GPU-Assisted Malware

Giorgos Vasiliadis — ICS-FORTH, Greece
Michalis Polychronakis — Columbia University, USA
Sotiris Ioannidis — ICS-FORTH, Greece

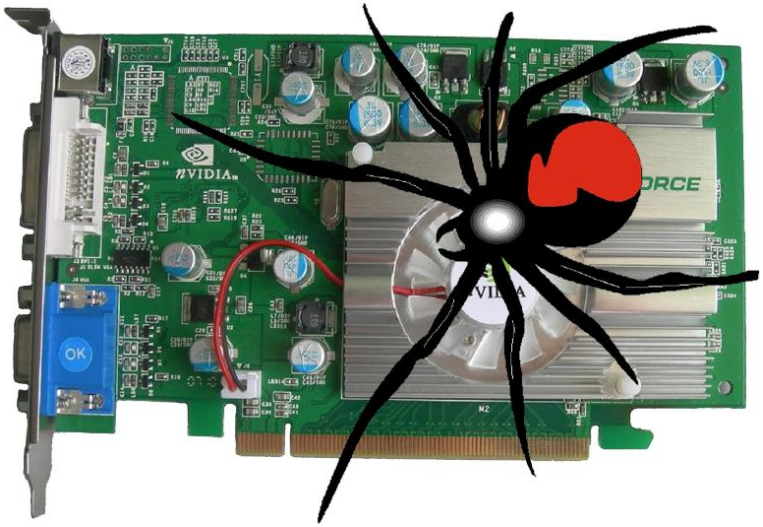
MALWARE 2010
KNOW YOUR ENEMY
Research • Practical Solutions (Industry Track) • The Law

19 October 2010

IEEE computer society

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 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 = ?

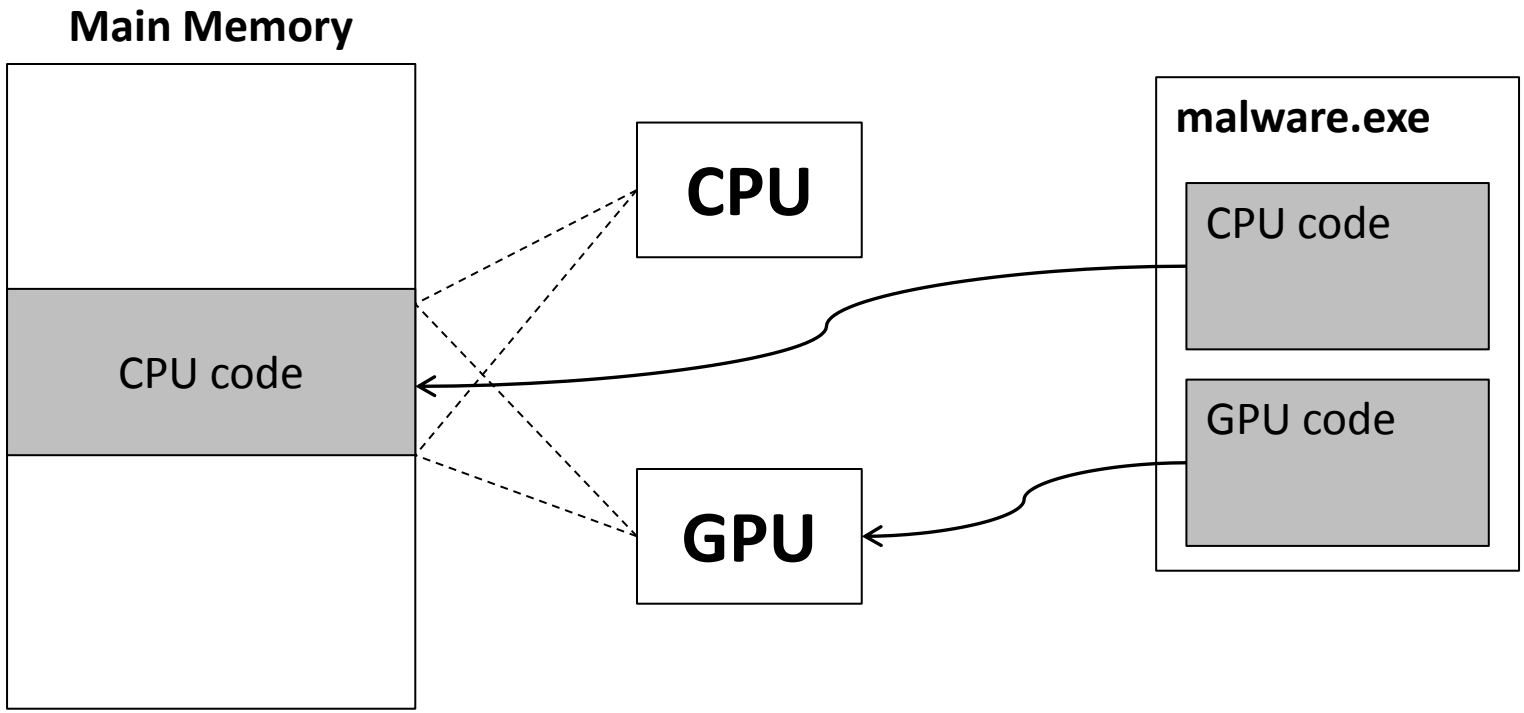
- Is it possible for a malware to **exploit** the rich functionality of **modern GPUs**?
  - GPU Powered Malware [Reynaud'08]

This work

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

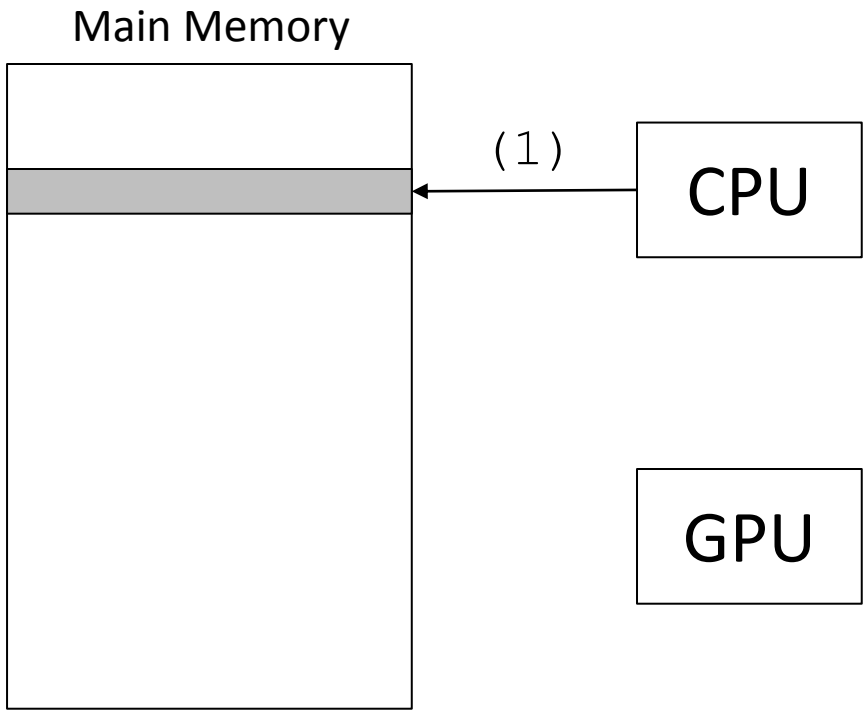
SELF-UNPACKING

Basic design

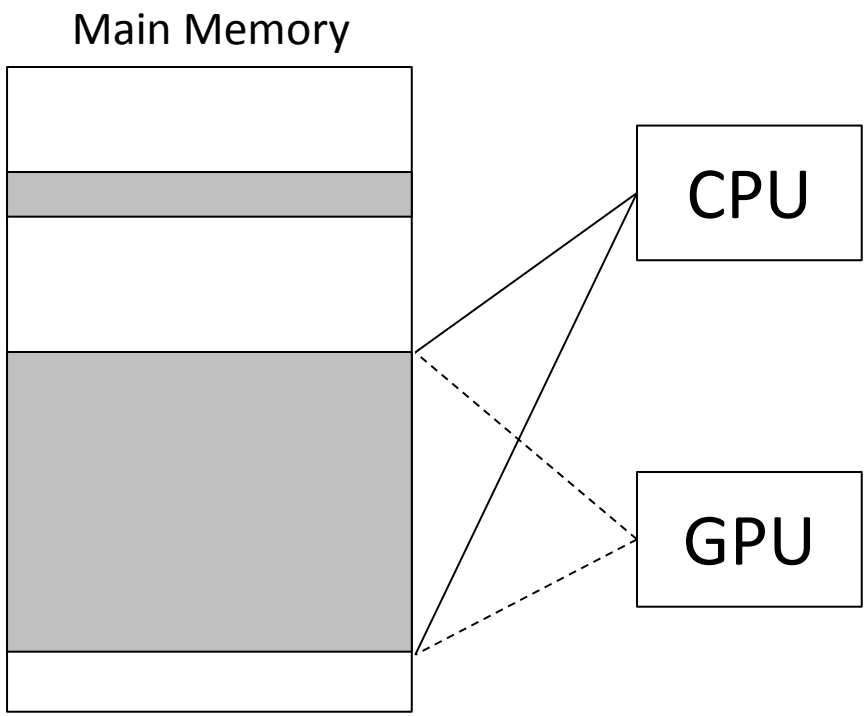


Execution example (1/7)

