Feuriges Hacken - Spaß mit Firewire

Firewire als Penetrationsvektor

{becher,dornseif}@i4.informatik.rwth-aachen.de

RWTH AACHEN

Michael Becher, Maximillian Dornseif - Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen

Wer sind wir?

- **Lehr- und Forschungsgebiet Informatik 4 der RWTH Aachen**
- **Gegründet im Oktober 2003 für theoretische und praktische Forschung im Bereich Sicherheit**
- **Ausbildung in Sicherheit**
- **Forschung an Honeypots & Honeynets**
- **Lehre u.a. "Hacker-Seminar", "Hacker-Praktikum", "Summer School of Applied Security", "Computer Forensik", "PenTest-Praktikum"**
- **Netz:** `http://www-i4.informatik.rwth-aachen.de/lufg`
- **Aktuelle Informationen:**
  `http://mail-i4.informatik.rwth-aachen.de/mailman/listinfo/lufgtalk`

Ablauf

- **allgemeine Einführung in Firewire**
- **Details der Schwachstelle zeigen**
- **praktische Demonstration**
- **Möglichkeiten, den Zugriff zu verhindern**
- **Ideen für Forensik**

Was ist Firewire?

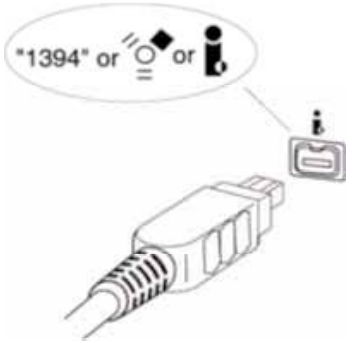
- **Apple Computers entwickelt es seit 1985**
- **IEEE 1394 (1995), IEEE 1394a (2000), IEEE 1394b (2002)**
- **von Apple vermarktet als Firewire oder FireWire**

- **von Sony vermarktet als i.Link**

- **serieller Bus, ähnlich wie USB, nur besser**
  - **schneller (800Mbit/s verglichen mit 480Mbit/s bei USB2)**
  - **peer-to-peer, braucht keinen Host-Rechner**
  - **liefert mehr Strom (1,5A verglichen mit 500mA bei USB2)**
  - **Unterstützung für isochrone Anwendungen (DV & Audio): garantierte Bandbreite**

Stecker & Kabel

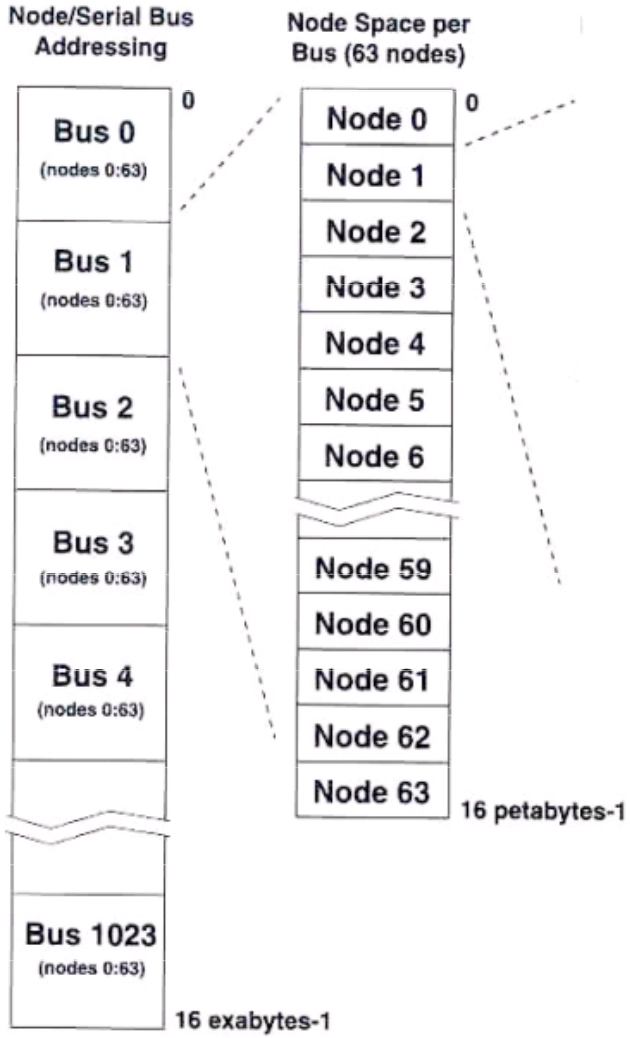
- **2 verschiedene (kompatible) Steckertypen**
  - **6-polig: Firewire**
  - **4-polig: i.Link**

