

GPU-Assisted Malware

Giorgos Vasiliadis

Michalis Polychronakis

Sotiris Ioannidis

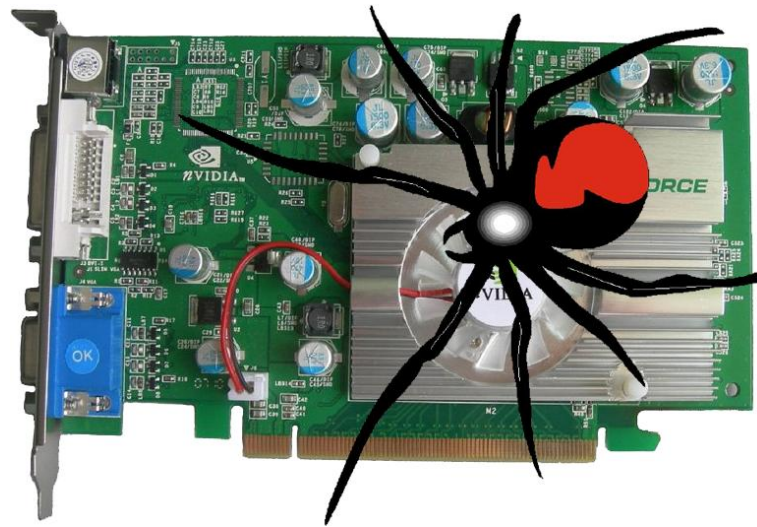
ICS-FORTH, Greece

Columbia University, USA

ICS-FORTH, Greece

Main idea

- Demonstrate how **malware** can increase its **robustness** against detection by taking advantage of the ubiquitous **Graphics Processing Unit (GPU)**



Outline

- Introduction
- Proof-of-concept implementation
- Future attacks
- Conclusions

INTRODUCTION

MALWARE 2010
KNOW YOUR ENEMY

Research • Practical Solutions (Industry Track) • The Law

19 October 2010

IEEE
 **computer
society**

Malware evasion

- Malware continually seek new methods for **hiding** their malicious activity, ...
 - Packing
 - Polymorphism
- ... as well as, **hinder** reverse engineering and code analysis
 - Code obfuscation
 - Anti-debugging tricks

Countermeasures

- Signature-based antivirus scanners
 - Disk and memory scanning
- Static/dynamic code analysis
 - PolyUnpack [15], OmniUnpack[11]
- System emulation
 - Renovo[9]

➔ ***Unfortunately***, malware defense and analysis mechanisms focus on **IA-32 code**

GPU = Graphics Processing Unit

- Dominant co-processor in home personal computers
- The heart of graphics cards
- Traditionally, used for handling 3D graphics rendering
- Over the years, GPUs have been constantly evolving



General-purpose GPU

- Powerful computation unit
 - General-purpose code execution
- Specialized APIs exposing several hardware features
 - Fully cooperate with CPU
 - DMA over main memory
- Portable code
 - No need to install any files
 - No administrator privileges

Malware + GPU = ?

- Is it possible for a malware to **exploit** the rich functionality of **modern GPUs**?
 - GPU Powered Malware [Reynaud'08]

This work

- Design and implementation of **code armoring techniques** based on **GPU code**
 - Self-unpacking
 - Run-time polymorphism



SELF-UNPACKING

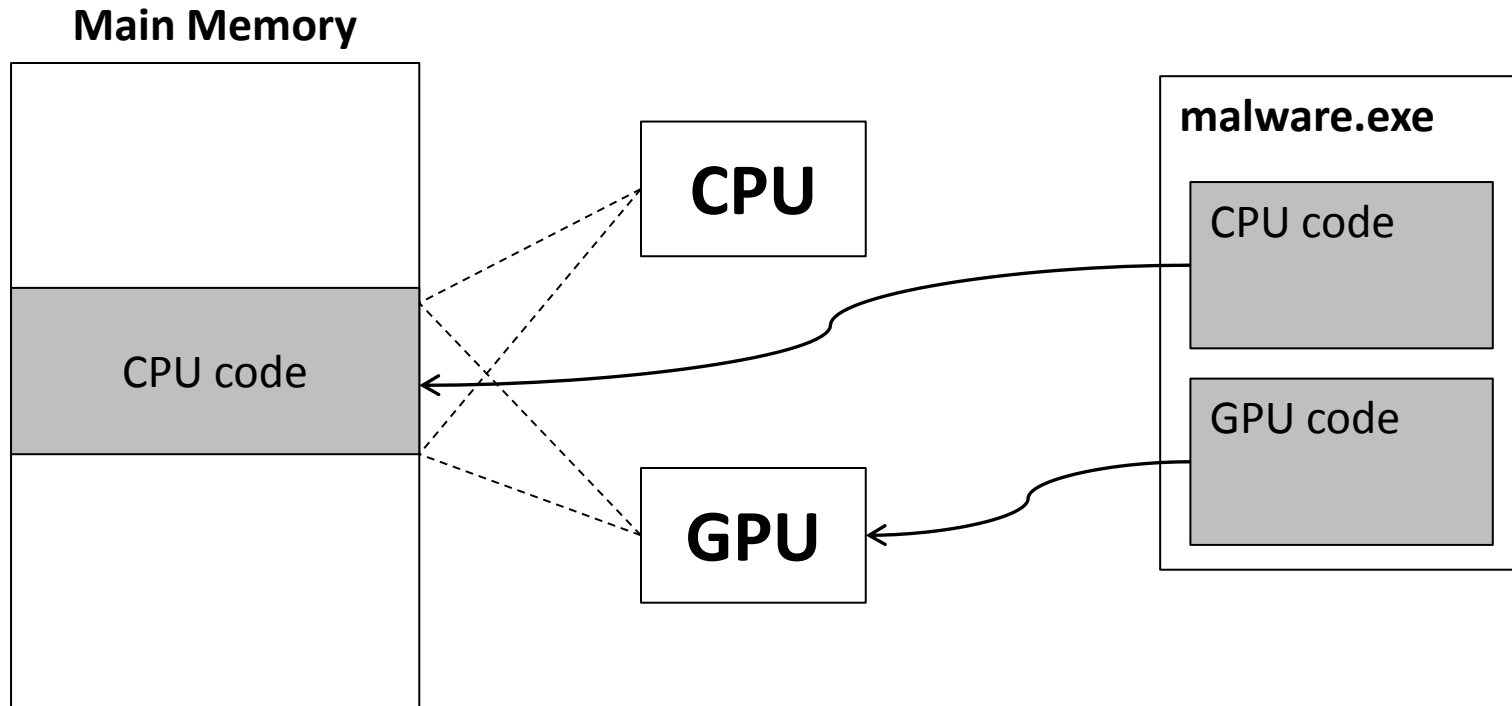
MALWARE 2010
KNOW YOUR ENEMY

Research • Practical Solutions (Industry Track) • The Law

19 October 2010

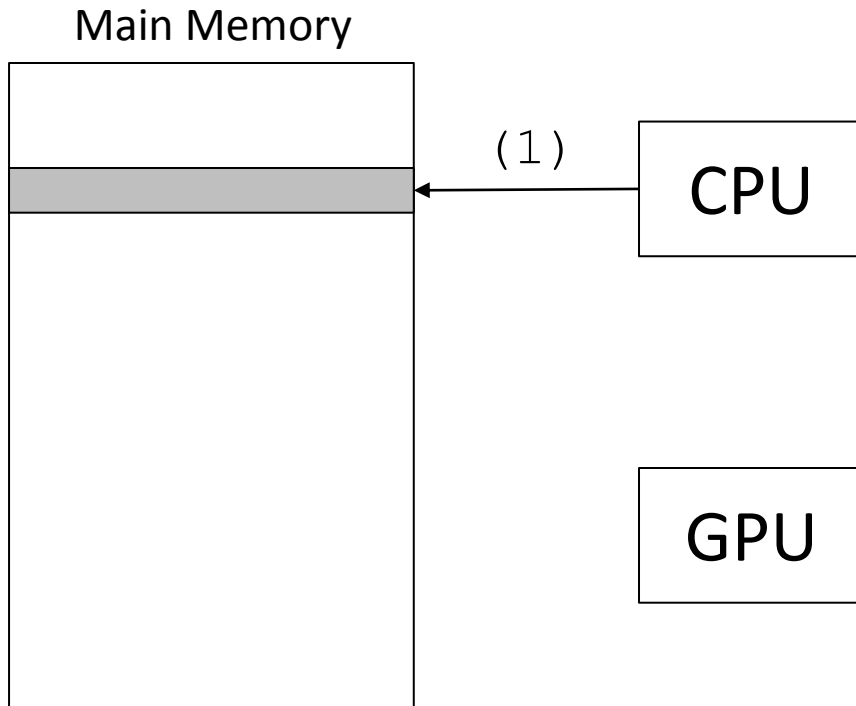
IEEE
 computer
society

Basic design



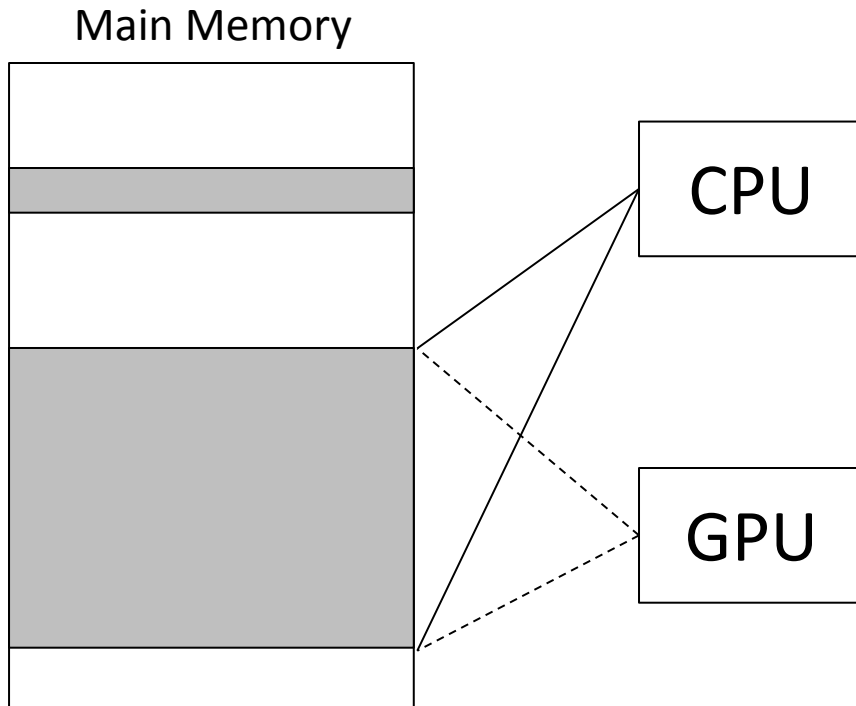
Execution example (1/7)