1. **`init / bootstrap`**

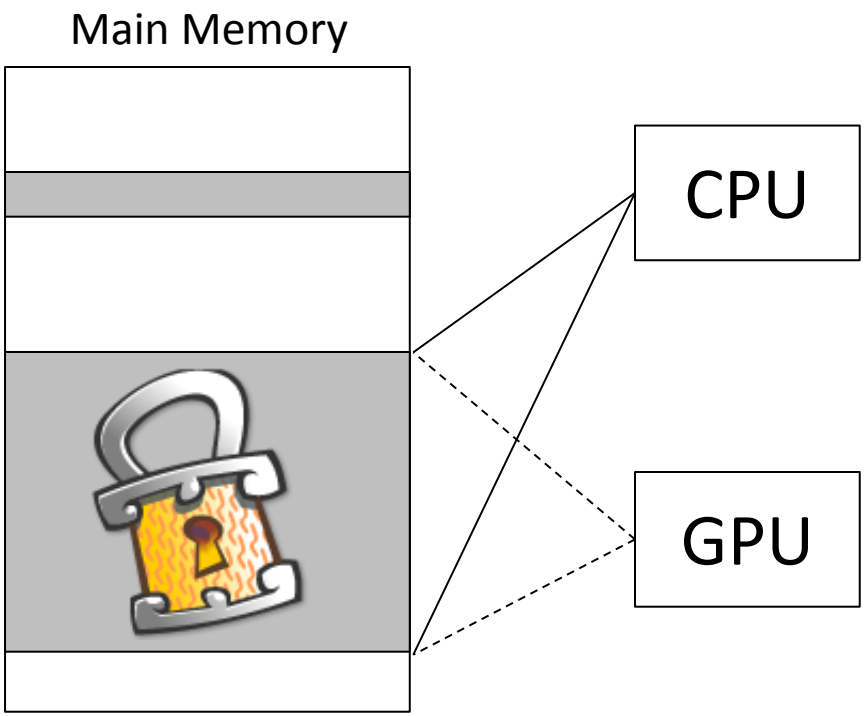


Execution example (2/7)



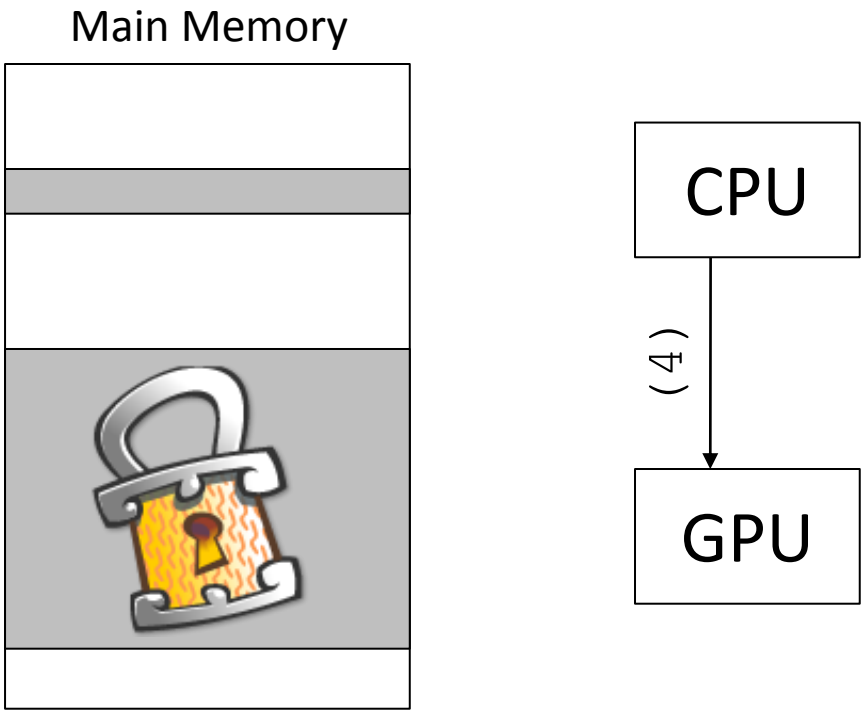
1. `init / bootstrap`
2. **`alloc mmap'ed buffer`**

Execution example (3/7)



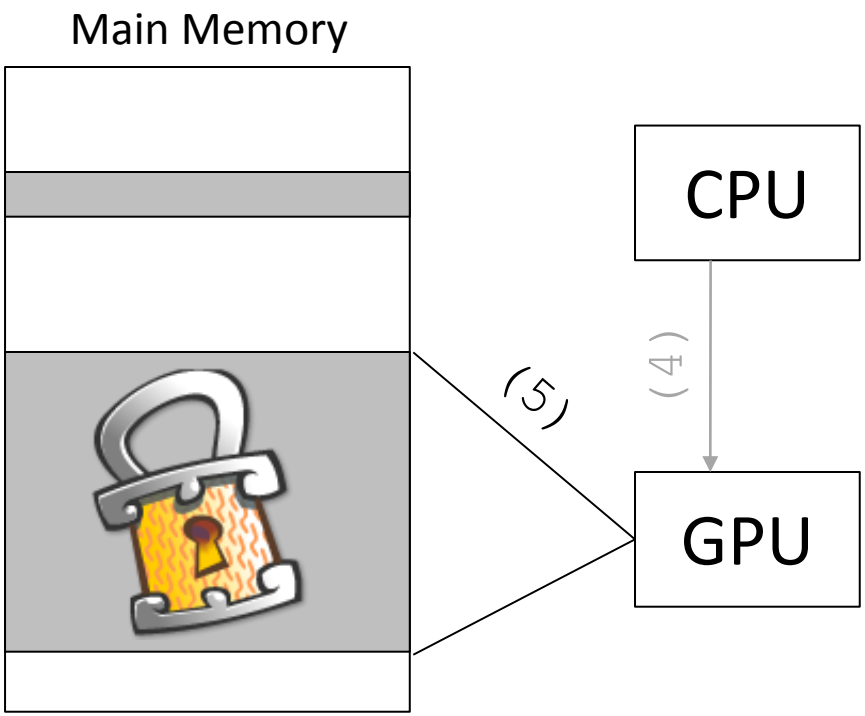
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. **`store packed code`**

Execution example (4/7)



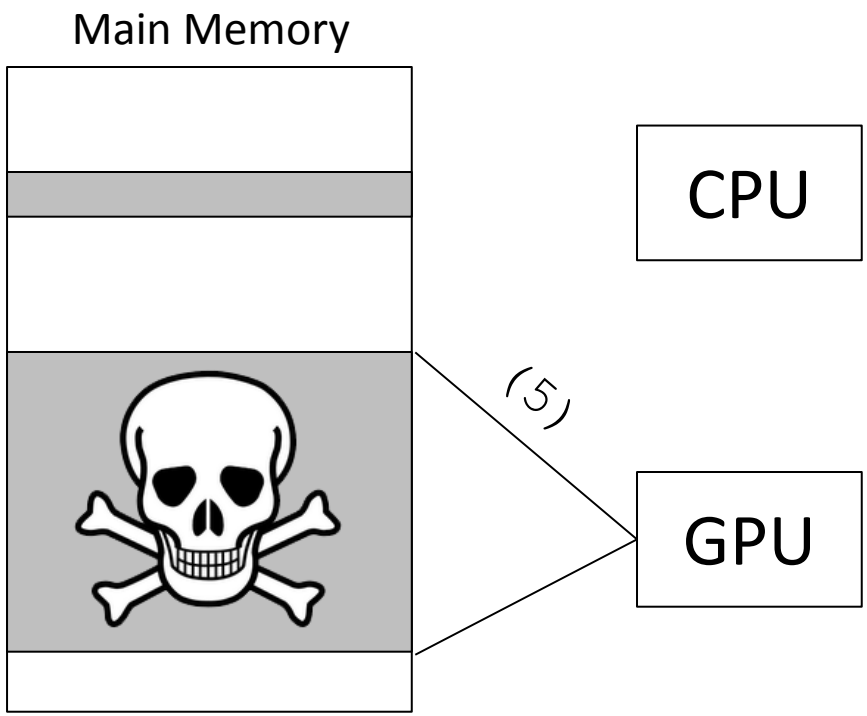
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. **invoke GPU**

Execution example (5/7)



