The Jedi Packet Trick takes over the Deathstar

(or: "taking NIC backdoors to the next level")

Arrigo Triulzi
arrigo@sevenseas.org

Previously on Project Maux

- 2006-2007 "the early years"
  - find out by accident about NIC offloading of checksum routines...
    ⇒ can we hook something to that?

  - Broadcom's Tigon firmware says it is based on MIPS, the firmware is downloadable from the Internet, there is no firmware installation security and I happen to have a DECstation 3000 in the basement...
    ⇒ dbx and go!

  - transform a few cards into doorstops and eventually hook the IP checksum...
    ⇒ 5 second sniffer in a circular buffer

Previously on Project Maux

root, firmware, OS-independent

- 2007-2008 "Mummy, mummy, I want a shell!"
  - nVidia releases the CUDA development toolkit
    ⇒ the GPU becomes interesting
  - PCI-to-PCI transfers are not marshalled by the OS
    ⇒ PCI-to-PCI between the NIC and the GPU
  - the NIC gets to see the packets first
    ⇒ we use the checksum hook to interpret and forward
  - a PCI card has DMA over the whole RAM
    ⇒ we can play in memory and the OS shall never know

Previously on Project Maux

- 2007-2008 net result:
  - nicssh

```
archimede:~/nicssh$ nicssh 10.4.4.233
Connecting to 10.4.4.233
ICMP Echo Reply from OS - no nicfw
archimede:~/nicssh$ nicssh 10.4.4.234
Connecting to 10.4.4.234
ICMP Echo Reply from nicfw (Windows system)
Requesting tcp/80 with cloaking
nicssh> ?
help memory* sniff* send* reboot cleanup
quit
nicssh>
```

- no installer
- no GPU persistence

Stop press: it is called ASF/RCMP and is shipped by default with Broadcom cards!

Allez Loïc!

Previously on Project Maux

- 2007-2008 "now what?"
  - Jedi Packet Trick: if I have two (vulnerable) NICs in a card what can I possibly do? How about sending packets between them over the PCI bus?
  - Driver Takeover aka "attack from below": drivers tend to assume that NICs will not attack them...
  - Installation and Persistence: "hey, click here to use my new firmware" only works a few times, not at every boot...

"I have a cunning plan" (once again)

- Jedi Packet Trick: "easy"
  - take over NIC1, inject nicssh, use nicssh to take over NIC2
  - magic packets travel between NIC1 and NIC2 over the PCI bus
- Driver Takeover: OS-dependent, not for me
- Installation: remote factory diagnostics
- Persistence: EFI module

Some preliminary notes

- This is still not a funded project but personal curiosity driven what-if research,
- I am using the old stock of NICs I bought back in 2008 to replace the doorstops,
- The old motto still holds:
  Given no prior knowledge, "the Internet", a cheap 10-pack of NICs and a PC can we develop the ultimate rootkit?

Deathstar Mk.I design

vulnerable

- EAL level $1 \cdot 10^{23}$ firewall with two NICs
- NIC1 is the external interface
- NIC2 is the internal interface
- nVidia GPU
- EFI BIOS

But of course, all firewalls have gaming GPUs!

... and an EFI BIOS!

Jedi Packet Trick

- nicssh extended with:
  - "findnic" to find other NICs on the system by scanning the PCI bus (lifted code from BSD)
  - "grabnic" to take it over by injecting the modified firmware into the other card simulating an OS pushing new firmware
  - "forward" to set up forwarding: like good old overlays it never returns... this turns nicssh into a two-way pipe between NICs. All magic packets are forwarded between NICs.
- nice but... it requires a suitable GPU

Jedi Packet Trick

```
archimede:~/nicssh$ nicssh 10.4.4.230
Connecting to 10.4.4.230
ICMP Echo Reply from nicfw (Linux system)
Requesting tcp/80 with cloaking
nicssh> ?
help memory* sniff* send* findnic* grabnic* forward*
reboot cleanup quit
nicssh> findnic 0 3 21
Hunting on bus0... nope
Hunting on bus3...
3:0:0: Tigon
5:0:0: Tigon
Hunting on bus21...
21:0:0: Intel 82571EB
21:0:1: Intel 82571EB
nicssh> grabnic 3:0:0
My man, it already runs nicfw!
nicssh> grabnic 5:0:0
Trying...done
nicssh> forward 3:0:0 5:0:0
Forwarding starting - shell being replaced now
I'm afraid. I'm afraid, Dave. Dave, my mind is going. I can feel it...
```

We want NIC-to-NIC!

We want NIC-to-NIC!

Jedi Packet Trick

- NIC-to-NIC requires
  - installation
    - PCI bus scan to locate other NICs
    - initiating a firmware update
    - pushing firmware to the other NIC
  - communication
    - PCI-to-PCI device data transfer
    - suitable marshalling of the above

Jedi Packet Trick

- The problems begin...
  - the space in the firmware is not huge so a PCI bus scanner is not really on the books
  - where do I get the firmware image from?
  - how do I efficiently push it?
  - (assuming we solve the above) how do we make sure our firewall bypass is not too obvious?