1. `init / bootstrap`



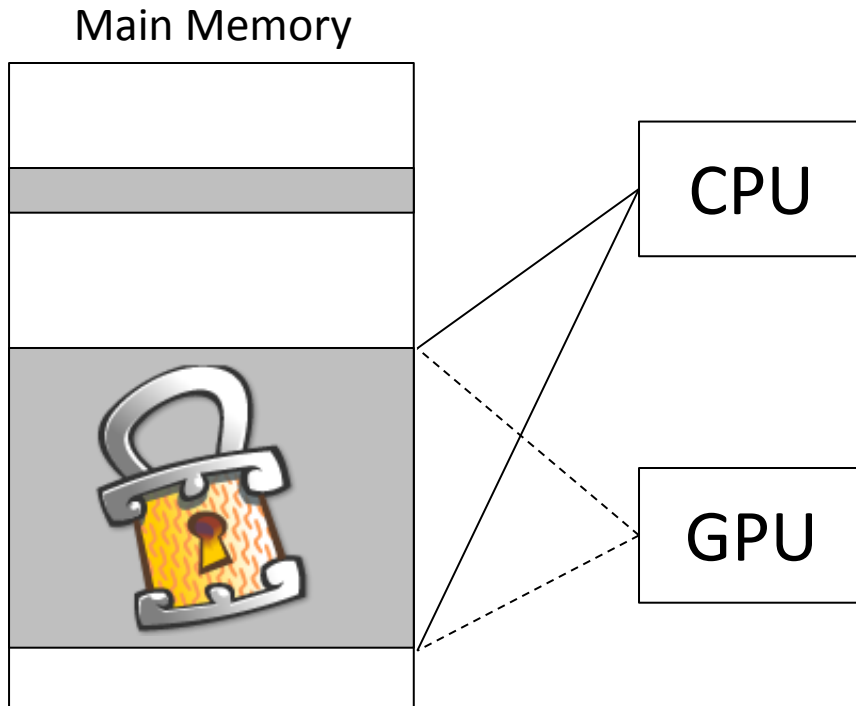
Execution example (2/7)

1. `init / bootstrap`
2. `alloc mmap'ed buffer`



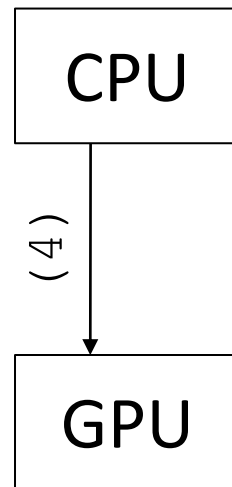
Execution example (3/7)

1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. **store packed code**



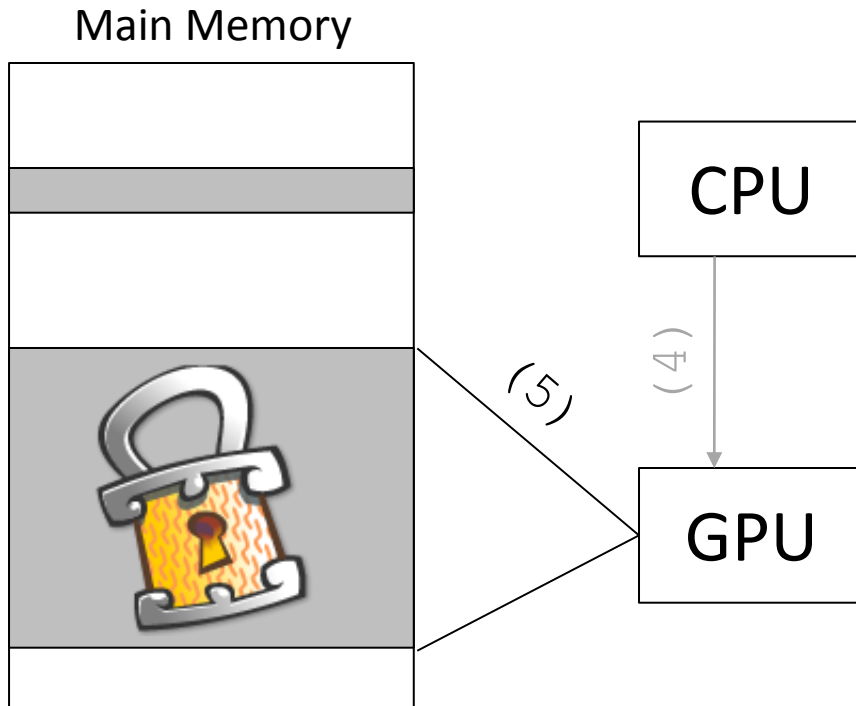
Execution example (4/7)

Main Memory



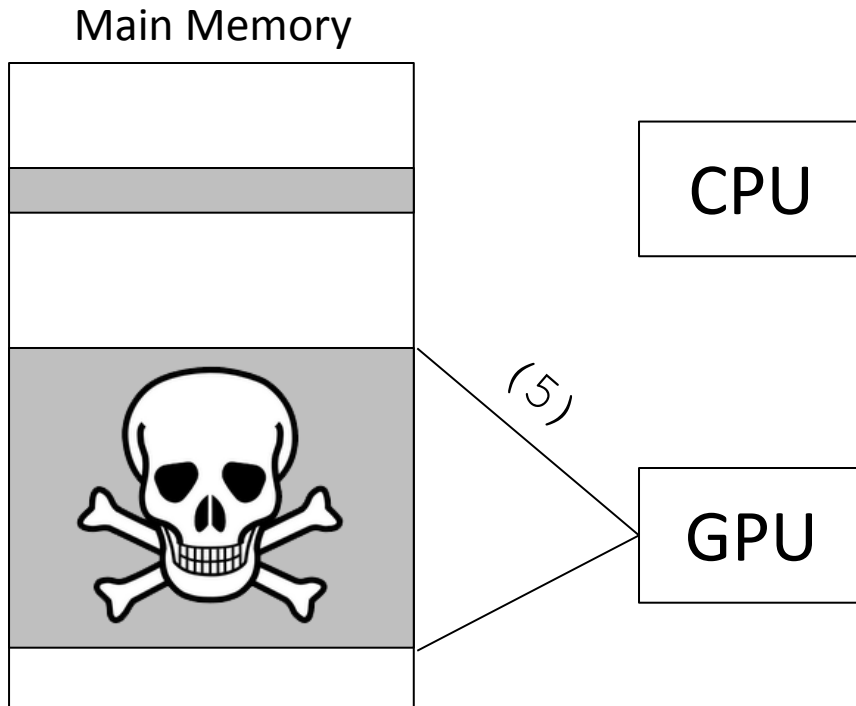
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. **invoke GPU**

Execution example (5/7)



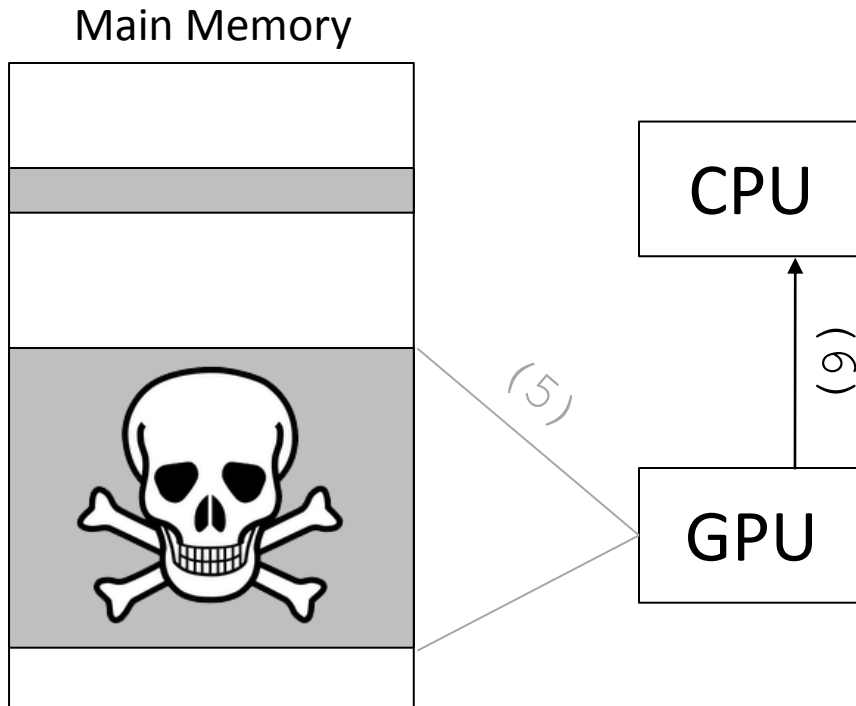
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. invoke GPU
5. **unpack malicious code**

