

Server

- usbipd
- usbp-host
- usbp-core
- driver
- USB hcd
- USB device

User

Kernel
USB/IP Linux implementation

Client

- usbp
- vhci-hcd
- usbp-core

Server

- usbpd
- usbp-host
- usbp-core
- USB hcd
- USB device

Drivers:
- driver

User/Kernel Interface:
- user
- kernel
$ usbip list -r 127.0.0.1

usbip: error: failed to open /usr/share/hwdata//usb.ids
Exportable USB devices
=========================
 - 127.0.0.1
  1-1: unknown vendor : unknown product
  (dead:beef)
    : /sys/fake/dangerous/usbipdemo
    : (Defined at Interface level) (00/00/00)
$ usbip list -r 127.0.0.1
usbip: error: failed to open /usr/share/hwdata//usb.ids
Exportable USB devices
======================
 - 127.0.0.1
   1-1: unknown vendor : unknown product
     (dead:beef)
       : /sys/fake/dangerous/usbipdemo
       : (Defined at Interface level) (00/00/00)

$ sudo usbip attach -r 127.0.0.1 -b 1-1
USB/IP Linux implementation

```
$ ps aux | grep usbip
root   884  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   886  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   887  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   888  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   889  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   890  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   891  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
root   892  0.0  0.0  0  0 ?    S  16:46   0:00 [usbip_eh]
ignat  895  0.0  0.0  14228  980 pts/1 S+ 16:46   0:00 grep usbip
```
USB/IP Linux implementation

Client
- Kernel
  - vhci-hcd
  - usbip-core

Server
- Kernel
  - usbip-host
  - usbip-core

get device list
USB/IP Linux implementation

Client
- Kernel
  - vhci-hcd
  - usbip-core
- usbp

Server
- Kernel
  - usbip-host
  - usbip-core
- usbp
- usbpd

get device list
import device
USB/IP Linux implementation

Client

Kernel
vhci-hcd
usbpip-core

socket fd

Server

Kernel

usbpip-host
usbpip-core

socket fd

Client

Kernel

usbpip

get device list

import device

Server

Kernel

usbpipd

socket fd
USB/IP Linux implementation

Client
- Kernel
  - vhci-hcd
  - usbip-core
- usbip

Server
- Kernel
  - usbip-host
  - usbip-core
- usbipd

get device list
import device
socket fd
URB traffic
socket fd
Vulnerable USB/IP code
USB/IP network protocol

USB/IP header | USB request block data

USB/IP network protocol

USB/IP header

USB request block data

length

USB/IP network protocol

static void vhci_recv_ret_submit(struct vhci_device *vdev,  
        struct usbip_header *pdu)
{
    ...
    
    /* unpack the pdu to a urb */
    usbip_pack_pdu(pdu, urb, USBIP_RET_SUBMIT, 0);

    /* recv transfer buffer */
    if (usbip_recv_xbuff(ud, urb) < 0)
        return;

    ...

static void vhci_recv_ret_submit(struct vhci_device *vdev,
        struct usbip_header *pdu)
{
    ...
    /* unpack the pdu to a urb */
    usbip_pack_pdu(pdu, urb, USBIP_RET_SUBMIT, 0);

    /* recv transfer buffer */
    if (usbip_recv_xbuff(ud, urb) < 0)
        return;

    ...

static void vhci_recv_ret_submit(struct vhci_device *vdev,  
                                       struct usbip_header *pdu)  
{  
    ...  
    /* unpack the pdu to a urb */  
    usbip_pack_pdu(pdu, urb, USBIP_RET_SUBMIT, 0);  
    /* recv transfer buffer */  
    if (usbip_recv_xbuff(ud, urb) < 0)  
        return;  
    ...
static void usbip_pack_ret_submit(struct usbip_header *pdu, struct urb *urb, int pack)
{
    struct usbip_header_ret_submit *rpdu = &pdu->u.ret_submit;

    if (pack) {
        ...
    } else {
        urb->status        = rpdu->status;
        urb->actual_length    = rpdu->actual_length;
        urb->start_frame    = rpdu->start_frame;
        urb->number_of_packets = rpdu->number_of_packets;
        urb->error_count    = rpdu->error_count;
    }
}
static void usbip_pack_ret_submit(struct usbip_header *pdu, struct urb *urb, int pack)
{
    struct usbip_header_ret_submit *rpdu = &pdu->u.ret_submit;

    if (pack) {
        ...
    } else {
        urb->status = rpdu->status;
        urb->actual_length = rpdu->actual_length;
        urb->start_frame = rpdu->start_frame;
        urb->number_of_packets = rpdu->number_of_packets;
        urb->error_count = rpdu->error_count;
    }
}
int usbip_recv_xbuff(struct usbip_device *ud, struct urb *urb)
{
    int ret;
    int size;

    if (ud->side == USBIP_STUB) {
        ...
    } else {
        ...
        size = urb->actual_length;
    }