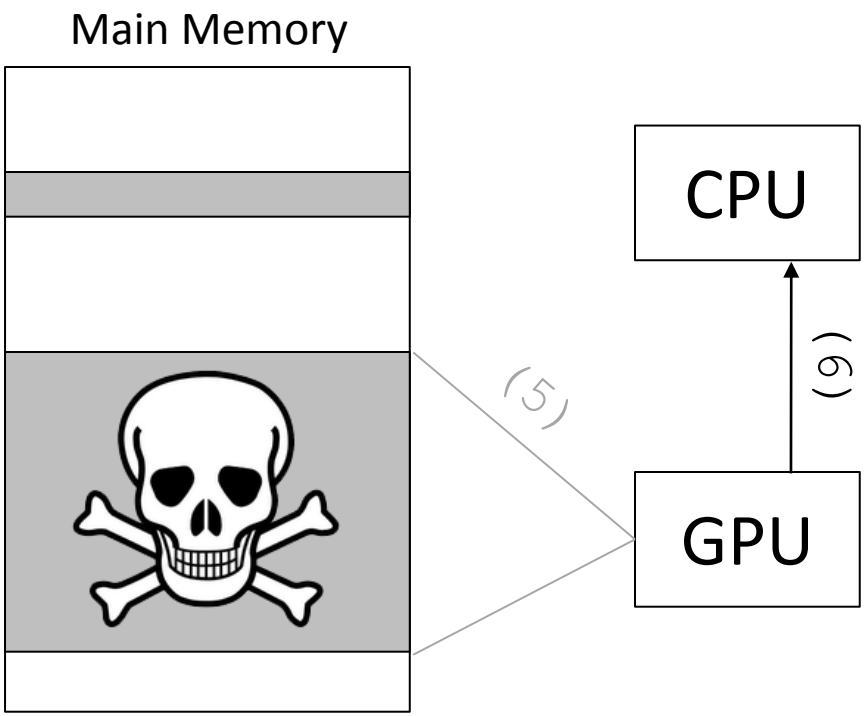
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. invoke GPU
5. **unpack malicious code**

Execution example (5/7)



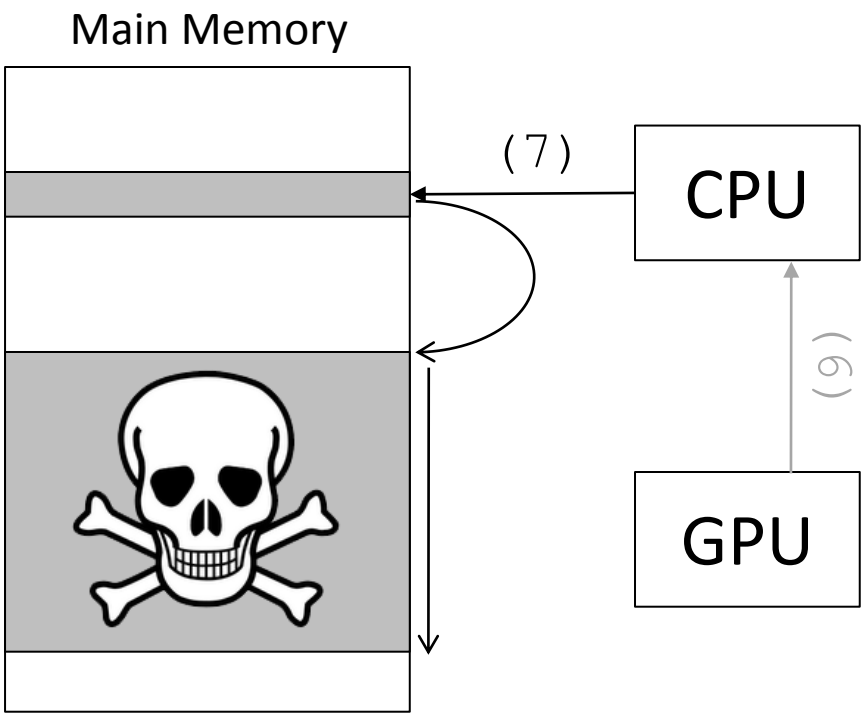
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. `invoke GPU`
5. **`unpack malicious code`**

Execution example (6/7)



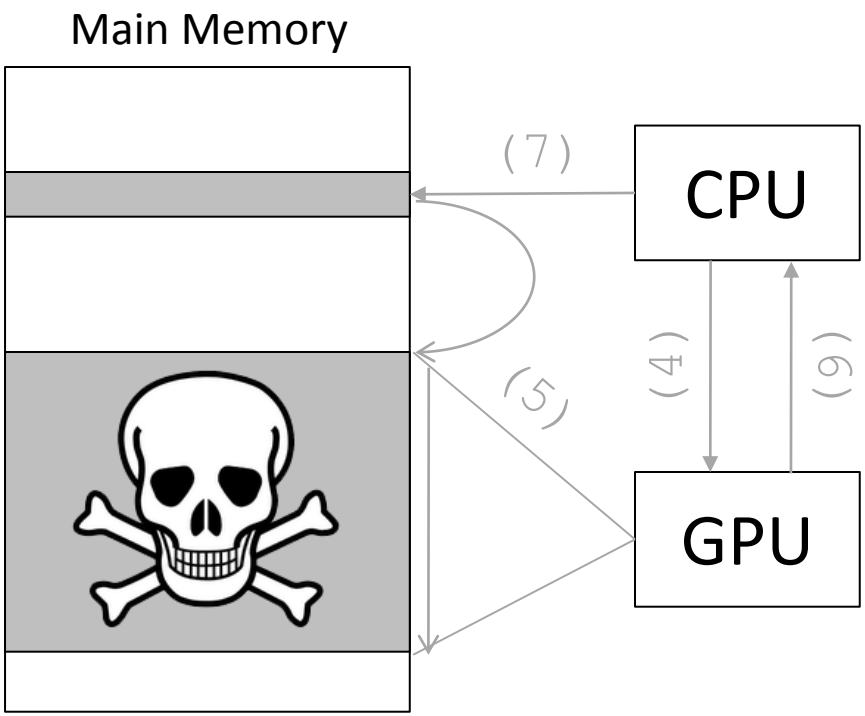
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. invoke GPU
5. unpack malicious code
6. **GPU return**

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

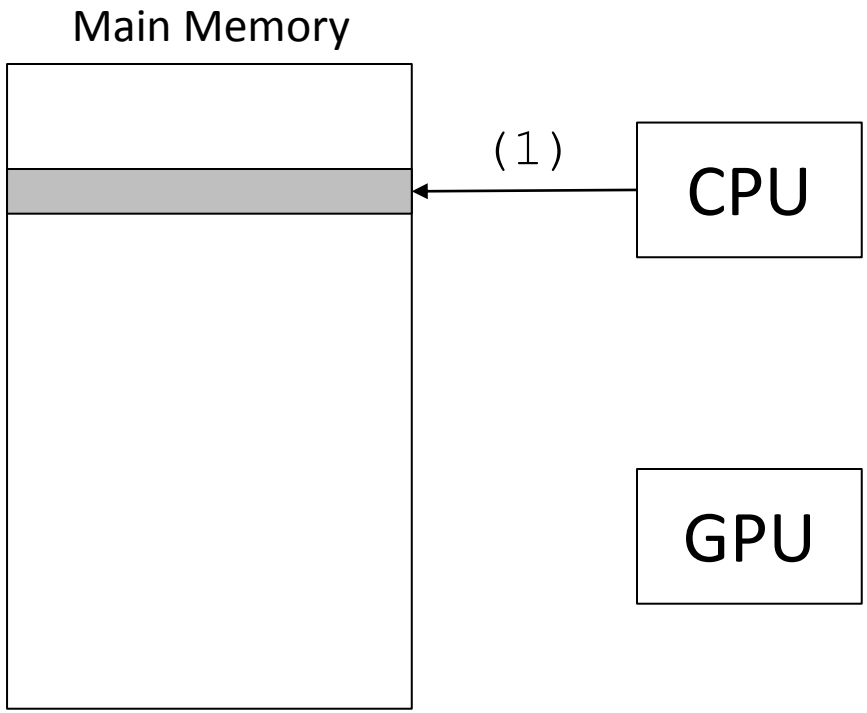
- Malware code lies unencrypted in main memory after unpacking
- Can be detected by dumping the memory

- Can we do better?

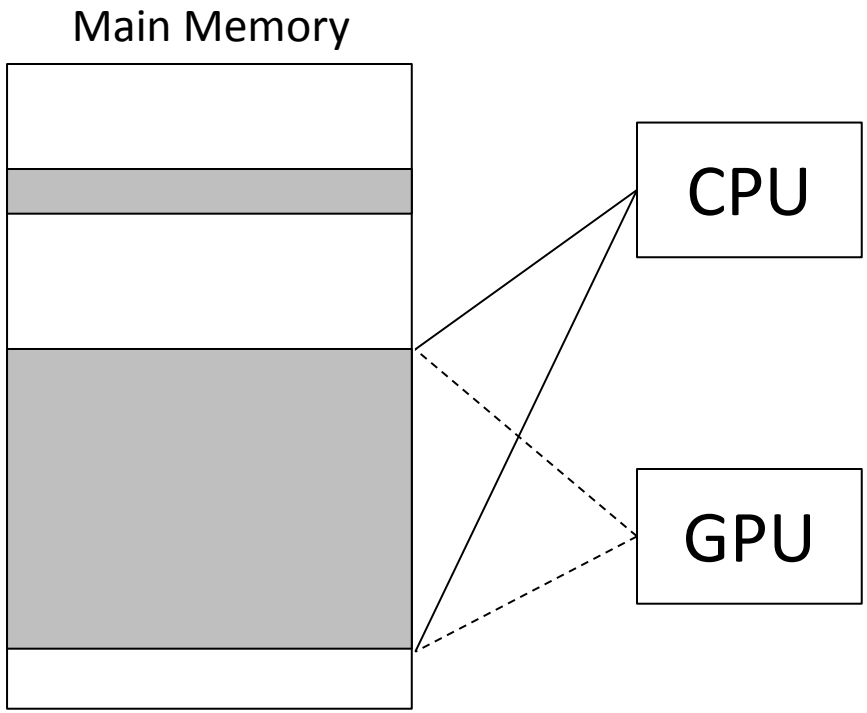
RUN-TIME POLYMORPHISM

Execution example (1/19)

