Can you still trust your network card?

Loïc Duflot, Yves-Alexis Perez, Guillaume Valadon, Olivier Levillain

Agence Nationale de la Sécurité des Systèmes d’Information
Modern network cards

They do not only connect the host to the network.

- Hardware architectures are complex:
  - several processors,
  - different kind of memories,
  - multiple network interfaces;
- Embedded softwares (firmwares) do more than you think:
  - remote administration: ASF, IPMI, AMT, etc,
  - TCP segmentation offloading,
  - radio with temporal constraints: GSM, 802.11, etc.

http://www.ssi.gouv.fr/trustnetworkcard
Impacts on security

If an attacker can execute arbitrary code on the card, she can do virtually anything:

- stop processing packets;
- drop some packets;
- ARP/DNS cache poisoning;
- implement SSLstrip-like attacks;
- attacks hosts on the LAN;
- replace the firmware;
- attack the host (read/write access to the main memory).

A "must read" on this topic:
Arrigo Triulzi, PacSec08, "Project Maux Mk.II"

http://www.ssi.gouv.fr/trustnetworkcard
What will be described today

We will present:

- architectures of modern network cards;
- remote administration protocols used in these cards;
- an actual vulnerability that we discovered;
- tools developed to debug a *Broadcom NetXtreme* card;
- exploitation proof of concept and demo;
- mitigations and workarounds to this attack.
What won’t be described today

This is not about:

▶ driver bugs;
▶ OS vulnerabilities.

And please note that:

▶ we won’t provide the network packets and the tools used in the demo;
▶ we worked with the vendors and they issued a patch for the vulnerability.

▶ CVE-2010-0104: *HP Small Form Factor or Microtower PC with Broadcom Integrated NIC Firmware, Remote Execution of Arbitrary Code*

http://www.ssi.gouv.fr/trustnetworkcard
Internal architecture of a network card

- **PHY**: send and receive signals on the wire;
- **DMA-engine**: exchange packets with the host;
- **negotiation and link control (speed, duplex)**, etc.
The NetXtreme architecture

Internal architecture of NetXtreme network card

- performs various operations on packets;
- offloads work from host;
- needs hardware and software to do that;
- runs as an intercepting proxy.
RX RISC

An on-chip RISC processor is provided for running value-added firmware that can be used for custom frame processing. The on-chip RISC operates independently of all the architectural blocks; essentially, RISC is available for the auxiliary processing of data streams.

- a MIPS CPU on the card;
- it has access to major components:
  - shared memory,
  - incoming and outgoing packets,
  - PCI Configuration Space,
  - SMBus;
- it executes a firmware.
The firmware in the NetXtreme

Different firmwares:
- ASF (Alert Standard Format protocol);
- TSO (TCP Segmentation Offloading).

The firmware is:
- loaded from an EEPROM;
- or by the driver from the filesystem:
  - Linux driver only has TSO,
  - Windows drivers rarely have a firmware,
  - the firmware seems protected;
- loaded to memory (SRAM) during execution.
Internal memory

- the card internal memory is mapped to the host one;
- internal memory can be accessed through a 32kb window;
- this window can be moved to read the whole internal memory space from the host.
Alert Standard Format (ASF) 1.0 1/3

ASF

- transmits alerts/events using the network:
  - hard disks failures, BIOS errors, ...
  - heartbeats ("machine is up");
- must operate if everything else fails (dead hard disks, OS).

The network card receives events from others devices using the SMBus (System Management Bus).

RMCP

- ASF uses a protocol called Remote Management and Control Protocol;
- RMCP can query system state;
- RMCP allows to remotely start, stop or reboot computers.
Remote administration protocols

Alert Standard Format (ASF) 1.0 (2/3)

- firmware parameters must be configured: IP address, netmask, heartbeats frequency,
  - a tool is provided by network cards vendors;
- a specific ACPI table is used by ASF;
- ASF can be deactivated from the BIOS on some hardware;
- at least one boot on an ACPI-enabled OS is mandatory.
Remote administration protocols

ASF 1.0 (3/3)

Security

- no security interfaces;
- vendors are discouraged to implement their own, proprietary security interface;
- security issues should be addressed at network infrastructure level.

SGDSN/ANSSI – [http://www.ssi.gouv.fr/trustnetworkcard](http://www.ssi.gouv.fr/trustnetworkcard)
ASF configuration

- IP addresses;
- RMCP;
- permissions;

[Image of Broadcom ASF Configuration window]

[Website link: http://www.ssi.gouv.fr/trustnetworkcard]
Remote administration protocols

Alert Standard Format (ASF) 2.0

ASF 2.0 adds a new protocol: RSP

- RMCP security-extensions protocol;
- adds authentication and integrity protection;
- no encryption.