Jedi Packet Trick

Type:	Ethernet Controller
Bus:	PCI
Vendor ID:	0x10de
Device ID:	0x0ab0
Subsystem Vendor ID:	0x10de
Subsystem ID:	0xcb79
Revision ID:	0x00b1

- Firmware is small...
  - don't perform a true PCI bus scan but cheat: we are looking for cards which look like us so how about we restrict ourselves to the identifiers we want?
- and the image is large...
  - so why don't we just copy our own image over since we are the same NIC as we only scanned for close relatives?

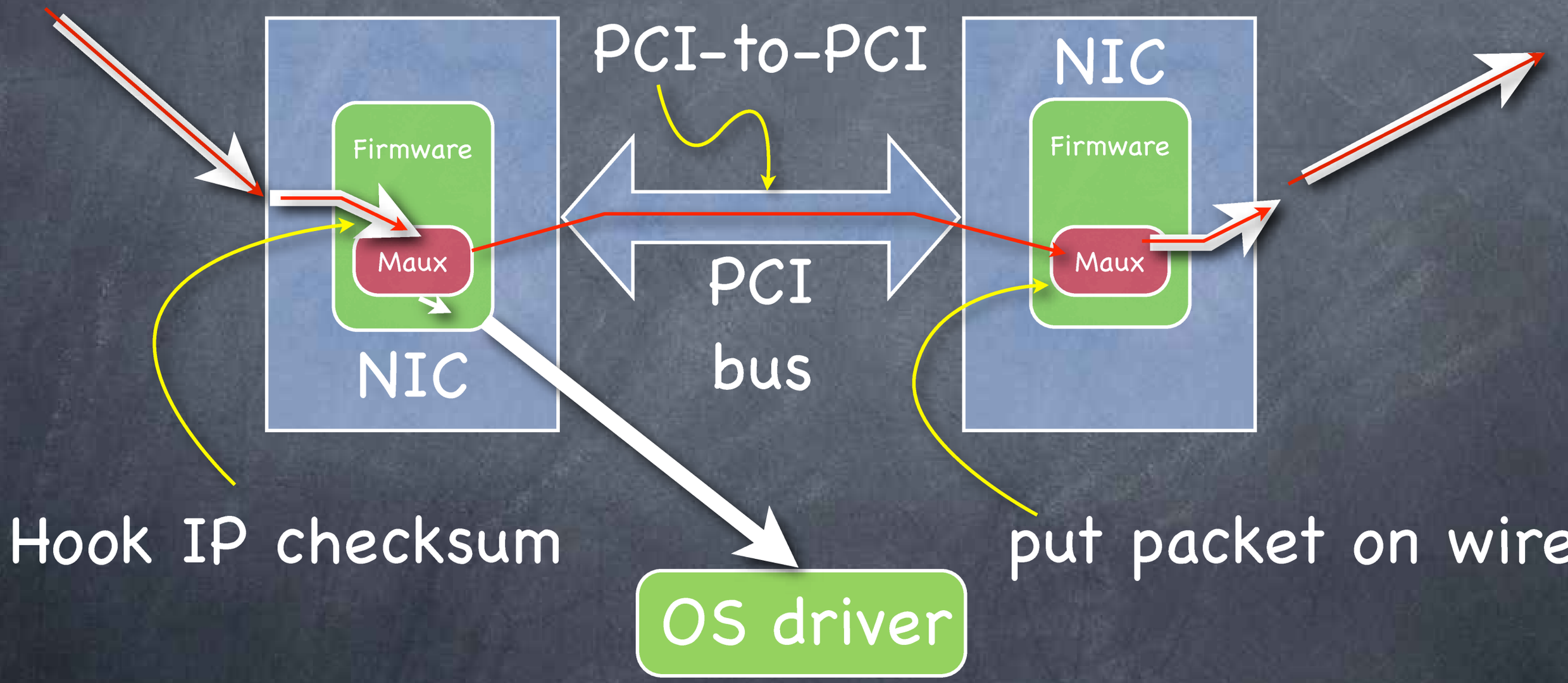
Jedi Packet Trick

- Now we need to push the image: this takes time and as it happens the NIC is non-responsive...
  - we could wait for a reboot and push it as part of our NIC initialisation routine?
  - wait for a quiescent time and then push it?
- just make the firewall hang for 30 seconds and who cares!

Jedi Packet Trick

- PCI-to-PCI transfer is simply a replacement of the nicssh channel with one to the other NIC, everything else stays the same.
- Stealth is our middle name so we cannot afford to have our NIC-to-NIC channel reduce the performance of the firewall:
  - rate-limit the NIC-to-NIC channel to approx. 64kbps. This is empirically slow enough that the kink during heavy load is not too noticeable.
  - if the transfer rate from the driver starts getting heavy simply shut the channel down.