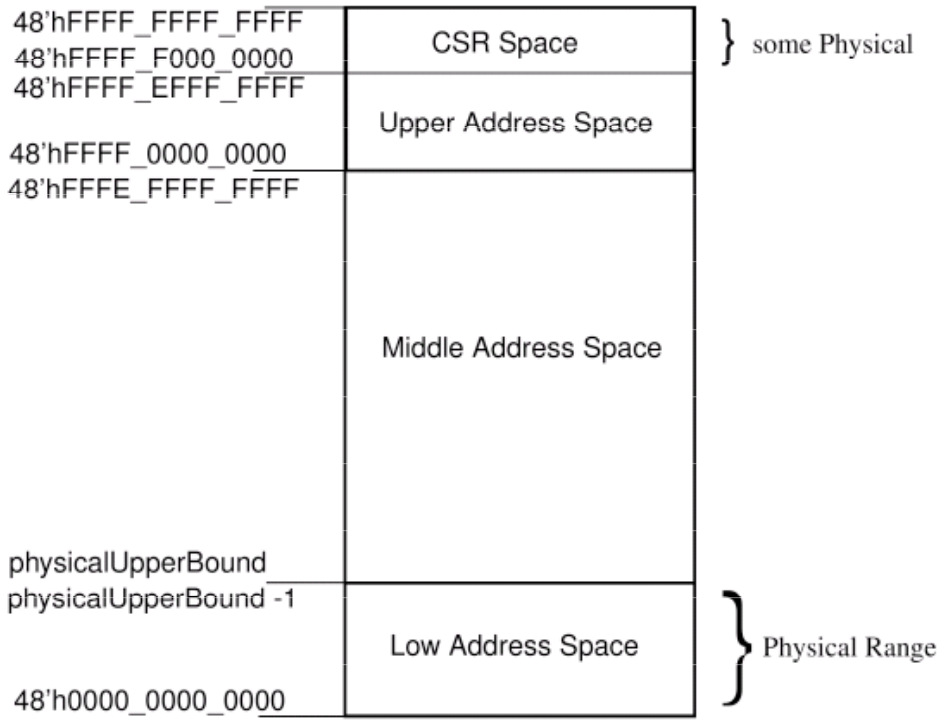


- **maximale Kabellänge: 4,5m**
- **maximal 63 Geräte am Bus, maximal 1023 Busse**
- **Bus ist Baumstruktur, maximal 16 Knoten Tiefe**

Speicheradressierung

- **64bit Unified Memoryspace: 10bit Bus ID, 6bit Node ID**

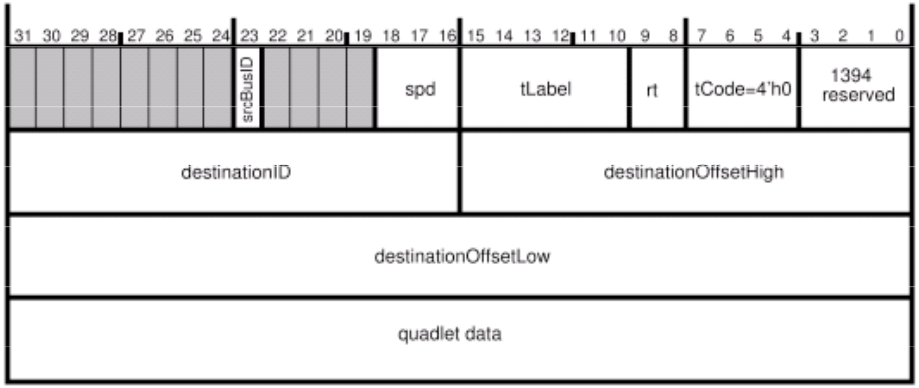




- **48bit Speicher pro Knoten = 256TB pro Gerät adressierbar**

Technische Details

- **hier geht es um asynchrone Pakete**
- **quadlet write request transmit:**



- **Speichereinheit 1 Quadlet = 4 Byte**
- **node = Gerät am Bus**
- **GUID = Global Unique ID**

- **weitere Details in "FireWire System Architecture"**
  (auch zu dem, was hier nicht gebraucht wird:
  Physikalische Schicht, isochroner Transfer, Initialisierung)