1. **`init / bootstrap`**

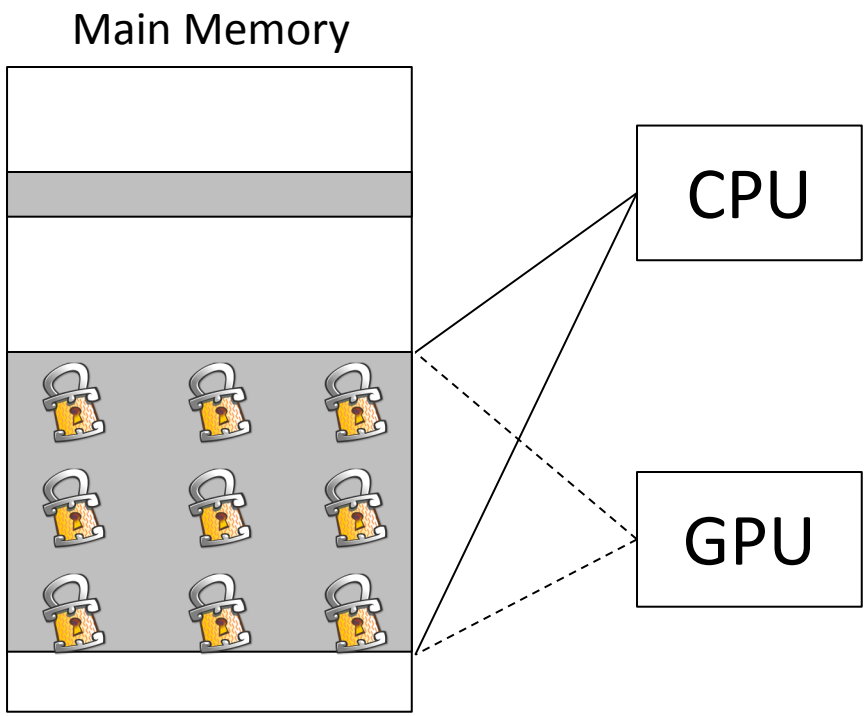


Execution example (2/19)



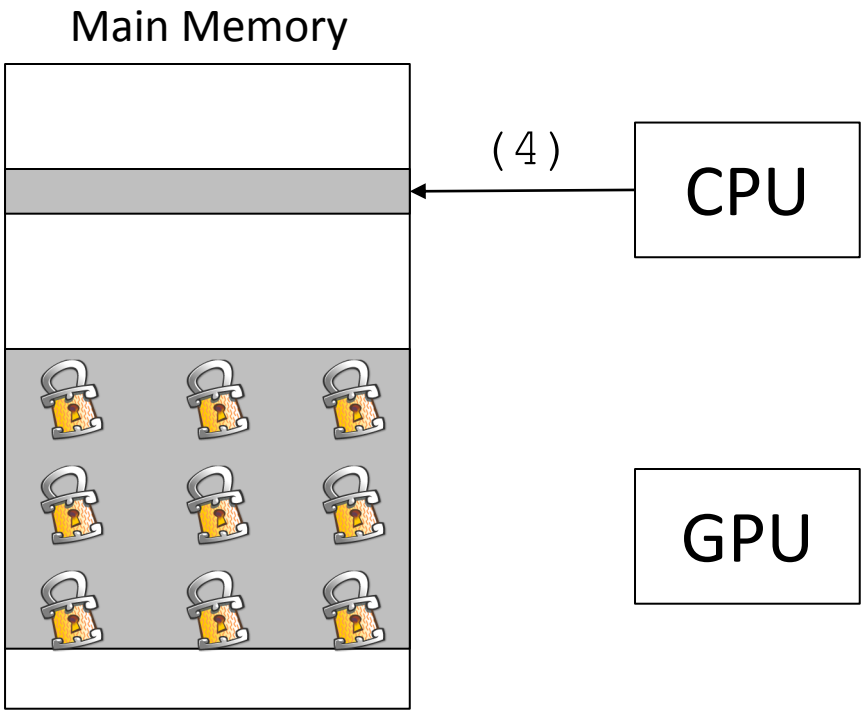
1. `init / bootstrap`
2. **`alloc mmap'ed buffer`**

Execution example (3/19)



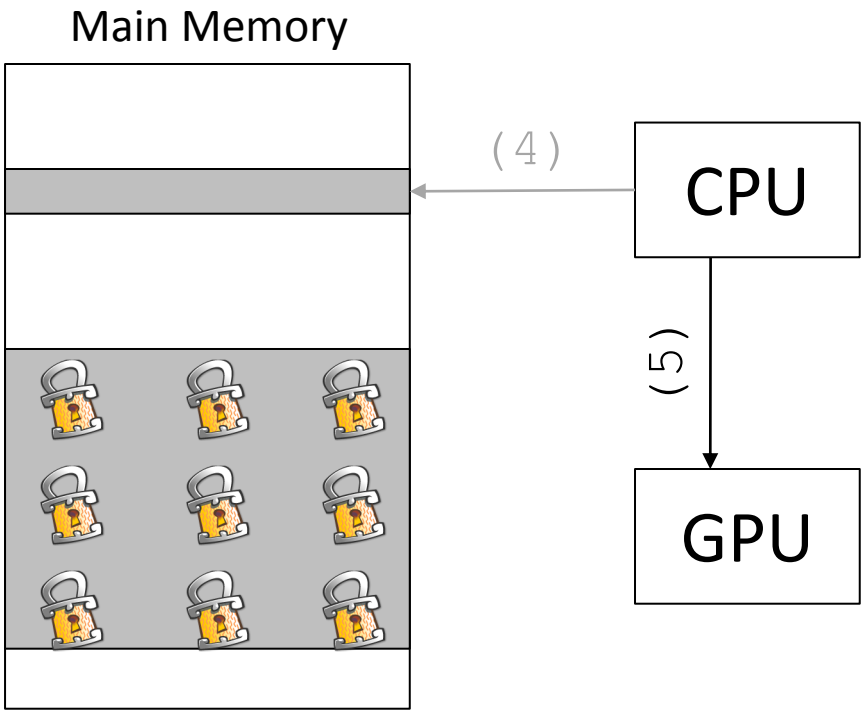
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. **`store packed code`**

Execution example (4/19)



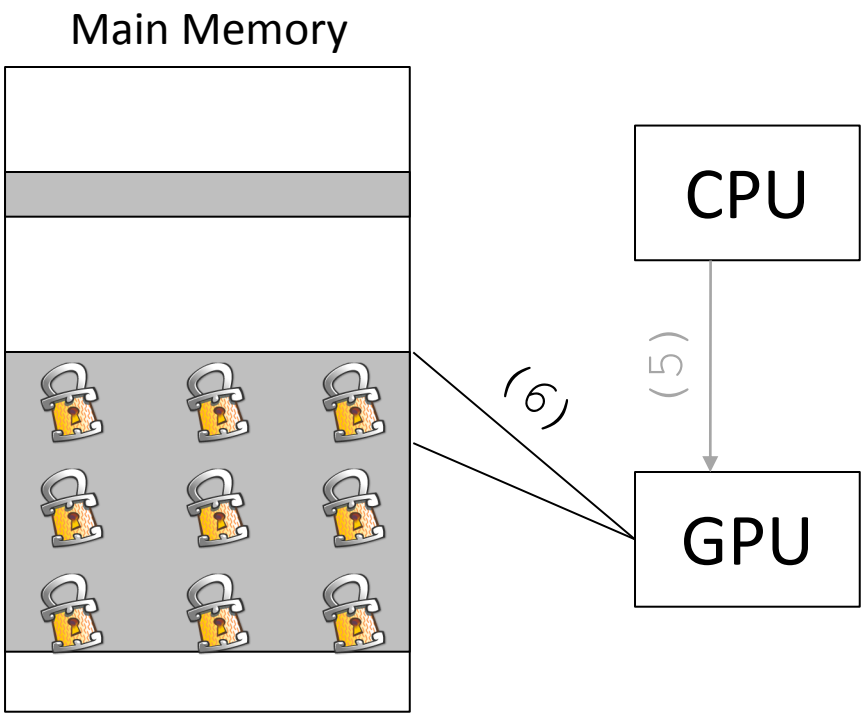
1. `init / bootstrap`
2. `alloc mmap'ed buffer`
3. `store packed code`
4. **`control`**

Execution example (5/19)