    ...

    ret = usbip_recv(ud->tcp_socket, urb->transfer_buffer, size);
}
int usbip_recv_xbuff(struct usbip_device *ud, struct urb *urb)
{
    int ret;
    int size;

    if (ud->side == USBIP_STUB) {
        ...
    } else {
        ...
        size = urb->actual_length;
    }

    ...

    ret = usbip_recv(ud->tcp_socket, urb->transfer_buffer, size);
int usbip_recv_xbuff(struct usbip_device *ud, struct urb *urb)
{
    int ret;
    int size;

    if (ud->side == USBIP_STUB) {
        ...
    } else {
        ...
        size = urb->actual_length;
    }

    ...

    ret = usbip_recv(ud->tcp_socket, urb->transfer_buffer, size);
}
It is possible to write arbitrary length data to `urb->transfer_buffer`
It is possible to write arbitrary length data to `urb->transfer_buffer`

- `urb->transfer_buffer` is usually allocated either by USB core code or USB device driver
It is possible to write arbitrary length data to `urb->transfer_buffer`

- `urb->transfer_buffer` is usually allocated either by USB core code or USB device driver
- `urb->transfer_buffer` is allocated on request submit, so always assumes some maximum length
It is possible to write arbitrary length data to `urb->transfer_buffer`

- `urb->transfer_buffer` is usually allocated either by USB core code or USB device driver
- `urb->transfer_buffer` is allocated on request submit, so always assumes some maximum length
- According to USB/IP protocol the packet with “large” amount of data is valid
It is possible to write arbitrary length data to `urb->transfer_buffer`

- Introducing CVE-2016-3955
It is possible to write arbitrary length data to `urb->transfer_buffer`

- Introducing **CVE-2016-3955**
- CVSS base score: 9.8 (v. 3.0) and 10 (v. 2.0)
It is possible to write arbitrary length data to `urb->transfer_buffer`

- Introducing **CVE-2016-3955**
- CVSS base score: 9.8 (v. 3.0) and 10 (v. 2.0)
Victim has to actually use USB/IP
Requisites

- Victim has to actually use USB/IP
- Victim has to be a client in USB/IP terminology
● Victim has to actually use USB/IP
● Victim has to be a client in USB/IP terminology
● Victim has to “import” at least one USB device
Requisites

- Victim has to actually use USB/IP
- Victim has to be a client in USB/IP terminology
- Victim has to “import” at least one USB device
- Attacker either has to control USB/IP server or do a MiTM on the network
Demo
Potential exploit impact
Linux kernel heap exploit

- DoS: crash USB/IP client
● DoS: crash USB/IP client

● Data injection
Linux kernel heap exploit

- DoS: crash USB/IP client
- Data injection
- Code execution
Linux kernel heap exploit

- DoS: crash USB/IP client
- Data injection
- Code execution
  - (much harder with heap exploits, but still possible)
Linux kernel heap exploit

- DoS: crash USB/IP client
- Data injection
- Code execution
  - (much harder with heap exploits, but still possible)

https://jon.oberheide.org/blog/2010/09/10/linux-kernel-can-slub-overflow/
## Linux SLUB caches

<table>
<thead>
<tr>
<th>32</th>
<th>32</th>
<th>32</th>
<th>32</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>64</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>128</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Linux SLUB caches

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>128</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Linux SLUB caches

<table>
<thead>
<tr>
<th>32</th>
<th>32</th>
<th>32</th>
<th>32</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>64</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Find out which USB device drivers are using the same cache size as the desired object to be exploited
Linux SLUB caches

- Find out which USB device drivers are using the same cache size as the desired object to be exploited
- Emulate the device from the USB/IP server or by modifying USB/IP network traffic
Linux SLUB caches

- Find out which USB device drivers are using the same cache size as the desired object to be exploited
- Emulate the device from the USB/IP server or by modifying USB/IP network traffic
- Perform the buffer overflow
Hardening USB/IP setups
Hardening USB/IP setups

- Reconsider
Hardening USB/IP setups

- Reconsider
- Patch your system
Hardening USB/IP setups

- Reconsider
- Patch your system
- Protect your traffic (TLS, IPSec)
Hardening USB/IP setups

- Reconsider
- Patch your system
- Protect your traffic (TLS, IPsec)
  - even in intranet
Hardening USB/IP setups

- Reconsider
- Patch your system
- Protect your traffic (TLS, IPSec)
  - even in intranet
- Ensure your USB/IP server is trustworthy with proper ACLs
Resources

- https://pqsec.org/uboat-CVE-2016-3955/
- https://github.com/pqsec/ubootdemo
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-3955
- https://nvd.nist.gov/vuln/detail/CVE-2016-3955
Never sacrifice security for performance
  - extra buffer copy is not an excuse to move everything to kernel space

Validate your input

Consider least privilege principle
  - break code into modules
  - pay more attention to high-privileged code
Thank you