
hacking in physically addressable memory
a proof of concept

David Rasmus Piegdon

Supervisor: Lexi Pimenidis

Lehrstuhl für Informatik IV, RWTH Aachen

http://www-i4.informatik.rwth-aachen.de

February 21st 2006
## Table of Contents

1. Introduction
2. Accessing memory
3. Virtual address spaces
4. Gathering information
5. Injecting code
6. Prospects, Conclusion

LosTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>  
RWTH Aachen University of Technology  
Hacking in physically addressable memory
# Table of Contents

1. Introduction
2. Accessing memory
3. Virtual address spaces
4. Gathering information
5. Injecting code
6. Prospects, Conclusion

IosTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>  
RWTH Aachen University of Technology  
hacking in physically addressable memory
physical addressable memory

“hacking in physically addressable memory”

- Hacking: using a technique for something it has not been designed for
- Physically addressable memory: direct memory access, “DMA”
hacking

- I will show mostly **attacks**
- So actually I will be **cracking** a systems security
- Exploiting et al is not hacking by definition
- “to hack” is mostly misused by media

IosTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>  
RWTH Aachen University of Technology

hacking in physically addressable memory
• I will show mostly attacks
• So actually I will be cracking a systems security
• Exploiting et al is not hacking by definition
• “to hack” is mostly misused by media
DMA

• DMA = Direct Memory Access
• Basic requirement for introduced approach
• Known for a long time: attacker has DMA -> own3d
  • own3d by an iPod [1]
  • and others [2, 3]
• This is a proof of concept
# Table of Contents

1. Introduction
2. Accessing memory
3. Virtual address spaces
4. Gathering information
5. Injecting code
6. Prospects, Conclusion
Many ways to gain access to memory:

- special PCI cards (forensic, remote management cards)
- special PCMCIA cards
- FireWire (IEEE1394) DMA feature
- anything with DMA
- `/dev/mem` (Linux)
- memory dumps
- Suspend2Disk images
- Virtual machines
- ...

IosTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>

RWTH Aachen University of Technology

hacking in physically addressable memory
Generic problems of DMA attacks

- Swapping
- Multiple accessors at any time
- Caching (?)
DMA hardware

Hardware we may use is

- expensive
- specially crafted
- selfmade (some)
- rare
- not hot-pluggable (depends)
- one exception: FireWire (IEEE1394)
FireWire overview

- FireWire a.k.a. iLink a.k.a. IEEE1394
  - Hot-pluggable
  - Wide-spread (even among laptops)
  - Expansion Bus (like PCI or PCMCIA)
  - Has DMA (if enabled by driver)
  - Guaranteed bandwith feature
  - Used alot for media-crunching
  - Most people are not aware of abuse-factor

losTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>
### FireWire DMA

- DMA only enabled if driver says so
  - Linux, BSD, MacOSX: by default (can be disabled)
  - Windows: only for devices that “deserve” it (more later)
- If DMA -> full access, no restrictions
Windows DMA

Devices that “deserve” DMA on Windows:
SBP2 (storage) devices, like
  • external disks
  • iPod (has a disk)

The iPod can run Linux...
Windows DMA

Devices that “deserve” DMA on Windows: SBP2 (storage) devices, like
- external disks
- iPod (has a disk)
The iPod can run Linux...
How to identify SBP2 devices

- Identify devices and features from their CSR config ROM
- Config ROM contains
  - GUID: 8 byte globally unique ID (like MAC address)
  - Identifier of driver
  - List of supported features
  - List of supported speeds
  - ...
- CSR config ROM can be faked (see [2])
- Copy config ROM from iPod and install it on any system (→1394csrtool)
- Magically Windows permits DMA for any device
How to identify SBP2 devices

- Identify devices and features from their CSR **config ROM**
- Config ROM contains
  - GUID: 8 byte globally unique ID (like MAC address)
  - Identifier of driver
  - List of supported features
  - List of supported speeds
  - ...
- CSR config ROM can be faked (see [2])
  - Copy config ROM from iPod and install it on any system
    (→1394csrtool)
  - Magically Windows permits DMA for any device
DMA hardware

How to identify SBP2 devices

- Identify devices and features from their CSR config ROM
- Config ROM contains
  - GUID: 8 byte globally unique ID (like MAC address)
  - Identifier of driver
  - List of supported features
  - List of supported speeds
  - ...
- CSR config ROM can be faked (see [2])
- Copy config ROM from iPod and install it on any system
  (→1394csrtool)
- Magically Windows permits DMA for any device
How to identify SBP2 devices