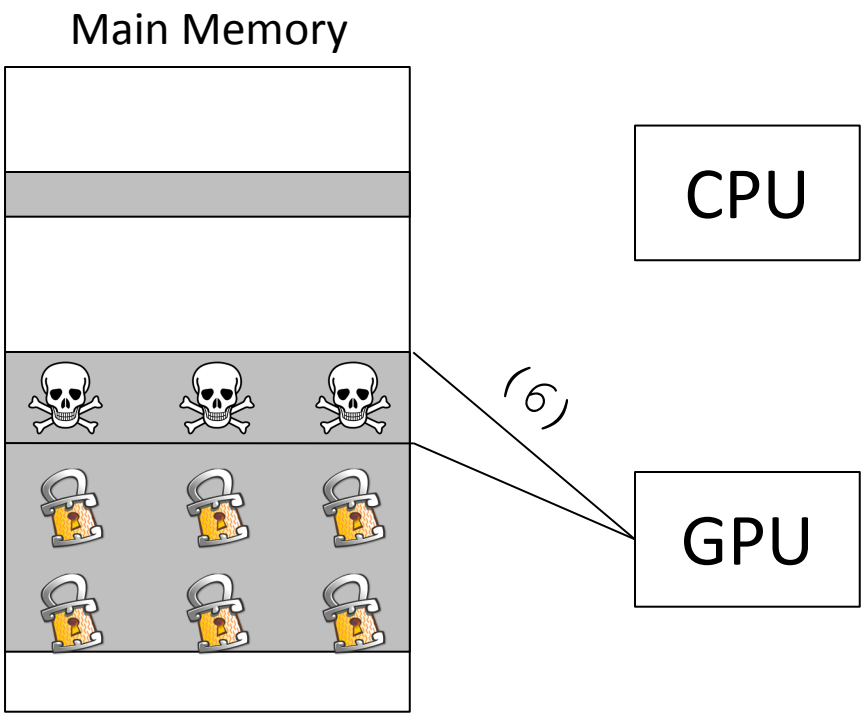
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. **invoke GPU**

Execution example (6/19)



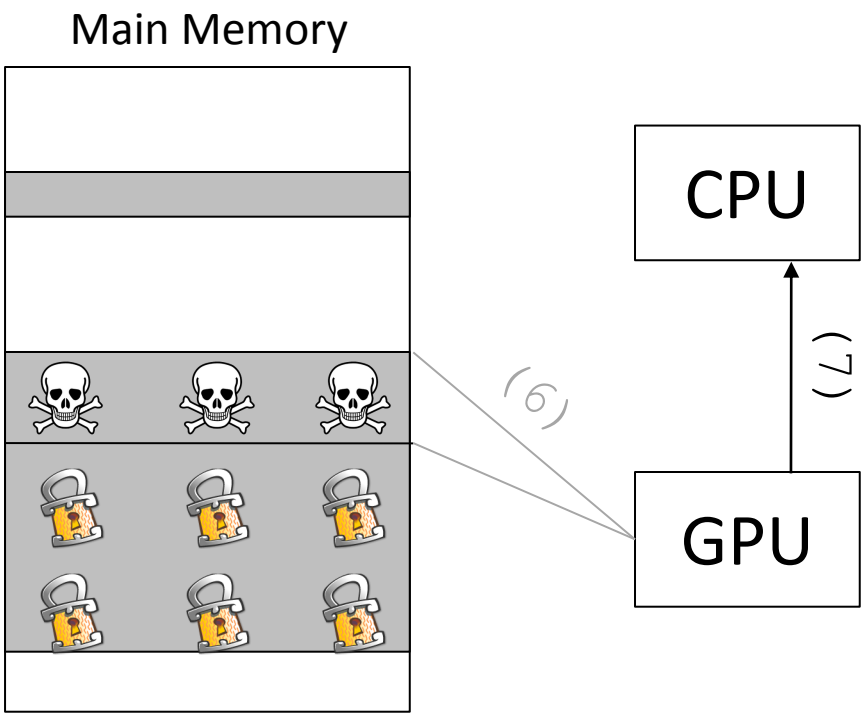
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. **unpack code segment**

Execution example (6/19)



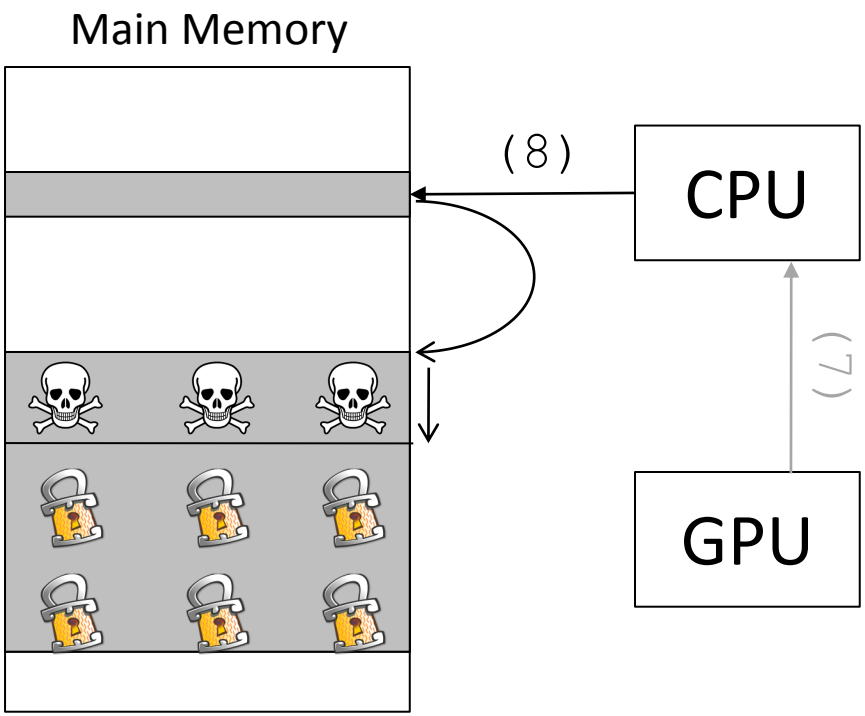
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. **unpack code segment**

Execution example (7/19)



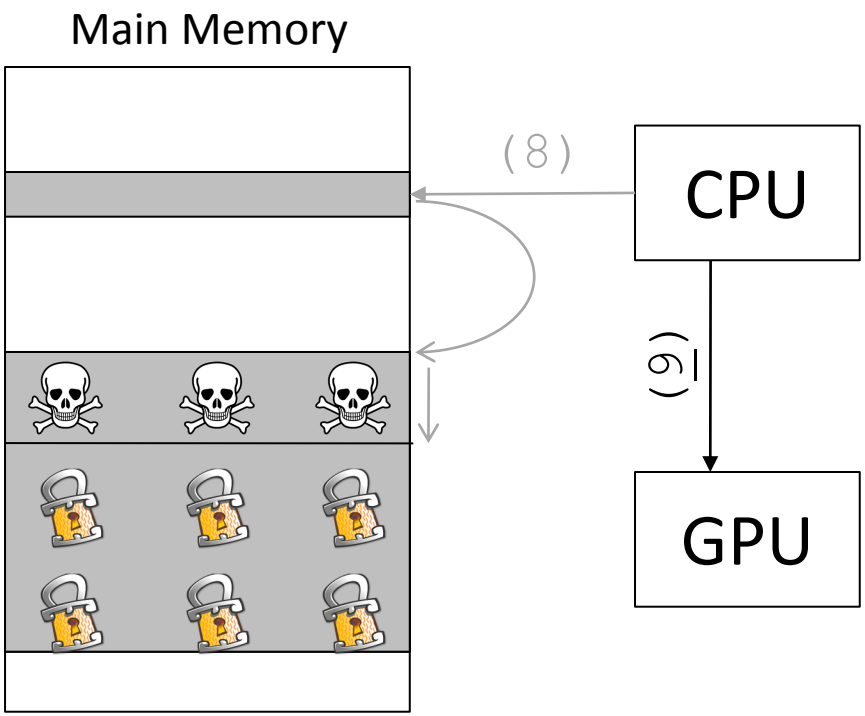
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. **GPU return**

Execution example (8/19)



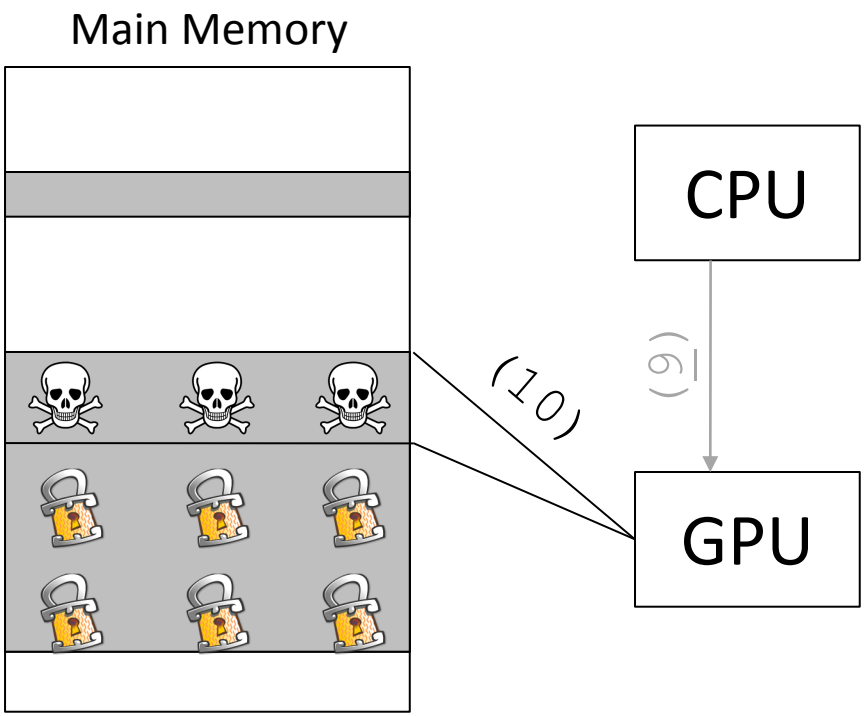
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. **exec malicious code**