Execution example (5/7)



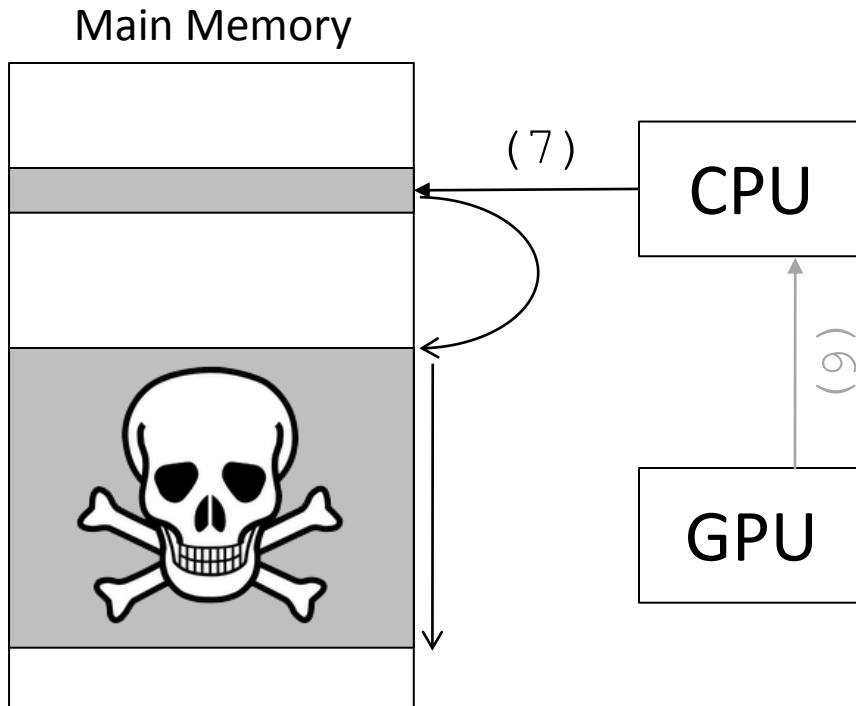
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `invoke GPU`
5. **`unpack malicious code`**

Execution example (6/7)



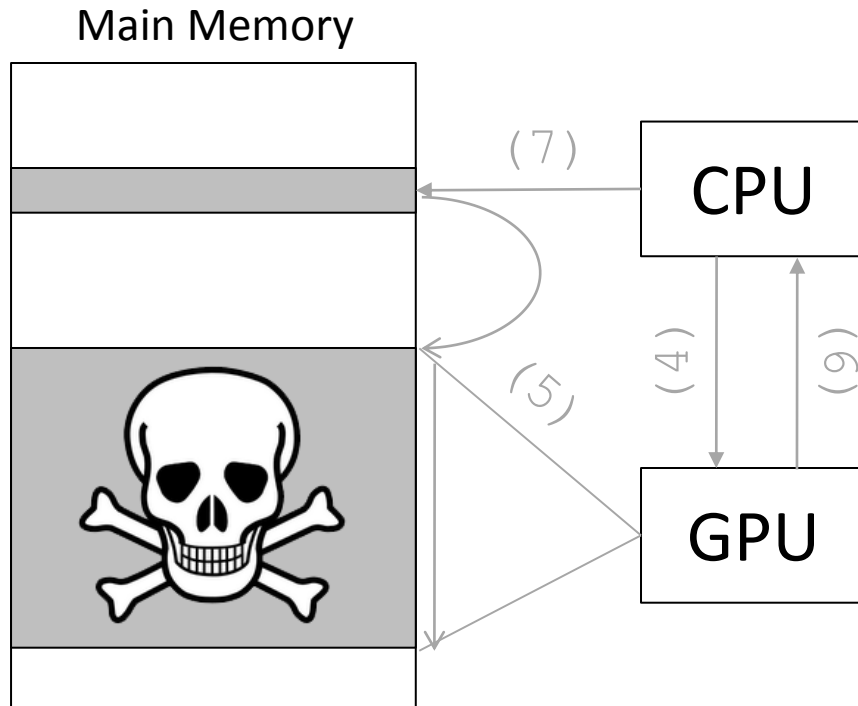
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `invoke GPU`
5. `unpack malicious code`
6. **GPU return**

Execution example (7/7)



1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. invoke GPU
5. unpack malicious code
6. GPU return
7. **exec malicious code**

Big Picture



1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. invoke GPU
5. unpack malicious code
6. GPU return
7. exec malicious code

Self-unpacking: Strengths

- Exposes minimal x86 code footprint
- Current analysis and unpacking systems cannot handle GPU code
- Cannot run on virtual-machines
- GPU can use extremely complex encryption schemes

Self-unpacking: Weaknesses

- Malware code lies unencrypted in main memory after unpacking
- Can be detected by dumping the memory
- Can we do better?

RUN-TIME POLYMORPHISM

MALWARE 2010
KNOW YOUR ENEMY

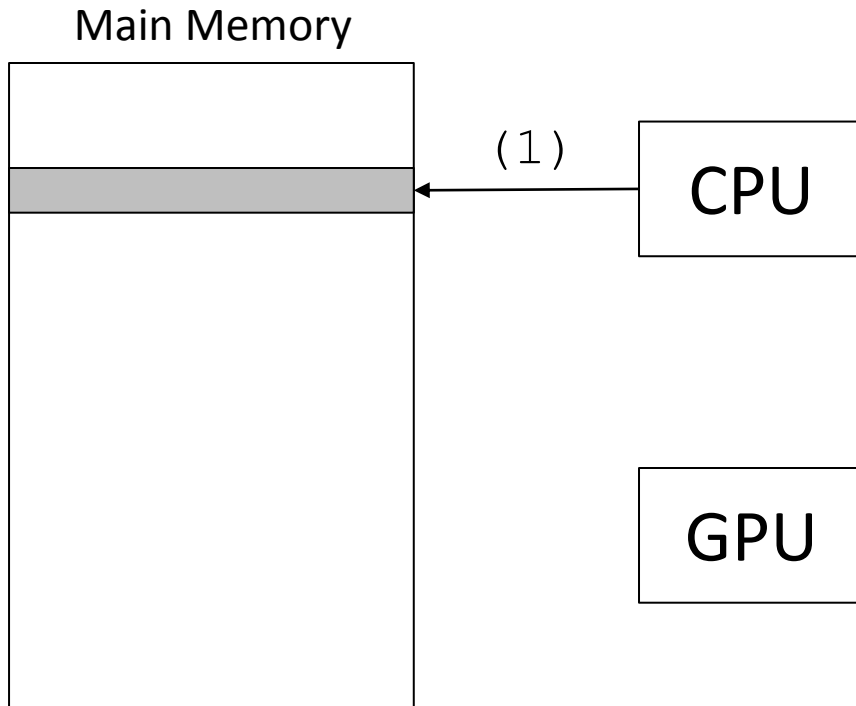
Research • Practical Solutions (Industry Track) • The Law

19 October 2010

IEEE
 computer
society

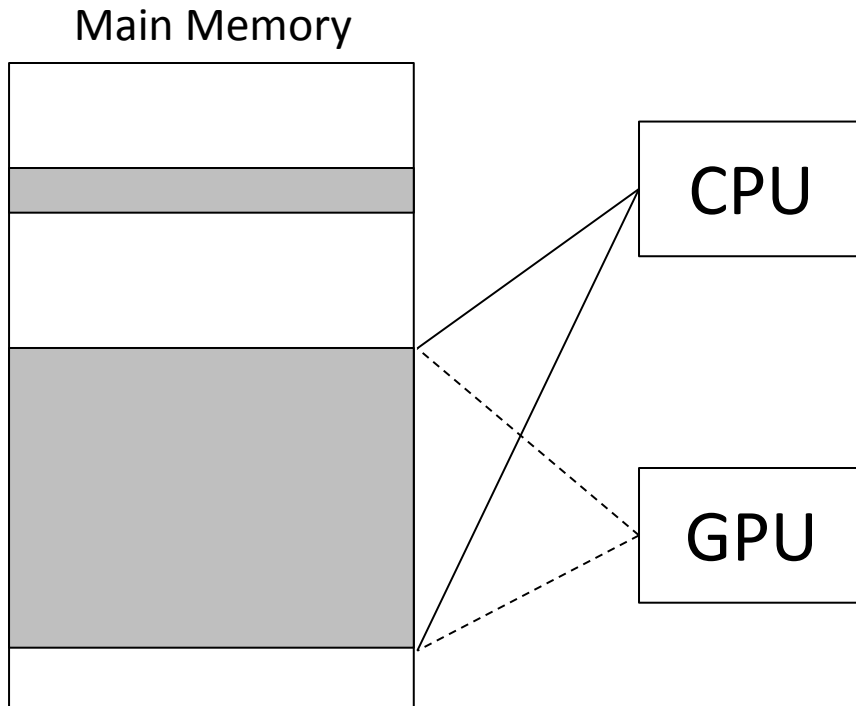
Execution example (1/19)