Jedi Packet Trick



Jedi Packet Trick

- So what?
  - complete bypass of any OS-based firewall as the OS is oblivious to the traffic being passed between the NICs
- How do you catch it then?
  - timing analysis is one possibility: the hidden channel will rob your firewall of performance
  - IDS on both sides testing the validity of the ruleset

Installation

- Project Maux Mk. I & II suffered from a major drawback: installation required admin privileges to run the firmware update.
- At the same time looking at the firmware there were hints of a "remote update" capability (at least in the cards I have).

Installation

- The "remote update" capability appears to be linked to some sort of factory testing. Once it completes it initiates a factory test on the firmware.
- The "remote update" works by sending a WOL followed by a UDP packet in a special format containing a header followed by as many UDP packets as needed for the firmware.

Installation

- UDP is good and bad...
  - easy to spoof: we could send our firmware updates from anywhere on the Internet
  - hard to confirm that the packet got there: how do we know that all the parts of the update got to the card?
- Net result: it works in the lab.

Installation

```
archimede:~/nicssh$ sudo nicssh -i nicfw.bin 10.4.4.230
Sending UDP magic to 10.4.4.230...done
ICMP Echo Reply from nicfw (Linux system)
Injection successful
archimede:~/nicssh$ nicssh -gi gpussh.bin 10.4.4.230
Connecting to 10.4.4.230
Preparing to send GPU code
ICMP Echo Reply from nicfw (Linux system)
Requesting GPU RAM injection
Sending GPU code...done
archimede:~/nicssh$ nicssh 10.4.4.230
Connecting to 10.4.4.230
[...]
```

Installation

- But is remote installation so important?
  - probably not: we have plenty of remote vectors which can be used to push the firmware at the OS level
- once the initial install is done we can leverage the remote firmware update capability of the Jedi Packet Trick

Persistence

(or "there's a 2006 Intel iMac there doing nothing!")

- Intel iMacs come with EFI
- EFI is modular
- Apple has an EFI Dev Kit on their Developer Connection...
- Why don't I write an EFI module to load the NIC firmware and nicssh 2.0?
- Why don't I hide the EFI module on the IDE/SATA disk?

Persistence

- EFI module design
  - responsible for loading NIC firmware
  - responsible for installing nicssh in GPU
  - responsible for maintaining hidden location on IDE/SATA disk
- All of the above still under development

Persistence

- What works?
  - EFI module which loads NIC firmware
  - EFI module which loads nicssh
- What doesn't work?
  - storing it on the IDE/SATA disk
  - loading the EFI module correctly
- Looking at PGP WDE for OS X design for ideas...

Putting it all together

- A staged attack against a firewall
  - "remote update" over UDP to NIC1
  - firmware update of NIC2
  - push EFI module into SATA disk to defend against NIC reflashing (we reflash it too!)
  - Initiate Jedi Packet Trick

What now?

- Attack cards with proper firmware security:
  - crypto vulnerabilities
  - bad key management (OEMs come to mind)
  - remote management (nice one Loïc!)
- Go further... how about CPU μcode?

µcode

- Yet another cunning plan...
  - most modern CPUs have µcode update functionality to patch errata in the CPU implementation
  - there is often more than one µcode update released during the lifetime of the CPU
  - each CPU µcode update has to contain the previous ones plus the new one
  - each erratum is known as manufacturers publish them
  - each µcode update states which errata are fixed by it

μcode

- Let's say that your family contains a former μcode guru
- then, in theory, he could figure out some μcode for a given set of errata
- you could then look at the μcode before and after the errata and, in theory, you could have some known plaintext with which to attack the μcode update...

μcode

- let's say that with enough known plaintext you recover the encryption key... then you could start modifying μcode for injection
- now you have to worry about reboots as μcode updates are not persistent across reboots (which is good for testing!)
- EFI comes to the rescue there...

μcode

- The ultimate hack
  - modify the NIC firmware remotely
  - install nicssh for backdooring
  - push modified μcode which blocks anything we believe hostile (e.g. using the GPU memory where our nicssh lives)
  - install EFI module for persistence of all the above

Thanks

- My family ∀ their ∞ patience while I play with my toys (and for having a μcode guru in it),
- Toby, Ryan and Brian for keeping the hard questions coming,
- Maya for project naming,
- Icars for being my ever-present simulacrum,
- $C_8H_{10}N_4O_2$

References

- Broadcom firmware development kit: http://www.broadcom.com/products/communications_processors_downloads.php
- Papers by John Heasman (ACPI, BIOS and PCI rootkits):
  - http://www.nextgenss.com/research/papers/Implementing_And_Detecting_A_PCI_Rootkit.pdf
- Network Interface Firmware Back Door with Tigon2, eEye Industry Newsletter, 25th April 2007, http://www.eeye.com/html/resources/newsletters/vice/VI20070425.html
- A. Singh, Mac OS X Internals, Addison-Wesley, 2006, http://osxbook.com/book/bonus/chapter4/efiprogramming/
- Rowan Atkinson, "Blackadder", BBC TV series.