Execution example (9/19)



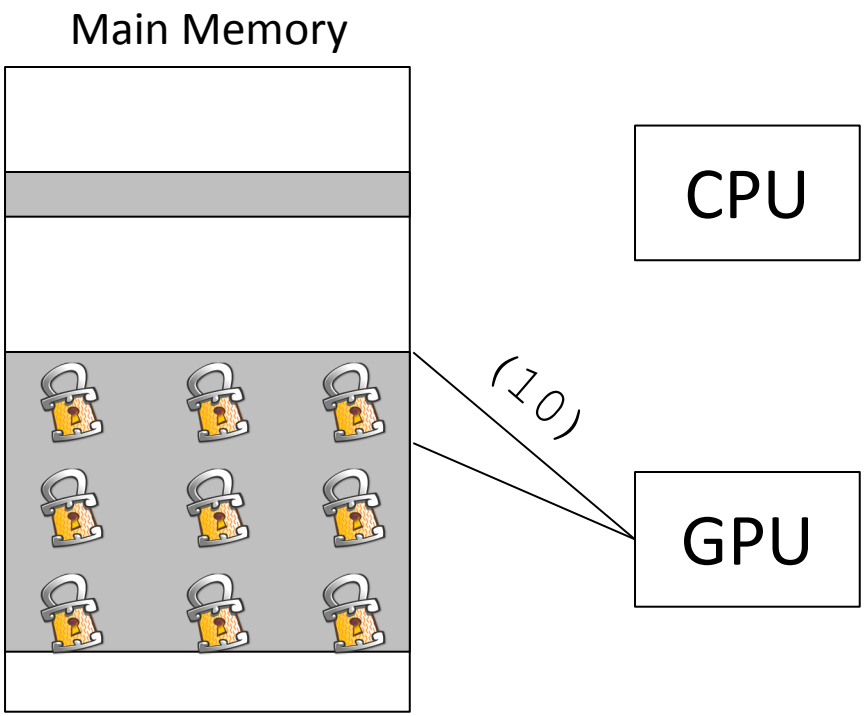
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. **invoke GPU**

Execution example (10/19)



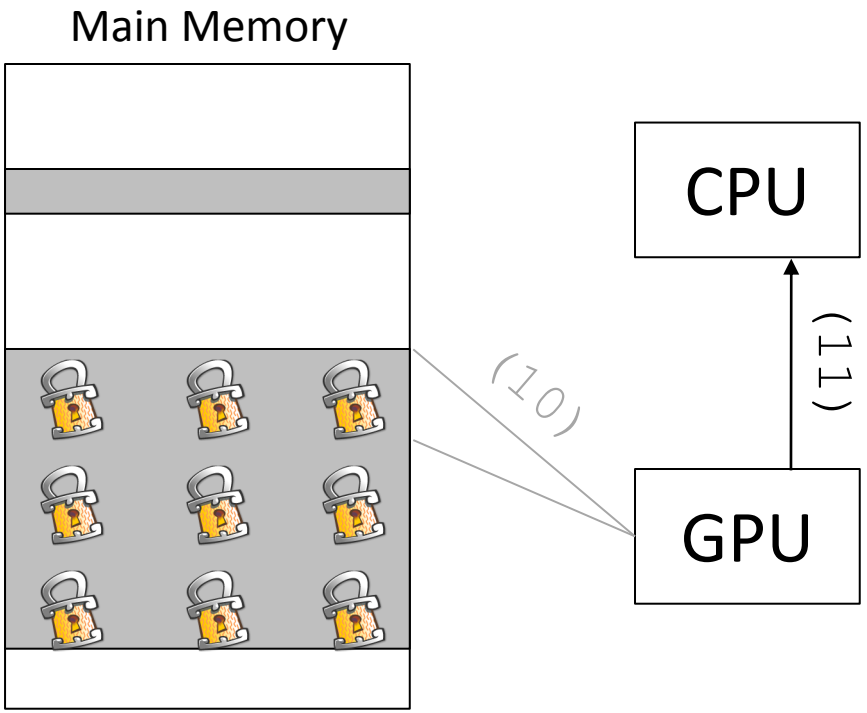
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. **pack code**

Execution example (10/19)



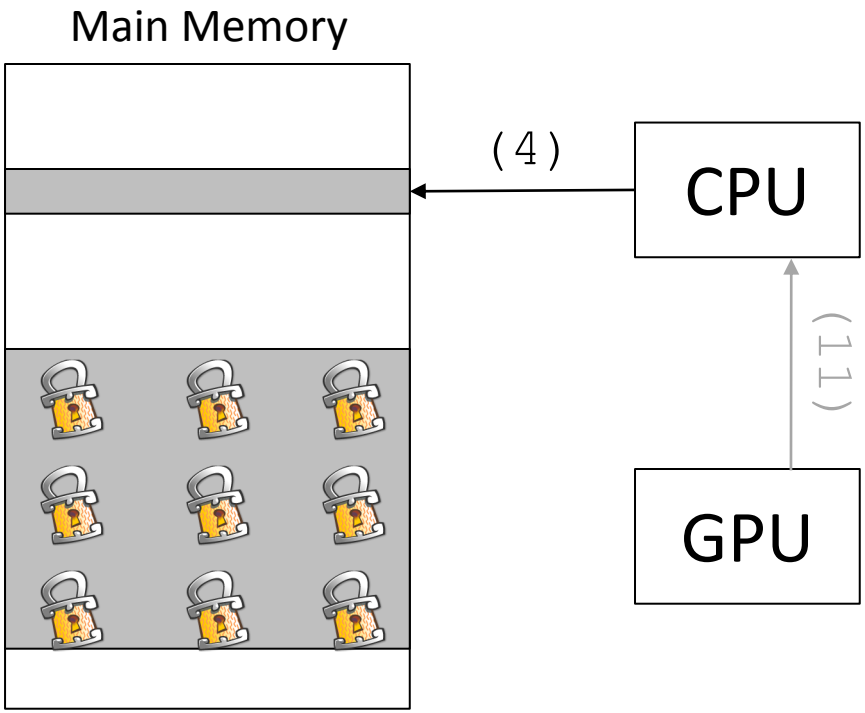
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. **pack code**

Execution example (11/19)



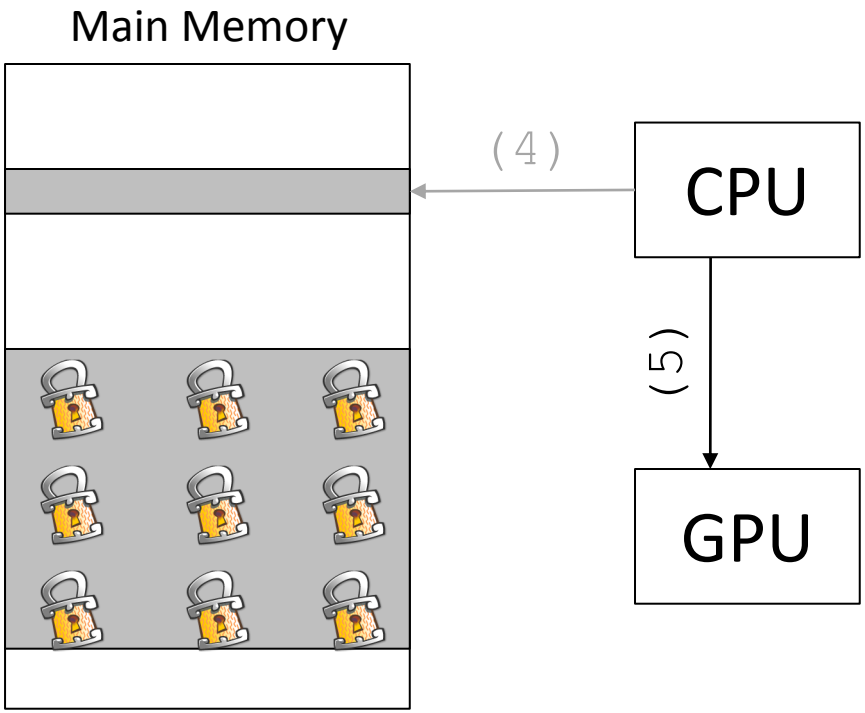
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. **GPU return**

Execution example (12/19)