1. `init / bootstrap`



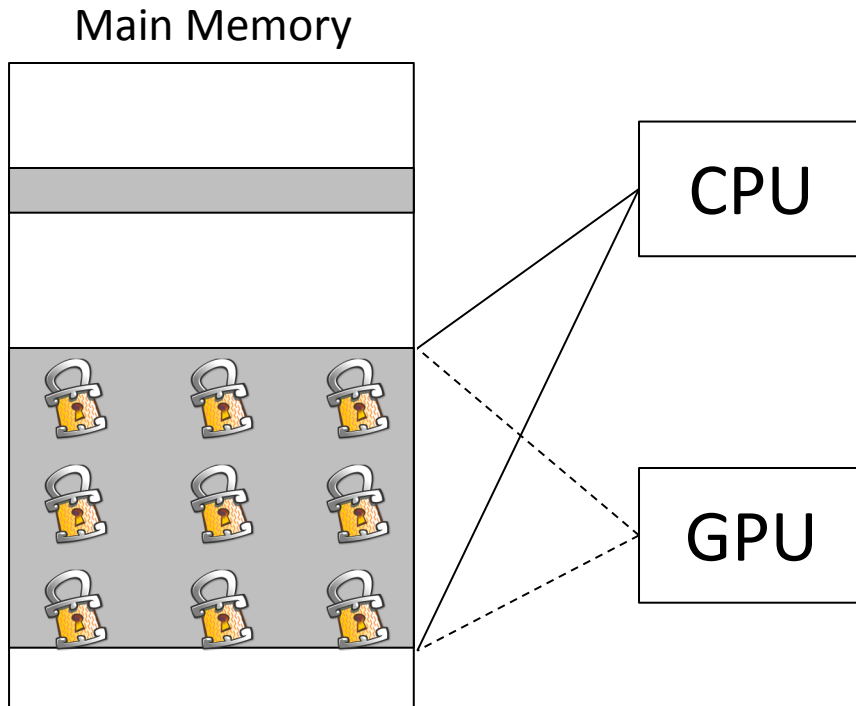
Execution example (2/19)

1. `init / bootstrap`
2. `alloc mmap'ed buffer`

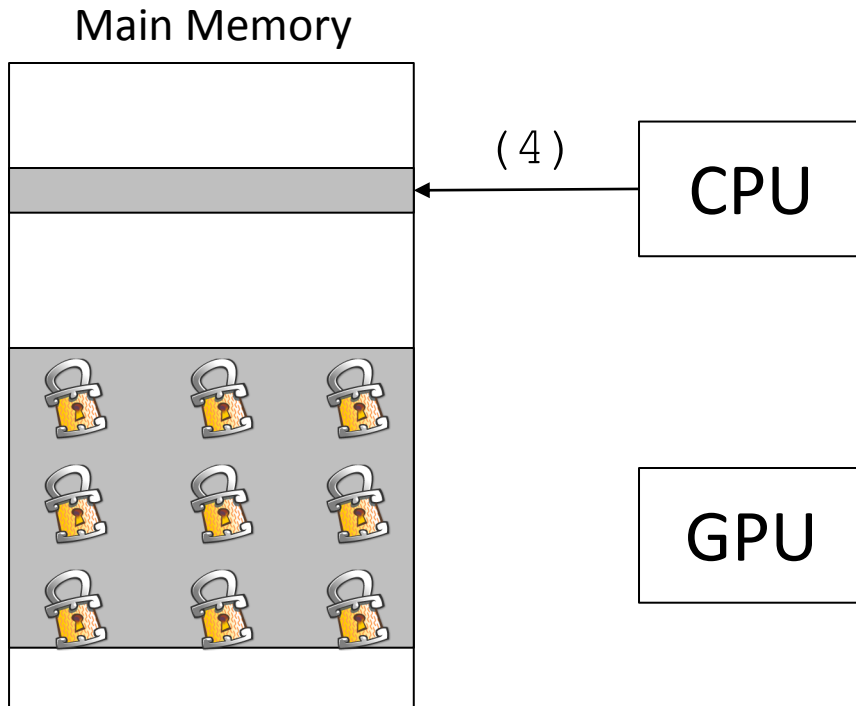


Execution example (3/19)

1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. **store packed code**

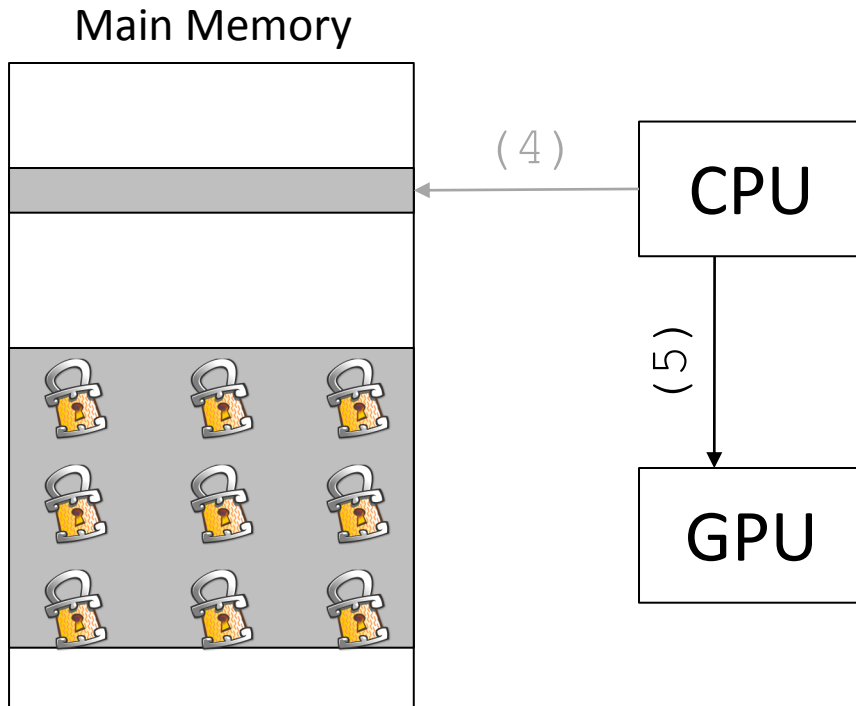


Execution example (4/19)



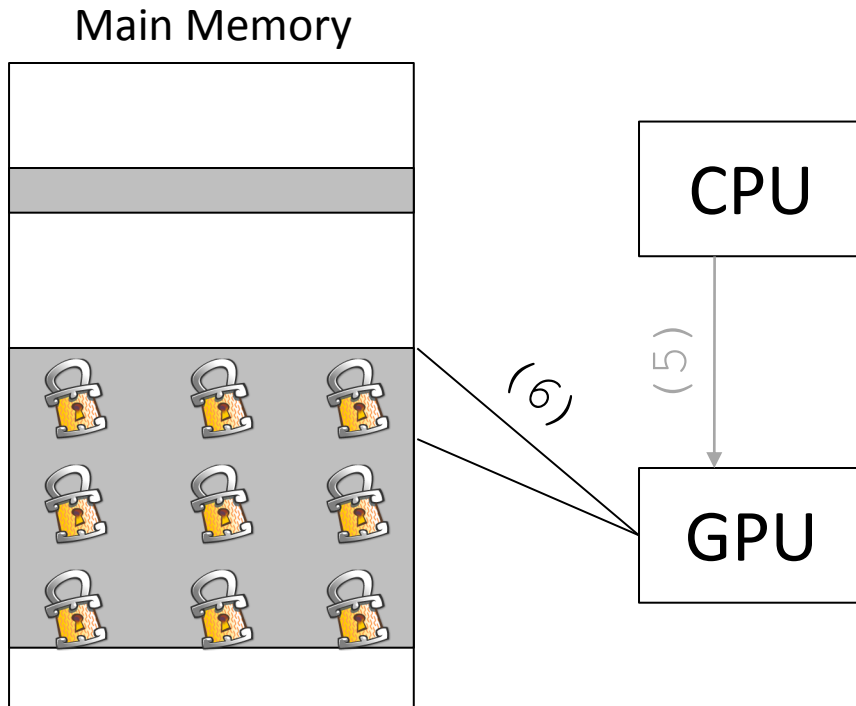
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. **control**

Execution example (5/19)



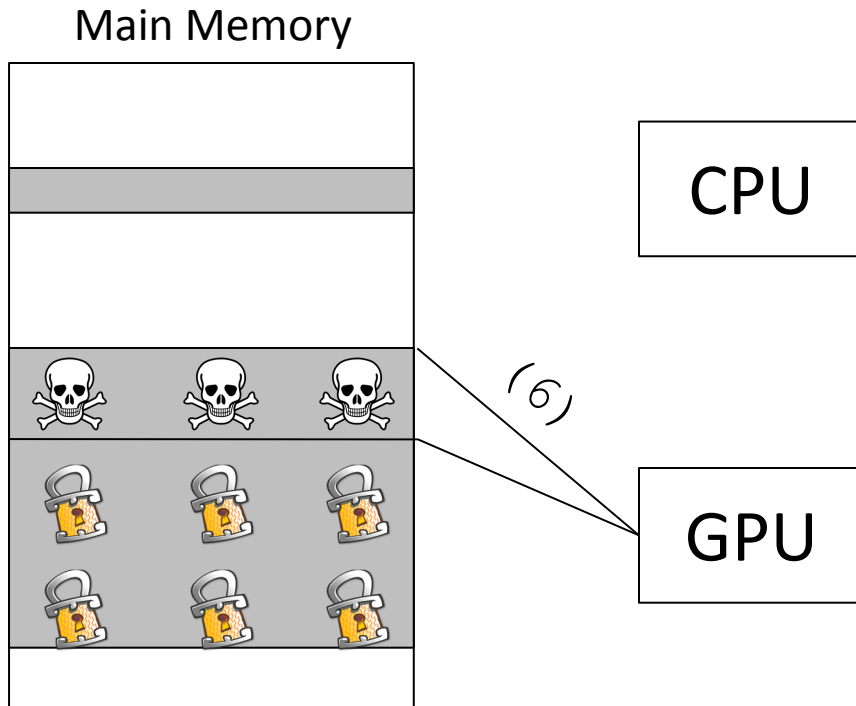
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `control`
5. **`invoke GPU`**

Execution example (6/19)