- Identify devices and features from their CSR config ROM
- Config ROM contains
  - GUID: 8 byte globally unique ID (like MAC address)
  - Identifier of driver
  - List of supported features
  - List of supported speeds
  - ...
- CSR config ROM can be faked (see [2])
- Copy config ROM from iPod and install it on any system (→1394csrtool)
- Magically Windows permits DMA for any device
Joana Rutkowska will introduce methods to “Cheat Hardware Based RAM Forensics” on Black Hat DC in March (see http://theinvisiblethings.blogspot.com/2007/01/beyond-cpu-cheating-hardware-based-ram.html)
DMAX software

/dev/mem

- Gives access to physically addressed memory (in opposite to /dev/kmem)
- Often needed by X-server
- Shall be obsoleted in future (X shall use DRI)
- Only gives access to lower 896MB RAM (only these are mapped)
One interface to access them all

- One generic interface: libphysical
- Backends for anything...
- Implemented so far:
  - Filedescriptor (/dev/mem, memory dumps)
  - FireWire
Table of Contents

1 Introduction

2 Accessing memory

3 Virtual address spaces

4 Gathering information

5 Injecting code

6 Prospects, Conclusion
so what now?

- Once we got access...we can see a bunch of random memory
- How does OS manage memory?
Could parse kernel data-structures (if found). But they are different for different

- hardware architecture
- operating system
- OS version
- and may not be documented (Windows)

Or we could do something else…
Could parse kernel data-structures (if found). But they are different for different

- hardware architecture
- operating system
- OS version
- and may not be documented (Windows)

Or we could do something else…
Virtual Address Spaces

- Multitasking Operating System
- System runs several processes “at once”
- Privilege separation required (see [5])
- Normally done in hardware

→ Each process has own virtual address space
→ Cannot access other processes memory or operating systems memory
→ Cannot circumvent protection mechanism
Virtual Address Spaces

- Multitasking Operating System
- System runs **several processes** “at once”
- Privilege separation required (see [5])
- Normally done in **hardware**

→ Each process has own virtual address space
→ Cannot access other processes memory or operating systems memory
→ Cannot circumvent protection mechanism
IA-32 Linux VM Layout

unknown

lowmem
mapped kernel space

heap

mapped executable

stack, growing downward
mapped libraries, growing downward

losTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>

hacking in physically addressable memory

RWTH Aachen University of Technology
IA-32 provides two techniques (that may be chained)

- **Segmentation** (required)
- **Paging** (optional)

Linux only uses paging, all segments span full 4GB of virtual memory
IA-32 virtual ("logical") address translation

(from [6])
Done in hardware

- Translation done in hardware (by CPU)
- Hardware needs to know how to do it:
  - Global Descriptor Table (GDT)
  - Local Descriptor Table (LDT)
  - Page Directory (PD), Page Tables (PT)
Once we got these structures, we know which page belongs where in which address space

- Linux: GDT, LDT are irrelevant (flat segments)
- only PD is required
- PD references PTs
  - PD may have recognisable patterns (has for Linux and Windows)
- one PD per process
Once we got these structures, we know which page belongs where in which address space

- Linux: GDT, LDT are irrelevant (flat segments)
- only PD is required
- PD references PTs
- PD may have recognisable patterns (has for Linux and Windows)
- one PD per process
Finding ATTs

Address Translation Tables (including PDs)...

- depend on architecture
- depend on operating system
- may have recognisable patterns

→ create signature for (arch, OS). so far:
  - (i386, Linux 2.4 and 2.6)
  - (i386, Windows XP)
Finding ATTs

Address Translation Tables (including PDs)...

- depend on architecture
- depend on operating system
- may have recognisable patterns

→ create signature for (arch, OS). so far:
  - (i386, Linux 2.4 and 2.6)
  - (i386, Windows XP)
Finding ATTs, details

1. Sieve all pages by simple, static pattern (e.g. 4 bytes)
2. For each possible do statistical analysis:
   - Normalized Compression Distance (NCD) to known true ATT
3. If possibility high enough, test integrity of data
   (for IA-32: try to load referenced PTs)
4. If ok, its (most probably) an ATT
Normalized Compression Distance

- **Normalized Information Distance:**
  - Minimal amount of changes required between two information
  - Uses *Kolmogorov Complexity* (KC) (size of minimal representation of information)
  - Incalculable

- KC can be approximated by compressor
  → **Normalized Compression Distance:**
  - Calculable
  - Very versatile
  - e.g. create relational trees of gene-sequences [4]
Normalized Compression Distance

- **Normalized Information Distance:**
  - Minimal amount of changes required between two information
  - Uses Kolmogorov Complexity (KC) (size of minimal representation of information)
  - Incalculable

- **KC can be approximated by compressor** → **Normalized Compression Distance:**
  - Calculable
  - Very versatile
  - e.g. create relational trees of gene-sequences [4]
Finding Address Translation Tables

Once a PD is found, we can do the translation by hand:

- Well-defined algorithm for architecture, e.g. for IA-32: [6]
- Implementation in software in liblinear. So far:
  - IA-32 Protected Mode, without PAE36
    (Linux with $\leq$ 4GB RAM)
Table of Contents

1. Introduction
2. Accessing memory
3. Virtual address spaces
4. Gathering information
5. Injecting code
6. Prospects, Conclusion
So far

- We can access physical memory sources in a generic way (`libphysical`)
- We can find and access virtual address spaces of processes (`liblinear`)

Now we want to identify processes we found.
Identifying Processes

```c
#include <stdio.h>

int main(int argc, char**argv)
{
    printf("my name is %s\n", argv[0]);
    return 0;
}
```
Identifying Processes

- `argv`, `envv` are somewhere in the address space
- They are on the stack, on first mapped pages below page `0xc0000`
- NUL-separated vector with
  - Path of binary
  - Environment
  - Arguments
Identifying Processes

- `argv`, `envv` are somewhere in the address space
- They are on the stack, on first mapped pages below page `0xc0000`
- NUL-separated vector with
  - Path of binary
  - Environment
  - Arguments
Identifying Processes

```bash
# OLDPWD=/home/lostrace PWD=/home/lostrace/documents/rwth/SEAT /
/attacks/userspace SHLVL=1 _=./victim 
./victim --arg=foo bar --baz
```

```
0xbfc5ff70 00 00 00 00 2e 2f 76 69 |...../vi| ARGV[]:
0xbfc5ff78 63 74 69 6d 00 2d 2d 61 |ctim.--a| [0] = bfc5ff74
0xbfc5ff80 72 67 3d 66 6f 62 |rg=foo.| [1] = bfc5ff7d
0xbfc5ff88 61 72 00 2d 2d 61 7a |ar.--baz| [2] = bfc5ff87
0xbfc5ff90 00 4f 4c 44 50 57 44 3d |OLDPWD=| [3] = bfc5ff8b
0xbfc5ff98 2f 68 6f 6d 65 2f 68 6f |./home/lo| [4] = NULL
0xbfc5ff90 63 74 6d 6d 65 00 50 61 |strace.|[P]
0xbfc5ff98 57 4d 3d 2f 68 6f 6d 65 |WD=/home/|
0xbfc5ff90 2f 6c 6f 73 72 61 63 65 |lostrac|.
0xbfc5ff98 65 2f 64 6f 63 75 6d 65 |e/docume|
0xbfc5ff90 6e 74 73 2f 72 77 74 68 |nts/rwth|
0xbfc5ff98 53 45 41 54 2f 61 74 74 |/SEAT/at|
0xbfc5ff90 74 61 63 6b 73 2f 75 73 |tacks/us|
0xbfc5ff98 74 61 63 6b 73 2f 75 73 |erspace.|.
0xbfc5ff90 53 48 4c 56 4d 31 00 SHLVL=1.|.
0xbfc5ff98 5f 3d 2e 2f 76 69 63 74 |._=./vict|
0xbfc5ff90 69 6d 00 2e 2f 76 69 63 2f 68 6f 6d 65 00 |im../vic|
0xbfc5ff98 74 69 6d 00 00 00 00 00 00 00 00 00 00 00 |tim.....|
```
Stack arguments

- Find page, parse structure back-to-front:
- Last 5 bytes are always NUL
- Previous string is always binary
- Problem: difference between argument and environment?
- Solution: find `argv[0]` on stack and use userspaces `argv[]`
Finding Specific Processes

1. Find all virtual address spaces
2. For each: look if binary matches searched binary, e.g.:
   - /usr/lib/mozilla-firefox/firefox-bin
   - /usr/bin/gpg
   - /usr/bin/psi
   - /usr/bin/openssl
   - /usr/bin/ssh-agent
3. If matches, steal a cookie or... a ssh-private key
Finding Specific Processes

1. Find all virtual address spaces
2. For each: look if binary matches searched binary, e.g.:
   - /usr/lib/mozilla-firefox/firefox-bin
   - /usr/bin/gpg
   - /usr/bin/psi
   - /usr/bin/openssl
   - /usr/bin/ssh-agent
3. If matches, steal a cookie or... a ssh-private key
Secrets

Stealing SSH private keys

Let’s get dangerous!
Steal SSH private key from `ssh-agent`:

- agent keeps key decrypted, locked in memory
- has timeout-function to wipe keys from memory
- stalled in `read()`-syscall on socket
- no timer-signal to check for timeout
- checks timer only on query

IosTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>
RWTH Aachen University of Technology

hacking in physically addressable memory
Stealing SSH private keys

Let’s get dangerous!
Steal SSH private key from `ssh-agent`:

- agent keeps key decrypted, locked in memory
- has timeout-function to wipe keys from memory
- stalled in `read()`-syscall on socket
- no timer-signal to check for timeout
- checks timer only on query
finding SSH Private keys

- Where (in filesystem) do you keep your keys?
- $HOME/.ssh/*
- comment := path of key

```
[foo@bar:~] > ssh-add -l
1024 00:11:...:ee:ff /home/foo/.ssh/id_rsa (RSA)
```
typedef struct identity {
    Key *key;
    char *comment;
    u_int death;
} Identity;

struct Key {
    int type;
    int flags;
    RSA *rsa;
    DSA *dsa;
};
finding SSH Private keys [2]

1. Find `comment-string` in heap
2. Find PTR to `comment (struct identity)` in heap
3. Follow `key`
4. Follow `key->RSA and key->DSA`
5. A lot of `BIGNUMs` (OpenSSL arbitrary precision integer implementation). *Copy relevant, test integrity* (see [7,8]).
6. **0wn3d**

(yes, there are better methods to find the keys, but this is just a proof of concept)
Resume

- So far: only read memory.
- Works with memory dumps
- No time to prepare an attack?
- → Just dump memory and do it later
Table of Contents

1 Introduction
2 Accessing memory
3 Virtual address spaces
4 Gathering information
5 Injecting code
6 Prospects, Conclusion

IosTrace A.K.A. David R. Piegdon <david.rasmus.piegdon@rwth-aachen.de>

hacking in physically addressable memory
Attacking by Writing

- No more sword to be feared than the learned pen.
- Even the virtual one.
Injecting the code

Inject where?

- Cannot allocate extra memory
- Cannot overflow a buffer (no IO with process)
- Need to overwrite code, data or stack
- Data: where IS data? is data mapped into multiple processes?
Inject into code

- Shared objects, binaries: mapped into multiple processes
- → Affect multiple processes at same time
- Needs to be PIC\(^1\) (mapped at different locations)
- Is there room to inject code?

\(^1\)Position Independent Code
Inject into stack

- Stack is easy to find
- Affect one process at a time (one stack per thread)
- Inject into zero-padded pages containing ENV and ARG.
- Possibly overwrite these (if little space):
  - ENV, ARG are rarely parsed
  - typically only during init
- If overwrites ENV, ARG: possibly visible via
  - /proc/$PID/env
  - /proc/$PID/cmdline
Executing injected code

Use program-flow:

- Typical process calls subroutines
- Stackframes on stack, including return-address

→ Overwrite return-addresses
Protection Mechanisms

- Stackoverflow protection checksums
  - Can manipulate checksum as well
- Page-level no-execute enforcements (Intels EXB, AMDs NX)
  - Manipulate Page Directory to allow execution of stack
Rootshell?

- Royal league of code-injection: interactive (root-)shell

→ Inject bindshell
- Network connection required
- Can be found simply:
  - `lssof -i -n`
  - Network sniffer
  - IDS, NIDS
Rootshell!

→ Inject Shellcode doing IEEE1394-stuff
  • Big, complex payload (IEEE1394 handling)
  • Attack via IEEE1394?
→ Inject Syscall-Proxy
  • Victim, self need to be same architecture, OS, syscall interface
  • I attacked IA-32 from PPC…
DMA-Shell

- Only thing that is for sure: DMA
  → Communication via DMA
Special “Beachhead” Shellcode:

ATTACKERS HOST

| DMA |

Attacker (master)

| DMA |

Ringbuffer FromMaster

| Reader Thread |

VICTIMS HOST

| Pipe ToShell |

STDIN

| STDOUT, STDERR |

Shell

| Pipe FromShell |

Writer Thread

| Ringbuffer ToMaster |
• Payload small (536 Bytes, yet big for shellcode)
• Independent of attackers arch, OS
• Only DMA required
Prospects

- Kernelspace Modifications:
  - Shellcode that injects LKM?
  - `/dev/kmem` already emulated by `liblinear`
  - Live kernel patching?
- Bootstrapping custom operating systems
Conclusion

- DMA attacks are mature
- Access to memory → 0wn3d!
- Keep your firewire-ports secured
- Some of the tools (*libphysical*, *liblinear*) can also be used for forensics
Questions?

Thank you for your attention!

All tools will be released at http://david.piegdon.de/products.html
Thanks...

- Maximillian Dornseif, Christian N. Klein and Michael Becher (basic idea)
- Lexi Pimenidis (supervisor)
- Timo Boettcher and Alexander Neumann (help)
- Swantje Staar (help with english)
- Chaos Computer Club Cologne (in general)

Thank you!
References (FireWire, DMA Attacks)

References


6 Intel Corp. Intel 64 and IA-32 Architectures Software Developer’s Manual.
References