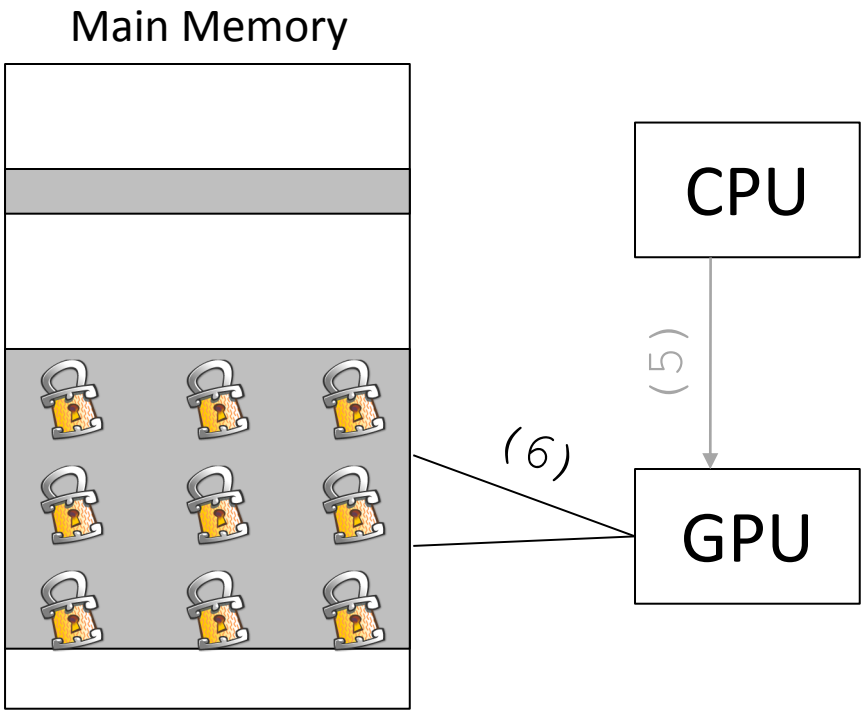
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. **control**
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (13/19)



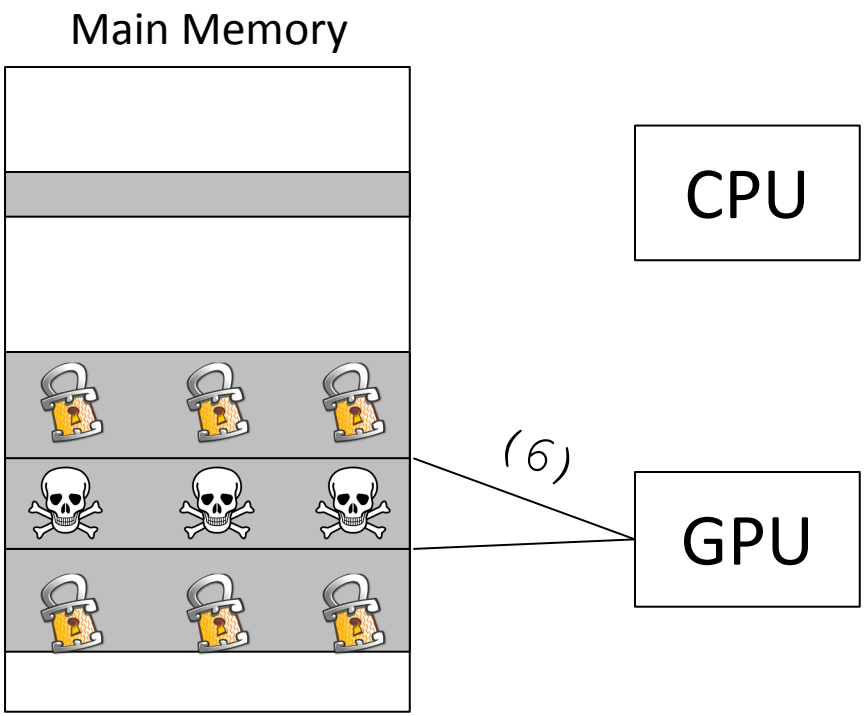
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. **invoke GPU**
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (14/19)



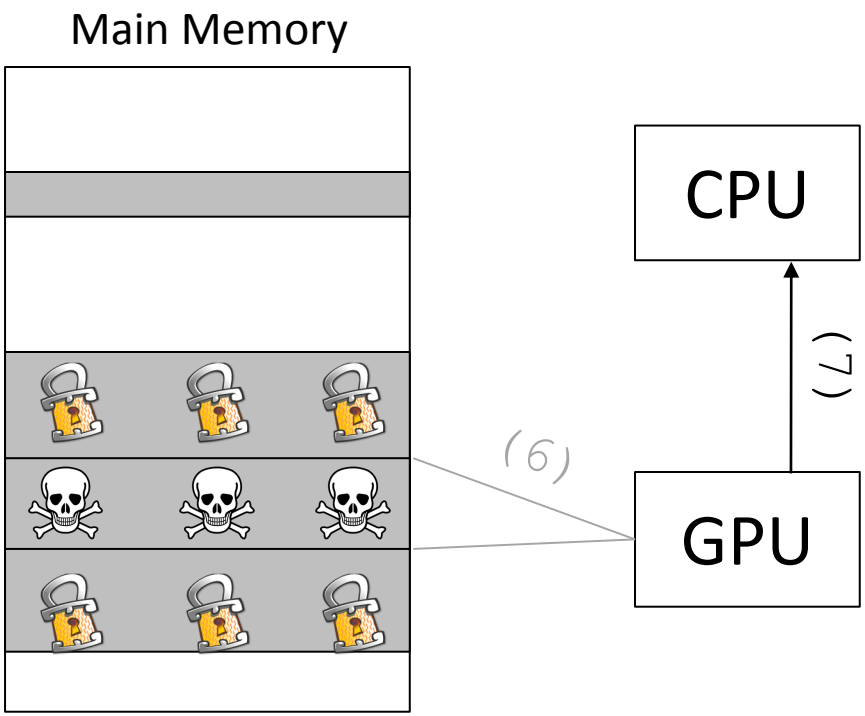
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. **unpack code segment**
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (14/19)



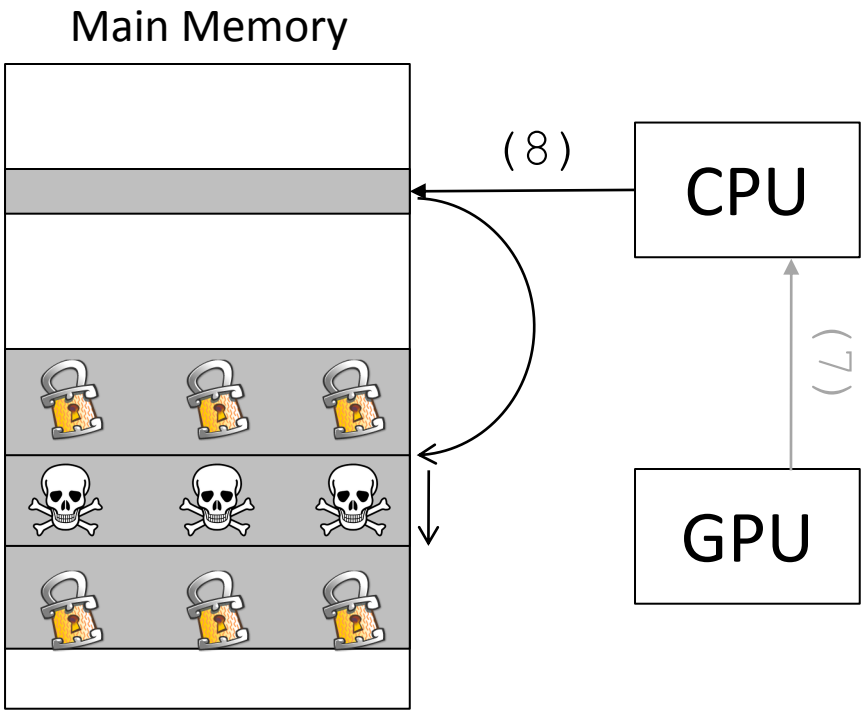
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. **unpack code segment**
7. GPU return
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (15/19)



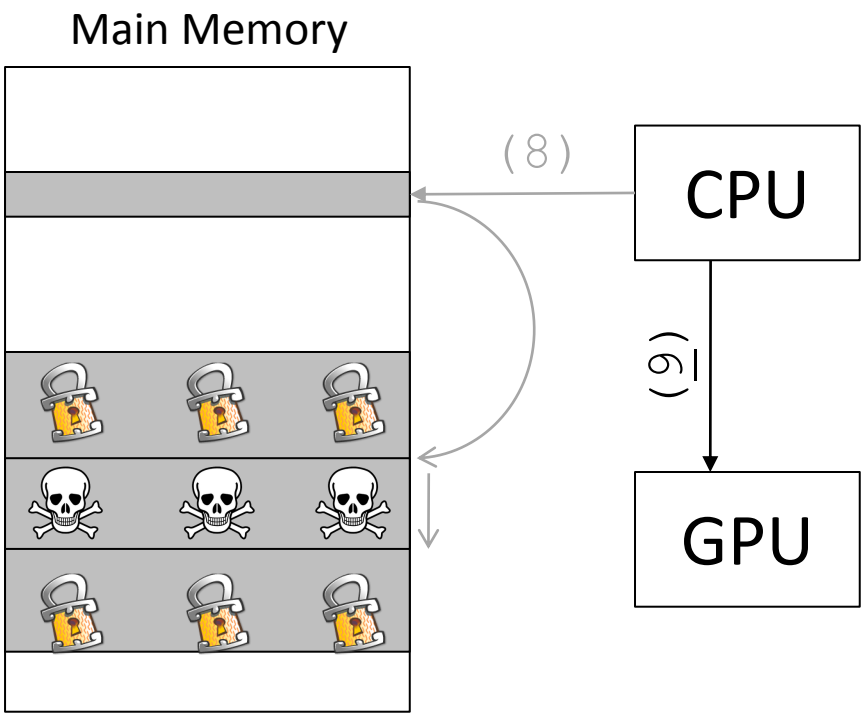
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. **GPU return**
8. exec malicious code
9. invoke GPU
10. pack code
11. GPU return