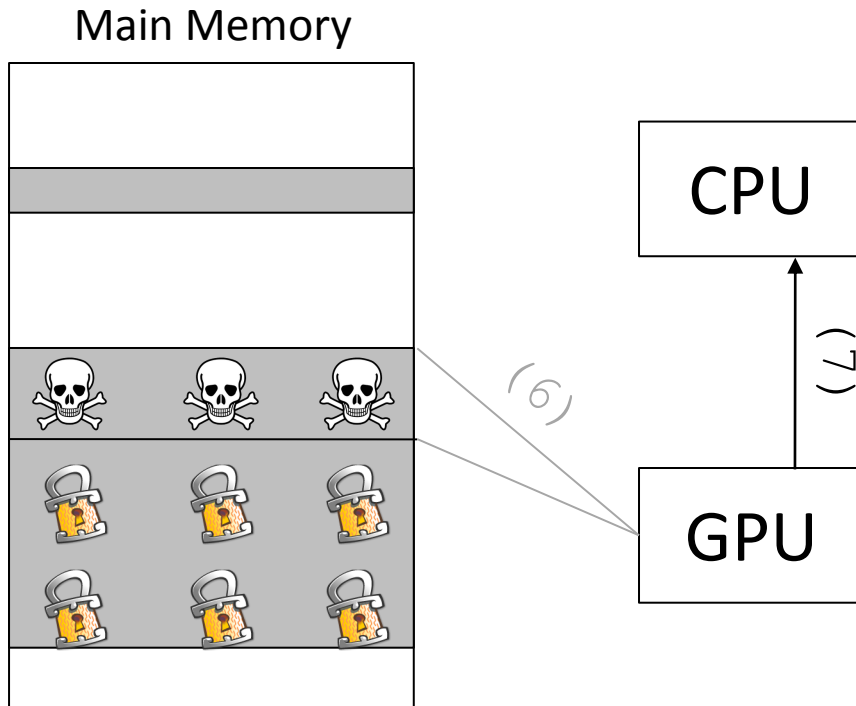
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `control`
5. `invoke GPU`
6. **unpack code segment**

Execution example (6/19)



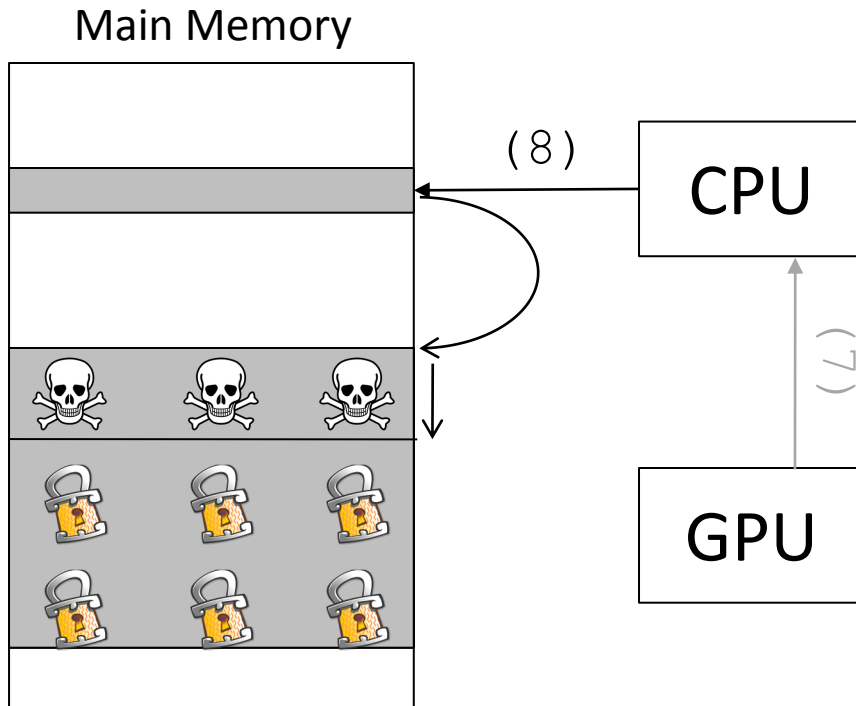
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `control`
5. `invoke GPU`
6. **`unpack code segment`**

Execution example (7/19)



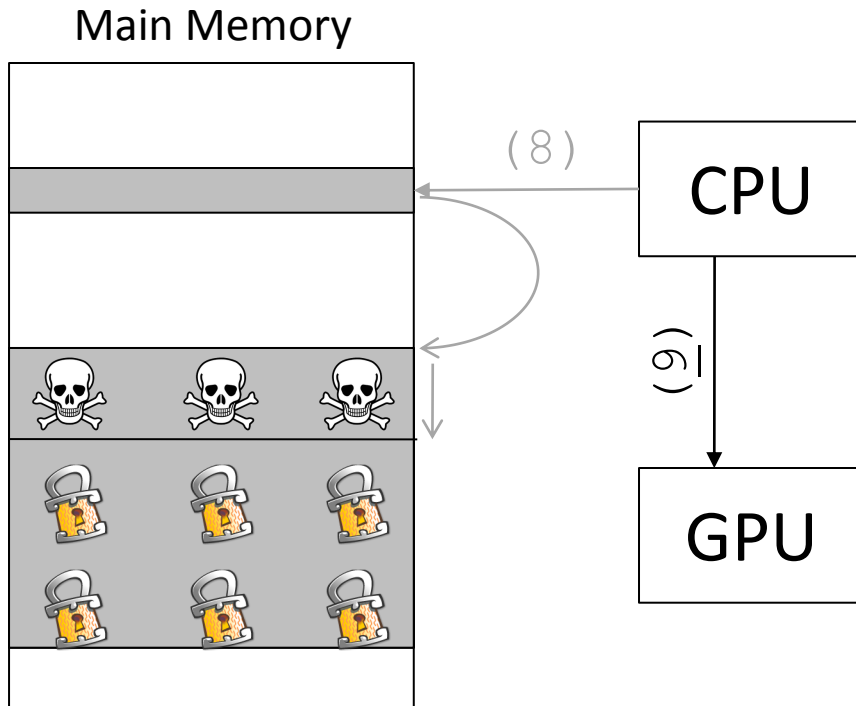
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `control`
5. `invoke GPU`
6. `unpack code segment`
7. **GPU return**

Execution example (8/19)



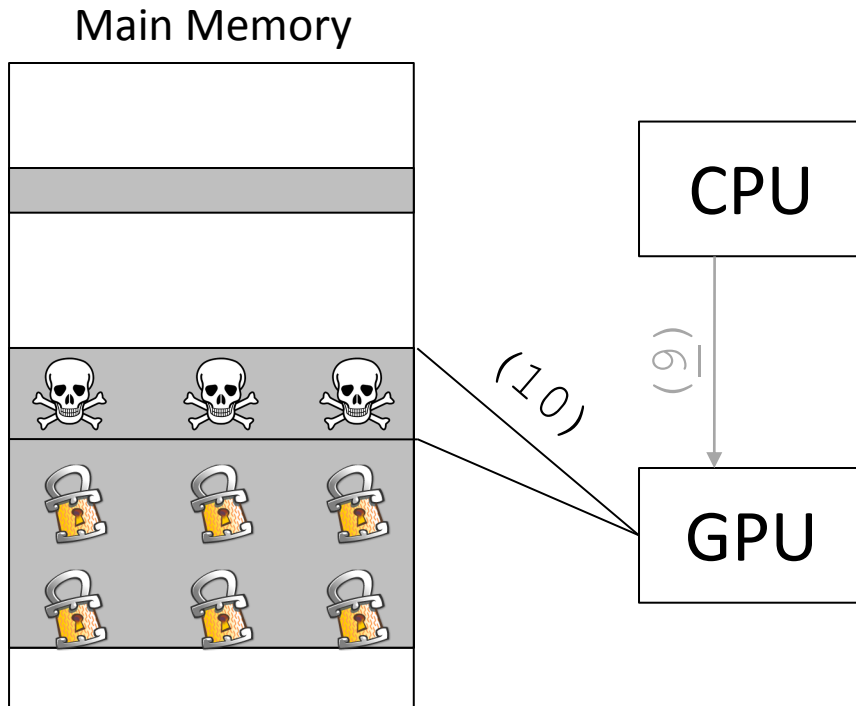
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `control`
5. `invoke GPU`
6. `unpack code segment`
7. `GPU return`
8. **`exec malicious code`**

Execution example (9/19)



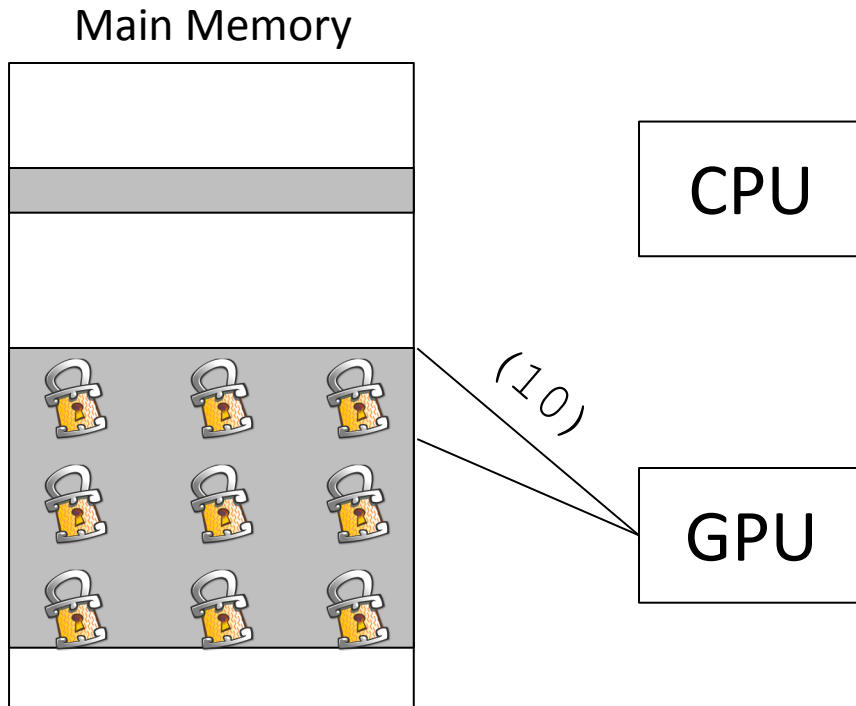
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. **invoke GPU**