This presentation focuses on ASF 2.0.
RMCP in ASF 2.0

- messages are sent over UDP;
- traffic must be either sent on:
  - the legacy port 623/udp: no authentication, no integrity,
  - the secure port 664/udp: RMCP messages are carried inside *RMCP Security-Extensions Protocol* (RSP);
- the network card grabs traffic on these ports, analyzes RMCP packets, and replies to queries.

The network card must implement the following stack: IP/UDP/RSP/RMCP.
Remote administration protocols

Protocols stacks

**RSP Session Protocol (RSSP)**

**RSSP Authentication and Key-Generation Protocol (RAKP)**

**RMCP Security-Extensions Protocol (RSP)**

Remote Management and Control Protocol (RMCP)

User Datagram Protocol (UDP)

Internet Protocol (IP)

IEEE 802.3 / Ethernet Protocol

[Link: http://www.ssi.gouv.fr/trustnetworkcard]
RMCP Security-Extensions Protocol (RSP)

- RSP adds mutual authentication of the remote console and the client;
  - the console is the management device used by the administrator,
  - the client is the remote workstation;
- messages are authenticated using pre-shared HMAC-SHA1 keys;
- in order to send a message on the secure port, the console must:
  - open a session,
  - negotiate a session key (three messages are exchanged),
  - send the RMCP message protected with the sessions key over RSP.
RSP session protocol (RSSP)

Console

OpenSessionRequest

OpenSessionResponse

RAKP1

RAKP2

RAKP3

Client
The RMCP messages

- Presence Ping / Presence Pong;
- CapabilitiesRequest / CapabilitiesResponse; describe which operations are possible on the legacy and the secure ports;
- System State Request / System State Response; ask system about status, last boot medium etc.
- startup request (device can be specified: PXE, hard disk, CD-ROM);
- reboot request;
- stop request.
Status of RMCP support

Some hardware with ASF support that we tested:

- HP Compaq dc7600
  - on the secure port, start/reboot/stop messages are processed;
- DELL Latitude D530 and Precision T5400
  - CapabilitiesRequest messages are processed,
  - CapabilitiesReply messages indicate that no remote administration function is supported.

Remarks

- no vendor enabled remote administration on the legacy port;
- why do some vendors disable administration functions but still implement RMCP and SMBus functions?

http://www.ssi.gouv.fr/trustnetworkcard
Card’s behavior when using ASF

When it receives a packet, the card

- intercepts the packet;
- before transmitting it to the OS;
- checks if it is a RMCP packet;
- process it:
  - open/close session,
  - send system state informations,
  - perform system administration tasks,
  - the packet is NOT transmitted to the host.
Protocol analysis

Protocol security (1/2)

Potential issues

▶ protocol uses 160bit pre-shared keys, which means all clients might have the same keys;
▶ messages are integrity protected but the integrity pattern does not include message ID;
▶ in order to act as the console, an attacker just has to send a RAKP\textsubscript{3} with a valid HMAC.

Exploitation

▶ is it possible to forge the HMAC?
▶ can the client act as a integrity oracle?
Protocol security (2/2)

Not so easy

- only two concurrent sessions on the implementation we tested;
- not all fields are under the attacker’s control;
- fields size problems.
Implementation problems (1/2)

Interesting fields under the attacker’s control

- management console username (RAKP₁ message);
- management console session ID (Open Session Request).

Trying to play with them, messing with

- size;
- content.
Implementation problems (2/2)

Username

- specifications limit the size to 16 chars, without NULL;
- the username size is coded using a 1 byte field.

What if we don’t play nice?

- "the card crashes";
- the host can only send Ethernet frames but not receive them.

What did exactly happen?
Proof of concept

- Is the vulnerability serious?
- What are the direct and indirect consequences?
- How can we build a *proof of concept*?

Speaker switch
Instrumenting the card (1/2)

How did we find out what happened?

▶ what is really crashing?
▶ how is it crashing?

Using the NetXtreme specifications

▶ public specs available for open-source developers;
▶ describe the internal card behavior;
▶ give informations about RX RISC components.
How to debug the RX RISC? We need to:

- follow the execution flow;
- know registers contents;
- know why the CPU stops;
- trace data.

We need a network card debugger.
What information can we retrieve from the card?

Specifications and experiments say we have access to:

- RX RISC mode register;
- RX RISC state register;
- RX RISC program counter;
- RX RISC hardware breakpoint register;
- some general registers.