Execution example (16/19)



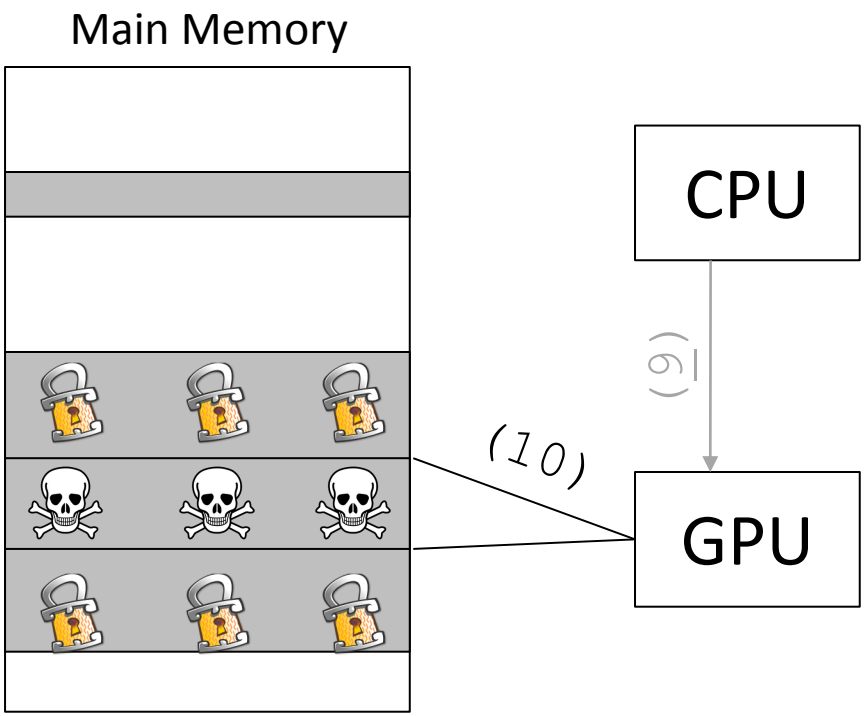
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. **exec malicious code**
9. invoke GPU
10. pack code
11. GPU return

Execution example (17/19)



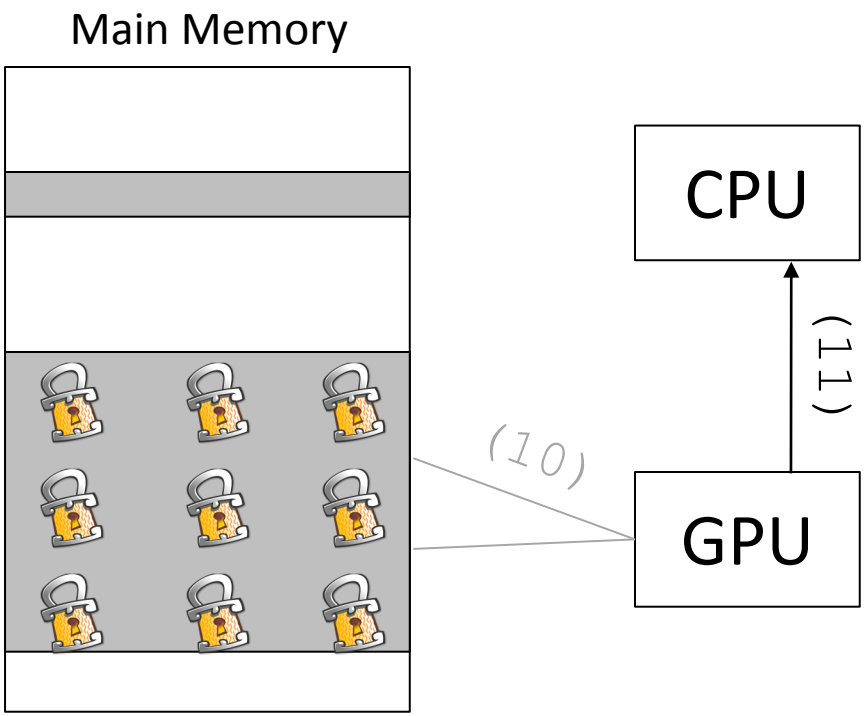
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. **invoke GPU**
10. pack code
11. GPU return

Execution example (18/19)



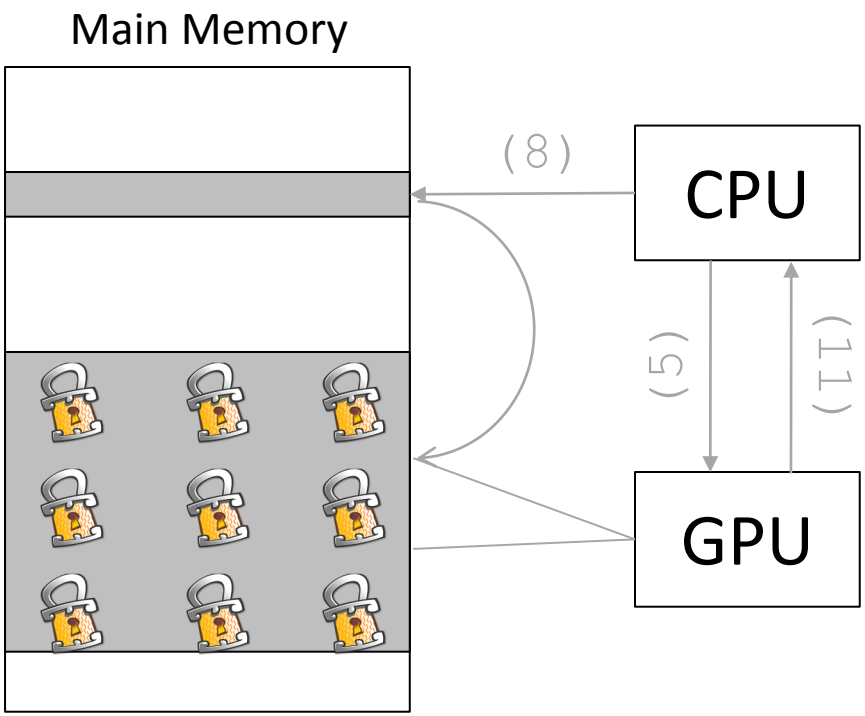
1. init / bootstrap
2. alloc mmap'ed buffer
3. store packed code
4. control
5. invoke GPU
6. unpack code segment
7. GPU return
8. exec malicious code
9. invoke GPU
10. **pack code**
11. GPU return

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

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

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 gvasil@ics.forth.gr
Michalis Polychronakis mikepo@cs.columbia.edu
Sotiris Ioannidis sotiris@ics.forth.gr

Publicity


VIZWORLD
Where the graphics &
HOT HARDWARE
THE HOTTEST TECH, TESTED AND BURNED IN
PROCESSORS
GRAPHICS / SOUND
MOTHERBOARDS
STORAGE
HOME
REVIEWS
VIDEOS
IMAGES
FORUMS
New Whitepaper Claims GPUs Threaten Malware
Wednesday, September 29, 2010
For the past 3.5 years or so,
computational platform capable
hasn't targeted, however, is the
malware authors. The idea that
before, but according to a paper
attack vector whose popularity
The trio argues that all the comp
scientific or graphical workloads
MALWARE 2010
KNOW YOUR ENEMY
Research • Practical Solutions (Industry Track) •
Slashdot
IT IS WHAT IT IS.
Malware Running On Graphics Cards
Posted by CmdrTaco on Monday September 27, @12:22PM
from the freeing-up-the-cpu dept.
An anonymous reader writes
HEXUS
DEFINITIVE TECHNOLOGY NEWS & REVIEWS
GIGABYTE
No.1 USB 3.0 Motherboard
Researchers create GPU-assisted malware
The Register
Hardware
Software
Music & Media
Networks
Security
Public Sector
Business
Crime
Malware
Enterprise Security
Spam
ID
Managing User Smartphones
Trying to keep smartphones off your network?
A few of the reasons to join El Reg
Researchers up evilness 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.
[H]ardOCP
ENERMAX
POWER. INNOVATION. DESIGN.
New Twister Bearing Fans
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 [H] forum member Cerulean for the heads up!
IEEE computer society