Execution example (10/19)



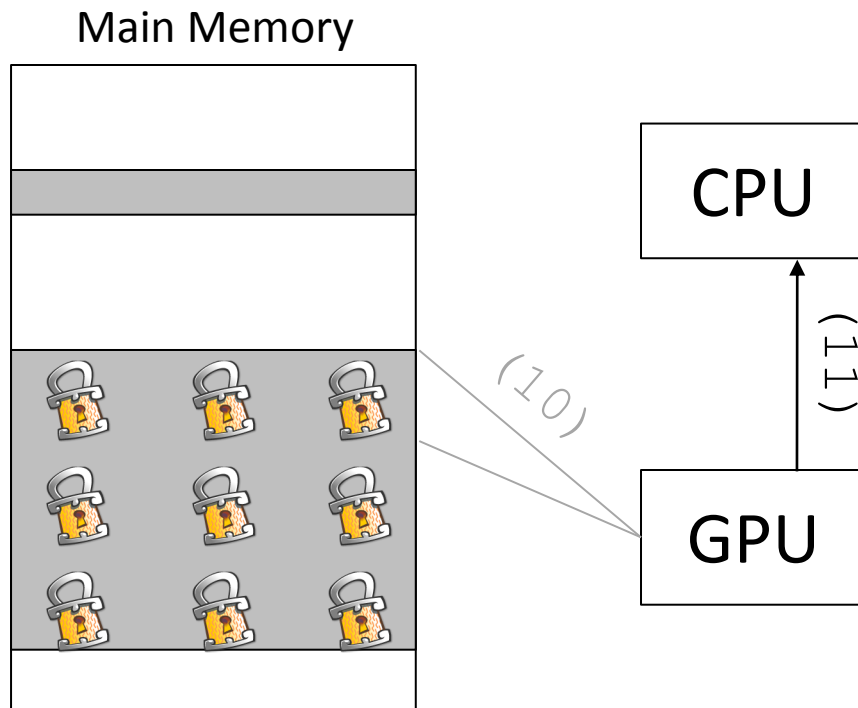
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. **pack code**

Execution example (10/19)



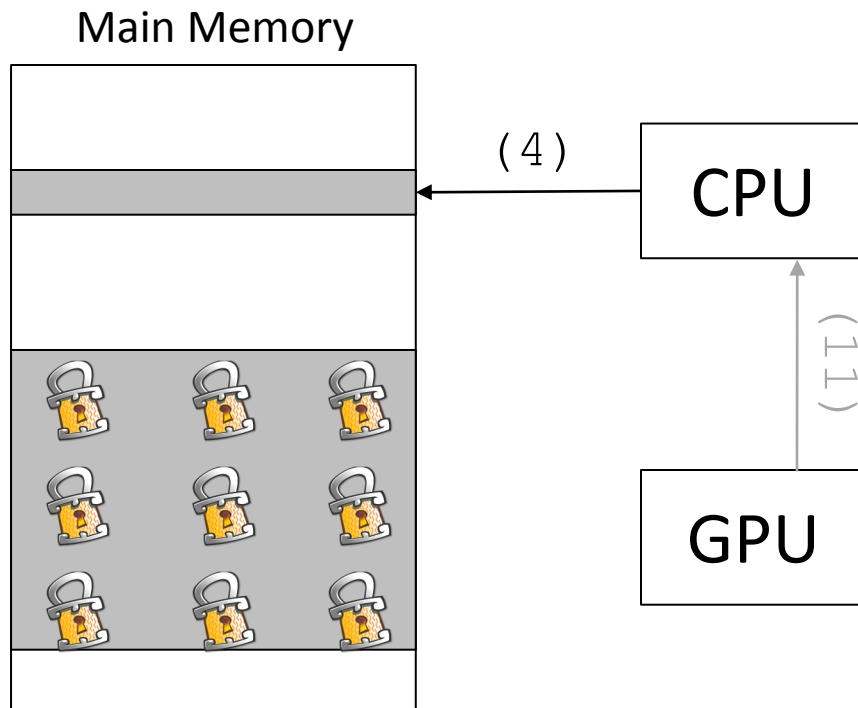
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. **pack code**

Execution example (11/19)



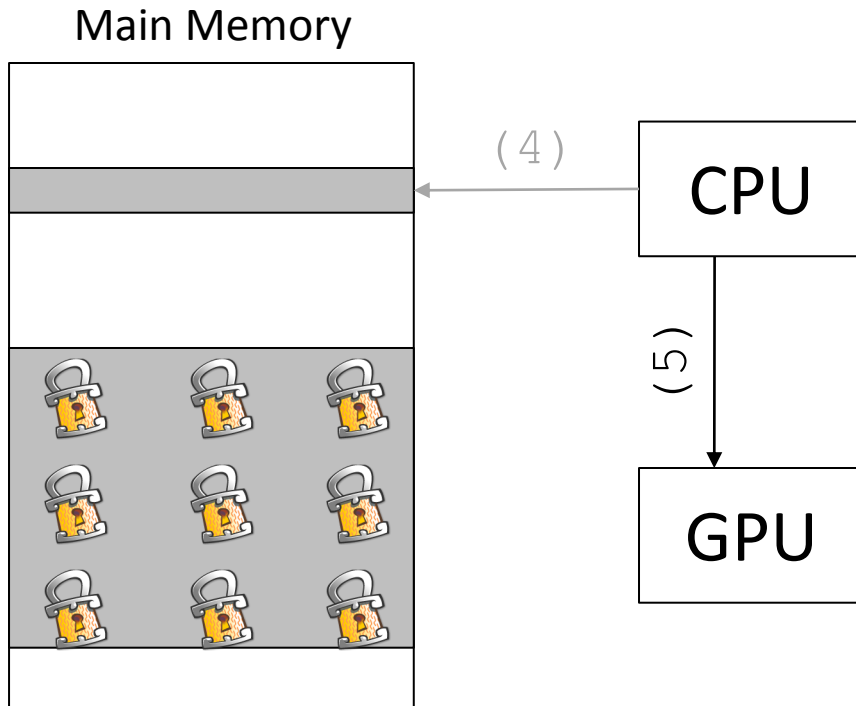
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
- 11. GPU return**

Execution example (12/19)



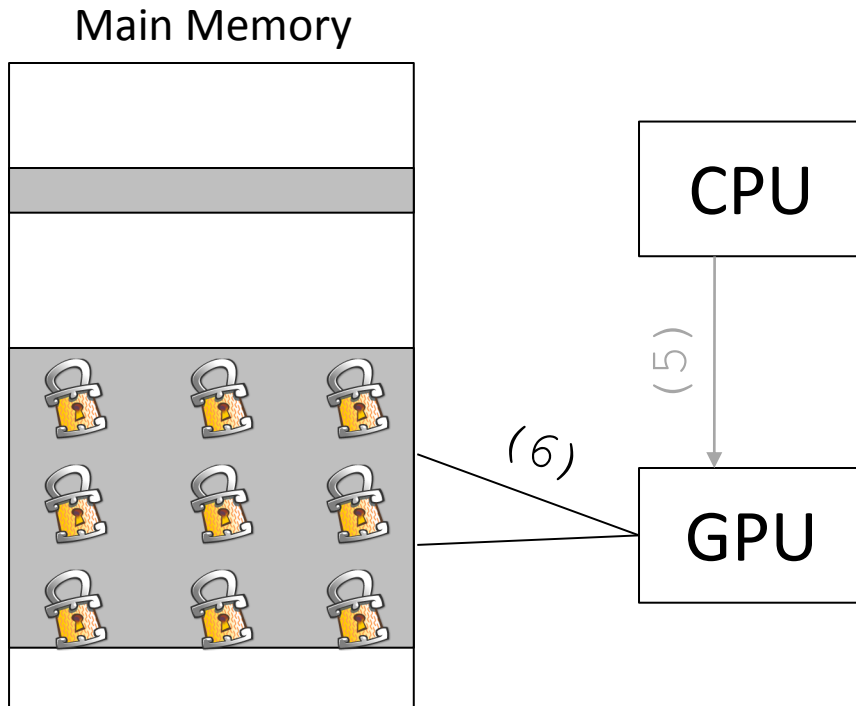
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
- 4. control**
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (13/19)