Die OHCI-Spezifikation

- **Open Host Controller Interface**
  `http://developer.intel.com/technology/1394/download/ohci_11.htm`
- **"When a block or quadlet read request or quadlet write request is received, the 1394 Open HCI chip handles the operation automatically without involving software if the offset address in the request packet header meets a specific set of criteria" (diese "criteria": später Details)**
- **"Low Address Space is from 48'h0 up to physicalUpperBound. Asynchronous read and write requests into this range can be handled by the Physical Request/Physical Response units, providing an efficient mechanism for moving asynchronous data."**
- **"Beweis": Debugging beim Linux OHCI1394-Treiber aktivieren & Logs ansehen**
  => *keine* Einträge für Adressen unter PhyUpperBound (=4GB)

Möglichkeiten Lesen

- **Bildschirmspeicher auslesen**
- **Speicher nach Zeichenketten absuchen**
- **Nach möglichem kryptografischen Schlüsselmaterial suchen**
- **Informationen über das System & Prozesse sammeln**

- **Firma nCipher hat Paper: Schlüssel-Material im Speicher finden**

  `http://www.ncipher.com/resources/downloads/files/white_papers/keyhide2.pdf`

- **manchmal reicht es auch schon, nur Zeichenketten zu finden**

Möglichkeiten Schreiben

- **System/Prozess zum Absturz bringen**
- **Bildschirminhalt verändern**
- **UID/GID von Prozessen verändern**
- **Code in einen Prozess injizieren**
- **einen zusätzlichen Prozess erzeugen**

Demonstrationen

- **Bildschirmspeicher lesen:**
  - **Text**
  - **Grafik**
- **Bildschirmspeicher schreiben: schwarzer Balken**
- **root-Rechte für einen Prozess setzen**

- **Angreifer: MacOS X.3**
- **Opfer: FreeBSD 5.3**

Code zum Lesen & Schreiben

MacOS:

```
IOCreatePluginInterfaceForService(self->aDevice, kIOFireWireLibTypeID,
                                  kIOCFPlugInInterfaceID, &cfPlugInInterface, &theScore);

(*cfPlugInInterface)->QueryInterface(cfPlugInInterface,
                     CFUUIDGetUUIDBytes(kIOFireWireDeviceInterfaceID), (void **)&fwIntf);

(*fwIntf)->Open(fwIntf);

(*fwIntf)->Write(fwIntf, self->aDevice, &fwaddr, (void *) buffer, &bufsize, false, 0);

(*fwIntf)->Read(fwIntf, self->aDevice, &fwaddr, (void *) buffer, &bufsize, false, 0);
```

Linux (libraw1394):

```
handle = raw1394_new_handle();

raw1394_set_port(handle, 0);

raw1394_write(handle, node_id, fwaddr, bufsize, (quadlet_t *) buf);
```

Bildschirm lesen

- **Text**
  - **Speicherlayout: Ein Zeichen alle 8 Byte**

MacOS

```
data = device.read(addrs[device.guid], 80*25*8)
```

```
(*fwIntf)->Read(fwIntf, self->aDevice, &fwaddr, buffer, &bufsize, false, 0);
```

- **Grafik**
  - **Parameter bleiben, zusätzlich bei Umwandlung in Bilddatei:**
    - **Anordnung der Farbwerte**
    - **Bitreihenfolge**

MacOS

```
data = device.read(pos, xres*bpp);
```

Bildschirm schreiben

- **schwarzen Balken einfügen**
- **Parameter:**
  - **Rechner (erkennbar an GUID)**
  - **Startadresse des Bildschirmspeichers (manuell)**
  - **Auflösung & Farbtiefe (manuell)**

Linux

```
unsigned char buf[2048]={0}; // Nullen für schwarze Farbe
for (i=start_address; i<start_address+write_length; i++) {
     raw1394_write(handle, node_id, i, 2048, (quadlet_t *)buf);
}
```

root-Rechte für einen Prozess setzen

- **Suchmuster aus FreeBSD-Kernelstrukturen extrahieren**
- **gesamten Speicher nach Muster absuchen**
- **wenn Suchmuster gefunden,**
  dann UID/GID ersetzen durch 0

Was funktioniert?

- **es funktioniert:**
  - **Lesen & Schreiben von MacOS-Rechnern**
  - **Lesen & Schreiben von FreeBSD-Rechnern**
  - **Schreiben von Linux-Rechnern**
- **es funktioniert nicht:**
  - **Lesen von Linux-Rechnern**
  - **Lesen & Schreiben von Windows-Rechnern**

Zugriff verhindern: OHCI Filter

- **Asynchronous Request Filters**
  "The 1394 Open HCI allows for selective access to host memory and the Asynchronous Receive Request context so that software can maintain host memory integrity. The selective access is provided by two sets of 64-bit registers: PhysRequestFilter and AsynchRequestFilter. These registers allow access to physical memory and the AR Request context on a nodeID basis." (OHCI Standard)