We can use this knowledge to build that debugger.
Our homemade debugger:

- uses information from mapped registers;
- runs in step-by-step mode;
- can perform register/memory tracking;
- can break on register/memory access;
- can do some pattern matching.
Help

What should I do next (h for help)? h

Usage:
'a' -> Advance n steps
's' -> Advance 1 step
't' -> Trace
'c' -> Continue
'C' -> Continue (step-by-step)
'g' -> Break on instruction
'R' -> Break on pattern in register
'S' -> Break on pattern in stack
'H' -> Break on pattern in internal memory
'M' -> Break on pattern in external memory
'n' -> Break on next pattern in stack
'l' -> Break on specific memory access
'm' -> Break on any memory access
'j' -> Break on register write
'i' -> Break on instruction
'T' -> Track register
'L' -> Track memory address
'Z' -> Track specific memory zone access
'I' -> Track pattern in memory
'P' -> Track pattern
'x' -> Clear tracking
'f' -> Find pattern in internal memory
'F' -> Find pattern in external memory
'A' -> Find all patterns in external memory
'd' -> Display memory address
'D' -> Display memory area
'w' -> Write a word to memory address
'r' -> Reset CPU
'q' -> Quit

SGDSN/ANSSI – [http://www.ssi.gouv.fr/trustnetworkcard](http://www.ssi.gouv.fr/trustnetworkcard)
CPU

********** Instruction
Instruction = 3c020001 LUI r2 = 00010000
Last memory access = 00000000
********** CPU Status Registers ******
RXPC = 00011078 RXHWBRK = 0000001d
RXMODE = 00009db0 RXSTATE = 80001400
The vulnerability

Instrumenting the card

General registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>00000000</td>
</tr>
<tr>
<td>$1</td>
<td>00010000</td>
</tr>
<tr>
<td>$2</td>
<td>00000000</td>
</tr>
<tr>
<td>$3</td>
<td>40000000</td>
</tr>
<tr>
<td>$4</td>
<td>0001b4b8</td>
</tr>
<tr>
<td>$5</td>
<td>0001b8e6</td>
</tr>
<tr>
<td>$6</td>
<td>00000000</td>
</tr>
<tr>
<td>$7</td>
<td>0001bfc4</td>
</tr>
<tr>
<td>$8</td>
<td>00000040</td>
</tr>
<tr>
<td>$9</td>
<td>00000050</td>
</tr>
<tr>
<td>$10</td>
<td>0001b8bc</td>
</tr>
<tr>
<td>$11</td>
<td>0001bfc0</td>
</tr>
<tr>
<td>$12</td>
<td>80000000</td>
</tr>
<tr>
<td>$13</td>
<td>00000001</td>
</tr>
<tr>
<td>$14</td>
<td>00000000</td>
</tr>
<tr>
<td>$15</td>
<td>ffffffff</td>
</tr>
<tr>
<td>$16</td>
<td>a4020000</td>
</tr>
<tr>
<td>$17</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>$18</td>
<td>00000000</td>
</tr>
<tr>
<td>$19</td>
<td>0001af48</td>
</tr>
<tr>
<td>$20</td>
<td>0000ad60</td>
</tr>
<tr>
<td>$21</td>
<td>018004f1</td>
</tr>
<tr>
<td>$22</td>
<td>000000fc</td>
</tr>
<tr>
<td>$23</td>
<td>00010000</td>
</tr>
<tr>
<td>$24</td>
<td>ffffffff</td>
</tr>
<tr>
<td>$25</td>
<td>80000000</td>
</tr>
<tr>
<td>$26</td>
<td>00000b50</td>
</tr>
<tr>
<td>$27</td>
<td>00011104</td>
</tr>
<tr>
<td>$28</td>
<td>c0000000</td>
</tr>
<tr>
<td>$29</td>
<td>0001bfd8</td>
</tr>
<tr>
<td>$30</td>
<td>0001c000</td>
</tr>
<tr>
<td>$31</td>
<td>000111f8</td>
</tr>
</tbody>
</table>

SGDSN/ANSSI – [http://www.ssi.gouv.fr/trustnetworkcard](http://www.ssi.gouv.fr/trustnetworkcard)
Stack

******** Stack ********************************************
Stack pointer: 0001bfd8 (max) stack size: 10
Stack bottom: 0001c000

******************************************************
0001bff4:0001a000 0001bfe0:0000ad60
0001bff8:00010044 0001bfe4:0001a918
0001bffc:73ffffff 0001bfe8:00010e00
0001bfec:0001a80c 0001bfdc:0001a000
0001bffe:0000ad64 0001bfe0:0000ad60

The vulnerability

Instrumenting the card

ANSSI

http://www.ssi.gouv.fr/trustnetworkcard
Why does the card crash?

RX RISC state register provides useful information:

1. bad memory alignment
2. invalid instruction fetch (jump to invalid location);
3. invalid data access (load/store in invalid location);
4. invalid instruction;

Points 2 and 4 can mean direct flow execution redirection. Points 1 and 3 can mean indirect flow execution redirection (try to overwrite a return address in the stack).
Changing the execution flow

When the RX RISC CPU is crashing, an attacker needs to:

▶ find the source of the data;
▶ tune it to fit her needs.