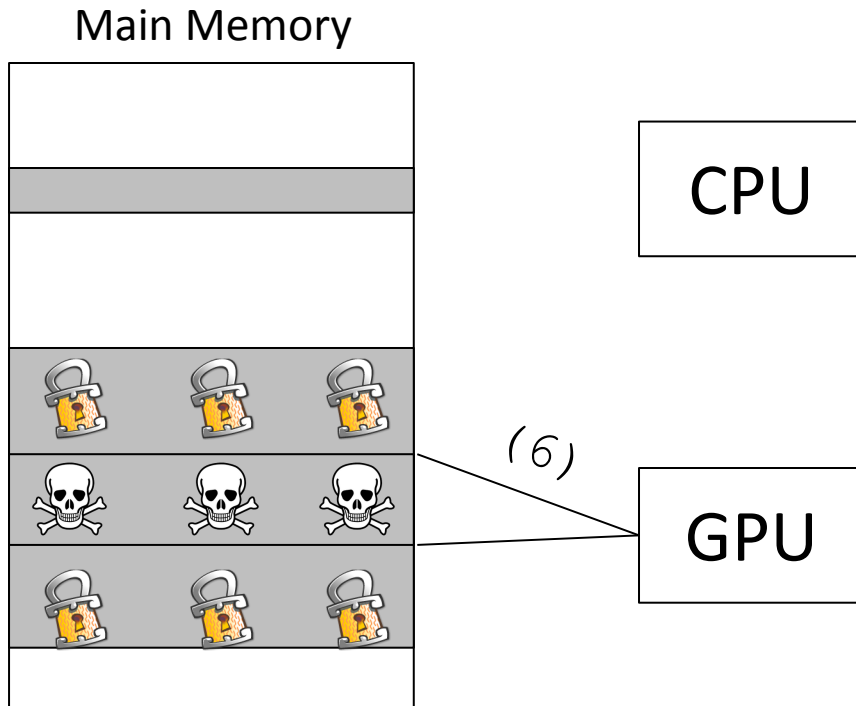
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
- 5. invoke GPU**
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (14/19)



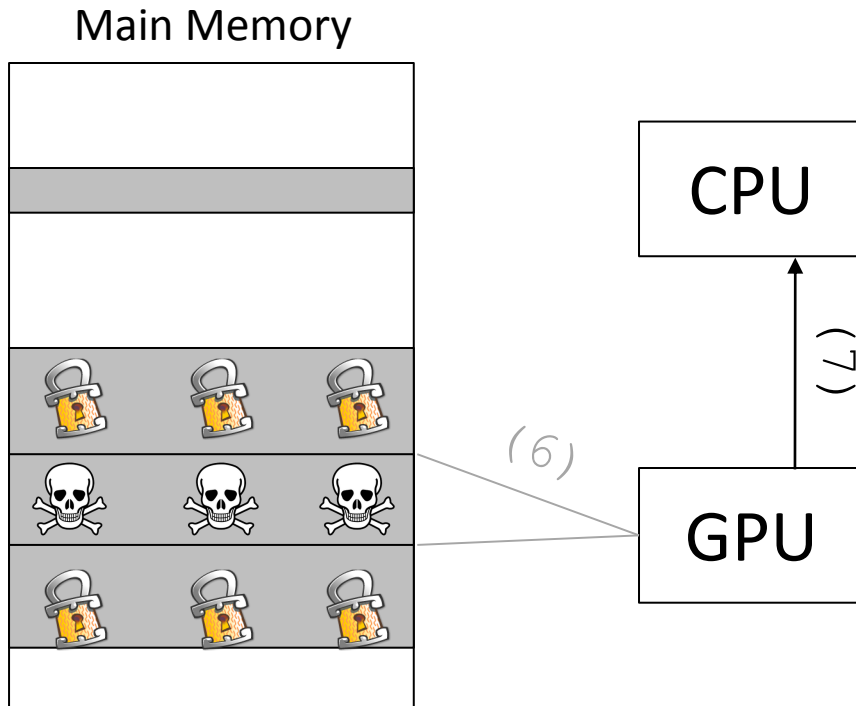
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
- 6. unpack code segment**
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (14/19)



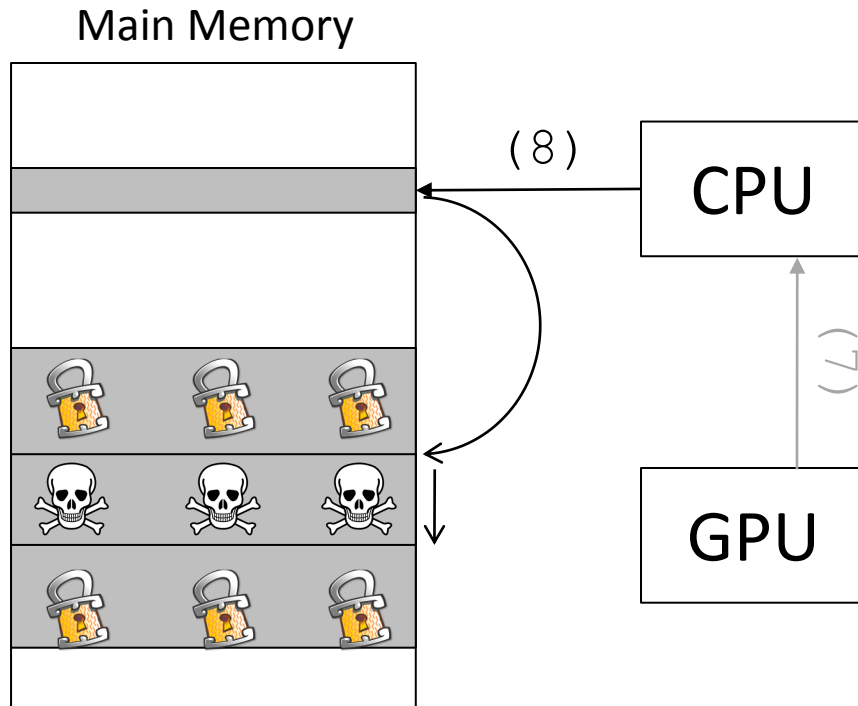
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
- 6. unpack code segment**
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (15/19)



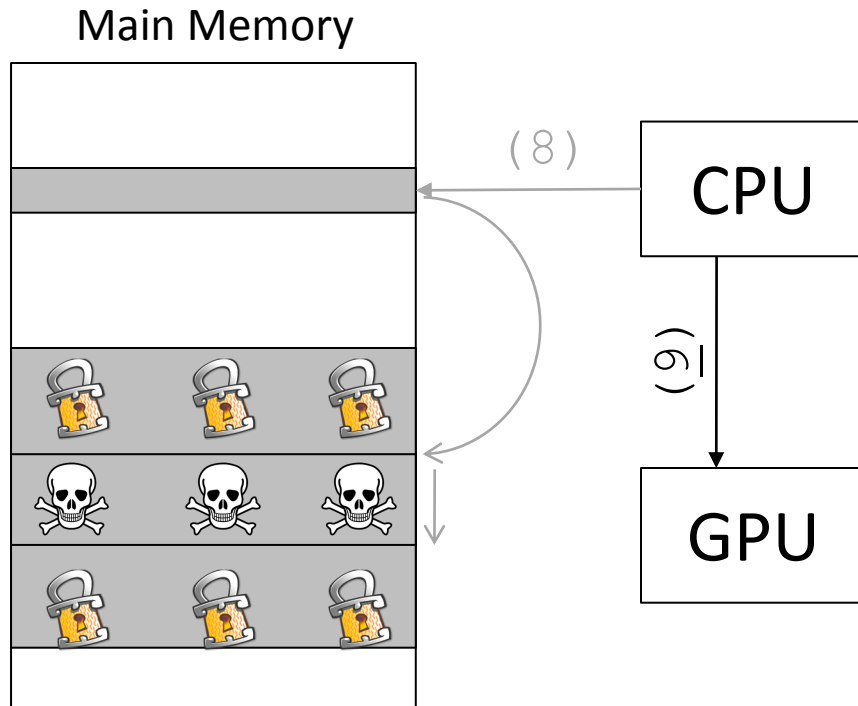
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. **GPU return**
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (16/19)



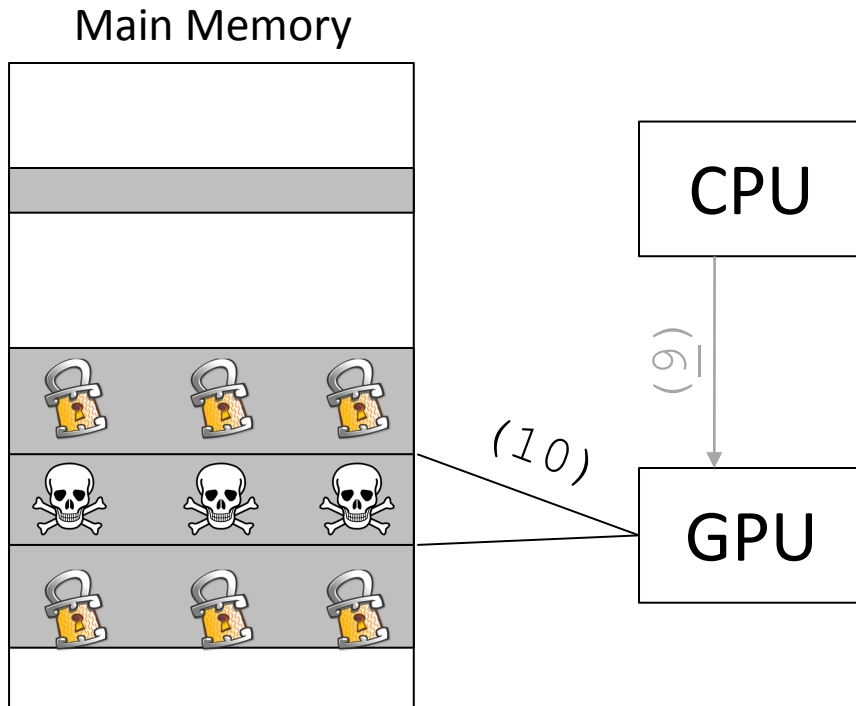
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. **exec malicious code**
9. invoke GPU
10. pack code
11. GPU return

Execution example (17/19)