- **PhysicalRequestFilter Registers**
  "If an asynchronous request is received, passes the AsynchronousRequestFilter, and the offset is below PhysicalUpperBound (section 5.15), the sourceID of the request is used as an index into the PhysicalRequestFilter. If the corresponding bit in the PhysicalRequestFilter is set to 0, then the request shall be forwarded to the Asynchronous Receive Request DMA context. If however, the bit is set to 1, then the request shall be sent to the physical response unit." (OHCI Standard)

- **Anfragen in diesen asynchronen DMA-Kontexten sieht der Treiber und kann frei darüber entscheiden**

OHCI Filter II

Kurzfassung:

- `AsynchronousRequestFilterHi`, `AsynchronousRequestFilterLo`:
  Jedes auf 0 gesetzte Bit verhindert generell jede Zugriffsmöglichkeit eines Knotens auf den lokalen Knoten

- `PhysicalRequestFilterHi`, `PhysicalRequestFilterLo`:
  Jedes auf 0 gesetzte Bit verhindert jede direkte Zugriffsmöglichkeit eines Knotens über die `physical response unit` **des lokalen Knotens**

- `PhysicalUpperBound`:
  Entscheidet über die maximale Adresse, auf die direkte Zugrifsserlaubnis erteilt wird (default mit 4GB initialisiert)

Zugriff verhindern: Linux

- **Quellcode Linux-Treiber**

ohci1394.c

```
/* Accept Physical requests from all nodes. */
reg_write(ohci,OHCI1394_AsReqFilterHiSet, 0xffffffff);
reg_write(ohci,OHCI1394_AsReqFilterLoSet, 0xffffffff);
/* Turn on phys dma reception.
 *
 * TODO: Enable some sort of filtering management.
 */
if (phys_dma) {
      reg_write(ohci,OHCI1394_PhyReqFilterHiSet, 0xffffffff);
      reg_write(ohci,OHCI1394_PhyReqFilterLoSet, 0xffffffff);
      reg_write(ohci,OHCI1394_PhyUpperBound, 0xffff0000);
} else {
      reg_write(ohci,OHCI1394_PhyReqFilterHiSet, 0x00000000);
      reg_write(ohci,OHCI1394_PhyReqFilterLoSet, 0x00000000);
}

DBGMSG("PhyReqFilter=%08x%08x",
       reg_read(ohci,OHCI1394_PhyReqFilterHiSet),
       reg_read(ohci,OHCI1394_PhyReqFilterLoSet));
```

=> Modul-Option 'phys_dma=0' für Modul ohci1394 benutzen!

Zugriff verhindern: MacOS

- **Quellcode MacOS-Treiber**

IOFireWireController.cpp

```
IOFWSecurityMode mode = kIOFWSecurityModeNormal;

OSString * securityModeProperty = OSDynamicCast( OSString,options->getProperty("security-mode") );

if( securityModeProperty != NULL &&
strcmp("none", securityModeProperty->getCStringNoCopy()) != 0 )
{
      // set security mode to secure/permanent
      mode = kIOFWSecurityModeSecurePermanent;
}
```

und dadurch später:

```
// shut them all down!
fFWIM->setNodeIDPhysicalFilter( kIOFWAllPhysicalFilters, false );
```

=> OpenFirmware security mode auf etwas anderes als 'none' setzen

- **Aber: Feature ist undokumentiert**

Forensik

Einander ausschließende Probleme:

- **Rechner ausschalten und eine post mortem-Analyse machen**
  - **zerstört Informationen über laufende Prozesse**
- **Informationen auf dem laufenden System sammeln**
  - **kann die wichtigen Beweise verwischen**
- **Problem wird gelöst, wenn durch Firewire ein Speicherabbild ohne zusätzliche Software auf dem Rechner möglich ist**

Forensik II

- **Firewire-Ports an Servern vorsehen**
  - **Zugriff nicht per OHCI Filter filtern**
  - **Aber: Anschlüsse physisch vor Missbrauch schützen**
- **System bereithalten, um memory dump über Firewire zu ziehen**

Forensik III - Ausblick

Herausforderung für Forensik:

- **Zuordnen des physikalischen Speichers zum logischen Speicherlayout (memory pages)**

- **Nächster Schritt:**
  Implementierung von /dev/kmem auf Firewire

Schlussfolgerungen

- **Firewire-Buchsen vor physischem Zugriff schützen**
- **nur vollkommen vertrauenswürdige Geräte an die Firewire-Anschlüsse lassen**
- **OHCI-Filter implementieren oder Workarounds benutzen**