Trials and errors
We managed to:

▶ make the *username* field overflow;
▶ overwrite a return address in the stack with an address under our control.
Proof of concept code injection (1/2)

On this particular NIC and firmware version, an attacker is able to perform arbitrary code execution:

**Initial jump**

- an attacker can overwrite a return address in the stack;
- she can find a stable (for a firmware version) memory address for *username*;
- she can put exploit code in *username* and jump there.

**Stage 1**

- *username* is 255 chars (minus padding), not much instructions;
- but the attacker has access to network buffers;
- she can put code in a previously sent packet and jump there.
Exploit and stage 1

Stack

Heap

0x1c000
0x1bfc4
0x1befc
0x14940

Username

crash analysis

SGDSN/ANSSI – http://www.ssi.gouv.fr/trustnetworkcard
Proof of concept code injection (2/2)

stage 2

- size virtually unlimited;
- sent like a normal packet before the exploit;
- prepended by a magic number so stage 1 can find it.

Now the attacker can:

- run arbitrary code on the RX RISC;
- provide new code using simple packets;
- rewrite the firmware if needed;
- ...

SGDSN/ANSSI – http://www.ssi.gouv.fr/trustnetworkcard
Stage 2

The vulnerability

Crash analysis

Heap

Username

nop

nop

nop

nop

addiu $1,$1, 1 look

for

magic

address

and

jump
to

it

number

magic

number

IP Packet (stage 2)

Scratchpad memory

http://www.ssi.gouv.fr/trustnetworkcard
Man in the middle

Every packet ends up in the card memory:
- received packets before reaching the host;
- sent packets before being emitted on the wire.

Play
- reroute DNS traffic;
- reroute all traffic;
- modify TLS negotiations;
- perform any conceivable MITM stealthily.
Remote management

Remember DELL disabled remote management?

- but the controller is connected to the SMBus;
- ASF! description table is present with the remote control functions;
- the exploit send messages to the SMBus;
- therefore it can perform \textit{power-up, power-down, power-cycle}, ... 

It can be reimplemented!
Take-over the host

The network card:

- is on the PCI/PCI-Express bus;
- can read/write to PCI configuration space;
- has Direct Memory Access (DMA) to the host.

The attacker taking over the NIC can read and write to main host memory!
Using DMA

DMA transfers

- NIC and host share network packets using DMA;
- meta-data (NIC address, host address, size) are stored in special structures, the *buffer descriptors*.

proof of concept code: write to main memory

- write an host address to a buffer descriptor address field in the NIC;
- send packets;
- packet is written to the main memory at given address.

(almost) reliable.

**SGDSN/ANSSI** – [http://www.ssi.gouv.fr/trustnetworkcard](http://www.ssi.gouv.fr/trustnetworkcard)
OS dependent

- like all DMA-based attacks;
- need to get around IOMMU;
- need to find out where to read/write;
- need to trigger the code execution.

- for the proof of concept, we used Linux (because we know how it works);
- same would work for any other OS;
Demonstration

What we do in the demo:

- write some code at address 0 to run a remote shell;
- hook ourselves into `icmp_rcv` to jump at address 0;
- send a *magic ping*. 
Countermeasures

▶ use a patched firmware;
▶ deactivate ASF (not only in the BIOS);
▶ filter ASF and RMCP UDP ports;
▶ use an IOMMU on a supported OS;
▶ deactivate remote administration protocols, or
▶ reserve remote administration to safe/separated networks.
  ▶ nobody ever enabled ASF on a laptop connected to Internet anyway
  ▶ is it really safe to assume that?
Conclusion

This vulnerability might seem scary, however remember:

- few cards support ASF;
- fewer cards enable ASF.

But,

- ASF is quite simple:
  - over UDP,
  - few cryptographic algorithms,
  - limited number of sessions,
  - no interaction with the network;

- AMT, IPMI, and the other remote management protocols are more complex:
  - over TCP,
  - heavy use of *webservices* (XML-RPC, SOAP, ...),
  - interactions with the whole network infrastructure (*Active Directory*, *Kerberos*, ...).
Conclusion (2/2)

- more and more devices require firmwares:
  - network cards,
  - wireless network cards,
  - GSM and UMTS chipsets,
  - RAID controllers;
- with common characteristics:
  - no source code available,
  - close to the hardware,
  - possible access to the outside world (network cards),
  - real-time constraints.

More issues are likely to appear in the future.
It is time to develop simpler network cards and smaller drivers.

ANSSI
http://www.ssi.gouv.fr/trustnetworkcard
Question & answers

FAQ are available at

http://www.ssi.gouv.fr/trustnetworkcard