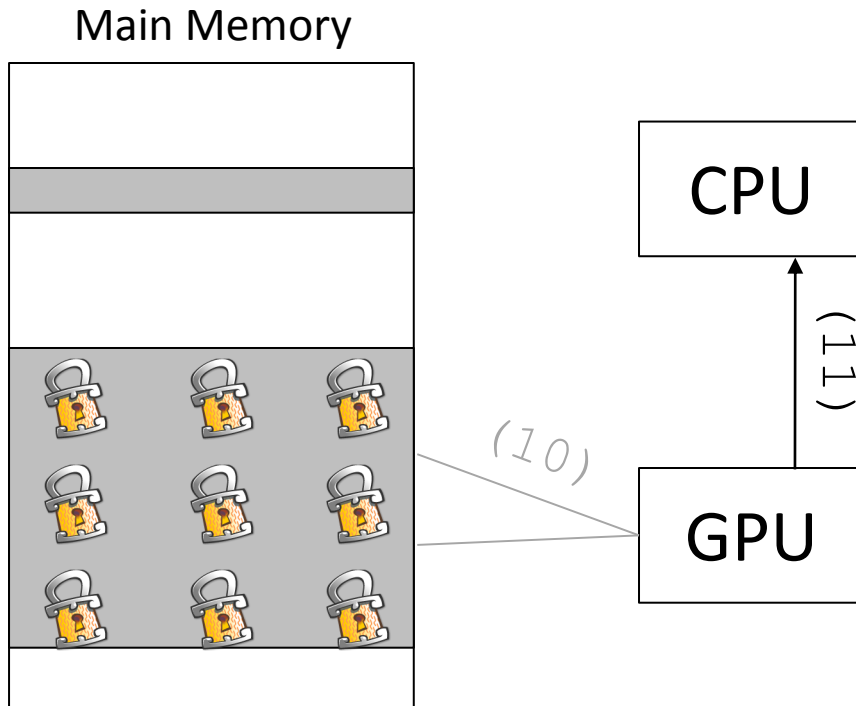
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
- 9. invoke GPU**
10. pack code
11. GPU return

Execution example (18/19)



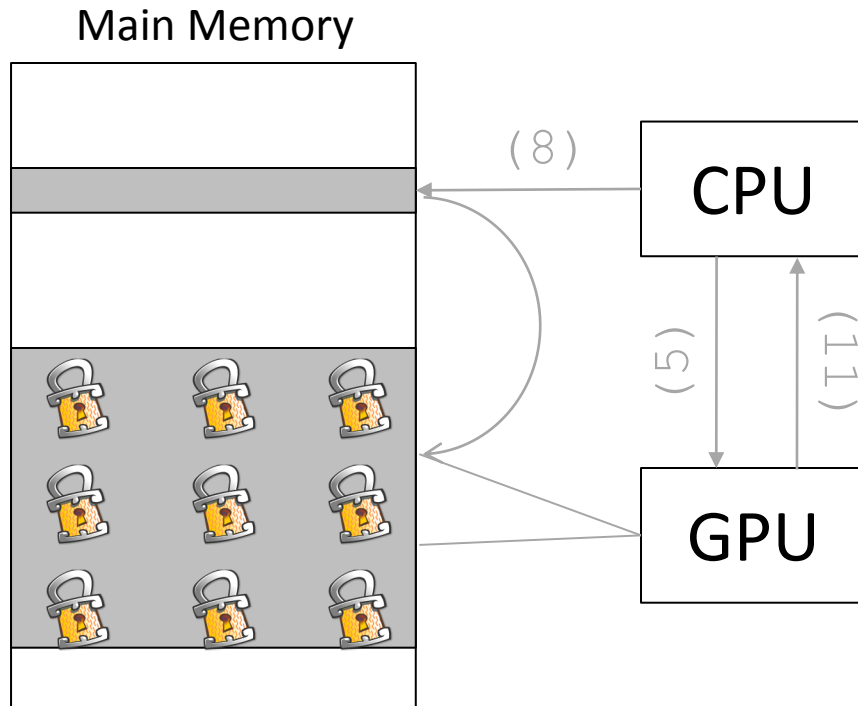
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
- 10. pack code**
11. GPU return

Execution example (19/19)



1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
- 11. GPU return**

Big Picture



1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Run-time polymorphism: Strengths

- Only the parts of code needed at any given point are decrypted each time
- GPU can use different randomly-generated encryption key every time
 - Malware code mutates constantly
- Each encryption key is stored in device memory
 - Not accessible from CPU

FUTURE ATTACKS

MALWARE 2010
KNOW YOUR ENEMY

Research • Practical Solutions (Industry Track) • The Law

19 October 2010

IEEE
 **computer
society**

Potential threats

- Offload computationally intensive operations
 - e.g. password cracking
- Access framebuffer contents
 - Harvest private data displayed on the user screen
 - Display false, benign-looking information
- Migrate completely to the graphics card
 - A malware that run solely on the GPU environment

CONCLUSIONS

MALWARE 2010
KNOW YOUR ENEMY

Research • Practical Solutions (Industry Track) • The Law

19 October 2010

IEEE
 **computer
society**

Summary

- We demonstrated how a malware can **increase its robustness** against detection using the **GPU**
 - Unpacking
 - Run-time polymorphism
- **Taking a step further**, graphics cards may be a promising **new environment** for future malware
 - Password cracking
 - Framebuffer attacks
 - ... and many more

GPU-Assisted Malware

Thank you!

Giorgos Vasiliadis
Michalis Polychronakis
Sotiris Ioannidis

gvasil@ics.forth.gr
mikepo@cs.columbia.edu
sotiris@ics.forth.gr

Publicity

Where the graphics &...

HOT HARDWARE

THE HOTTEST TECH,
TESTED AND BURNED IN

PROCESSORS GRAPHICS / SOUND MOTHERBOARDS STORAGE

HOME REVIEWS VIDEOS IMAGES FORUMS BLOG

September 28th

Just what we need using the GPU.

- mruef** Recommends <http://bit.ly/b4UP9q> about 2 hours ago via twitter
- cauce** Researchers The Register <http://bit.ly/b4UP9q> about 2 hours ago via bitly
- secureslinger** #co with GPU-assisted malware about 3 hours ago via divvy
- ITGuard** Researcher <http://reg.cx/1KTJ> about 4 hours ago via vix
- yongdaek** GPU-assisted malware about 4 hours ago via Twitter
- megatechnews** Researcher <http://vigo.im/4XUH> about 5 hours ago via VigoBB
- InternetSecure** Researchers up evlness ante with GPU-assisted malware <http://ow.ly/19cLZk> about 6 hours ago via Ping.fm
- graphicssoftware** Researchers up evlness ante with GPU-assisted malware <http://bit.ly/9iFiqT> about 6 hours ago via twitterfeed
- alexlevinson** @TeaWithCarl Researchers Up Evlness Ante With GPU-Assisted Malware <http://bit.ly/9ie8sy> about 6 hours ago via twitterfeed
- jaasiya** Researchers up evlness ante with GPU-assisted malware.

Electronics at amazon.com

Digital Camera TVs & More.

THQ Reco for the

IT IS WHAT IT IS.

Log In Create Account Help Subscribe Firehose

Malware Running On Graphics Cards

Posted by **CmdirTaco** on Monday September 27, @12:22PM from the freeing-up-the-cpu dept.

An anonymous reader writes "Given the great potential of general-purpose computing on graphics processors, it is only natural to expect that

Ultra Durable 3 2X more PCI Express lanes 2X more SATA ports Lower BIOS boot times Forme Core 3

No.1 USB 3.0 Motherboard Brand

Leader in Motherboard Innovation

DEFINITIVE TECHNOLOGY NEWS & REVIEWS

Latest content all content news reviews previews interviews first looks tech explained articles

chassis enterprise graphics mainboards network notebooks software storage systems

HEXUS.events :: Click here to read the latest from Intel Developer Forum Fall 2010 - San Francisco

Researchers create GPU-assisted malware

Published: Wed 29th Sep, 2010 | Author: Pete Mason
Computer Technology (CPU Technology) contact: NVidia (GPU Technology) contact:

New Whitepaper Claims GPUs Threaten Malware

Wednesday, September 29, 2010

For the past 3.5 years or so, the computational platform capable hasn't targeted, however, is the malware authors. The idea that before, but according to a paper attack vector whose popularity c

The trio argues that all the comp scientific or graphical workloads

Hardware Software Music & Media Networks Security Public Sector Business

Crime Malware Enterprise Security Spam ID

Live A Webcast 18th October - 3pm BST

Managing User Smartphones

Trying to keep smartphones off your network? **Register Here**

A few of the reasons to join El Reg

Print Post comment Retweet Facebook

Researchers up evlness ante with GPU-assisted Malware Coming to a PC near you

By **Dan Goodin** in San Francisco • **Get more from this author**
Posted in Malware, 28th September 2010 22:02 GMT

Computer scientists have developed proof-of-concept malware that evades traditional security defenses by running on a PC's graphics processor.

The prototype taps a PC's GPU to decrypt, or "unpack," a malicious payload from a file just prior to being run on a targeted machine. Self-unpacking techniques are a common way to defeat signature-based anti-virus scanning because they allow authors to make small changes to the compression or encryption every day or so without altering the core attack code. Up until now, the unpacking had to be performed by a PC's CPU, which places practical limits on the types of packing that can be used.

ENERMAX POWER. INNOVATION. DESIGN.

Twister BEARING New Twister Bearing Fans

TBVEGAS TBVEGAS

MAIN MENU FORUMS ARCHIVES SEARCH CONTACT FOLDING

Αυτοκατάσραση οθόνων laptop www.pcepress.gr

Γλιτώστε μέχρι και 50% από την τιμή του κατασκευστή

Περισσότερα >

Ads by Google

Tuesday September 28, 2010

GPU Assisted Malware

This paper on GPU assisted malware ([link to pdf](#)) is a bit frightening to say the least. Thanks to forum member Cerulean for the heads up!

MALWARE 2010

KNOW YOUR ENEMY

Research • Practical Solutions (Industry Track) •

IEEE computer society