Stealth Measurements for Cheat Detection in On-line Games

Ed Kaiser

Wu-chang Feng

Travis Schluessler

Portland State University

intel

Cheating Affects On-line Games

- Frustrates legitimate players
    - not fun to play against cheaters
    - can't tell if good opponents are cheating or not
- Impacts profitability of game developer
    - existing players quit in frustration
    - bad reputation inhibits potential new players

The Cheating Problem

- On-line games simulate environments too complex for a server to render for its clients
- Client is **trusted** to run the game accurately
  - obey physics
  - keep secrets from player
- Cheat is software that abuses the trust
  - accomplish feats that the cheater is <u>unable</u> or <u>unwilling</u> to do

Cheater has the Upper Hand …

- They have complete control of machine
  - grant cheat software any privileges necessary
  - they get to run first
- Use advanced techniques of adversary in ‘hard’ security problems (like Rootkits)
  - cloaking / timely unloading
  - debugging / virtualizing game
  - spoofing / disabling its defenses

… but Cheating is a Weak Threat

- ‘Security breach’ is not catastrophic
  - private data is not stolen
  - machine is not used to attack network hosts
  - can easily undo the damage
- No urgency for detection and capture
  - cheater is connected for long periods
  - cheats target small portion of system

Detection is Sufficient

- Cheater will eventually be caught
- Clean up is easy
  - ban the account
  - undo results of winning
- Cost of being caught is high
  - loss of CD-key ($30 to $50)
  - loss of paid subscription ($10 per month)
  - voiding any time they actually invested

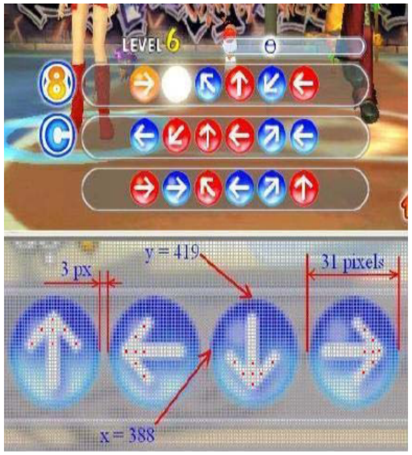
Our Approach

- Detect cheaters using hardware-based measurements
  - securely
  - stealthily with regard to
    - what is being measured
    - when measurements are made

Cheating by Manipulating Data

1) Authorized Automated Read

- collect information presented to user
  - use Graphics Device Interface `BitBlt()` to learn state from screen

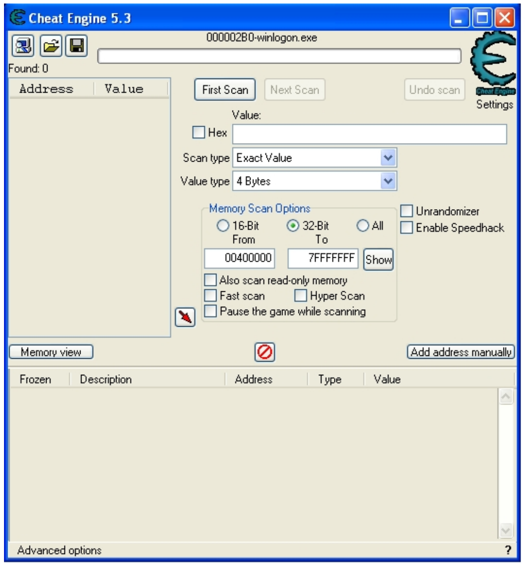


2) Unauthorized Data Read

- reveal state meant to be secret
  - `ReadProcessMemory()`

3) Unauthorized Data Write

- directly modify game state
  - static data (i.e. gravity constant)
  - dynamic data (i.e. as player location)
  - `WriteProcessMemory()`



Cheat Engine 5.3

00000280-winlogon.exe

Found: 0

Address	Value

Value:

□ Hex

Scan type: Exact Value

Value type: 4 Bytes

Memory Scan Options

○ 16-Bit ● 32-Bit ○ All

From: 00400000 To: 7FFFFFFF

□ Also scan read-only memory

□ Fast scan □ Hyper Scan

□ Pause the game while scanning

□ Unrandomizer

□ Enable Speedhack

Frozen	Description	Address	Type	Value

Cheating by Modifying Code

4) Code Injection

- modify game code or add cheat code
  - overwrite game code (*hot patch*)
  - inject code into pockets of allocated but unused executable memory (*code caves*)
  - allocate memory with **`VirtualAllocEx()`** and fill it with code
  - load Dynamic Link Library containing cheat code (*DLL injection*)
    - using **`LoadLibrary()`**
    - modifying the **`AppInit_DLL`** registry entry

Cheating by Changing Execution

5) Direct Function Calls

- execute game or OS functions to change state
  - input using **`keybd_event()`** or **`mouse_event()`**

6) Thread Manipulation

- execute code using a thread within the game process
  - inject a new thread via **`CreateRemoteThread()`**
  - hijack an existing thread (*detour* or *trampoline*)

7) Function Pointer Hooks

- redirect function pointers (*hook*) to cheat code
  - Import Address Table (IAT)
  - Interrupt Descriptor Table (IDT)
  - overwrite return address on stack
  - overwrite Secure Exception Handler (SEH) and raise exception

Cheating by Changing the Game

8) External Processes

- control or modify the game from another process
  - **`DebugActiveProcess()`** to control game execution
  - use **`SendMessage()`** to send input to the game window

9) File Replacement

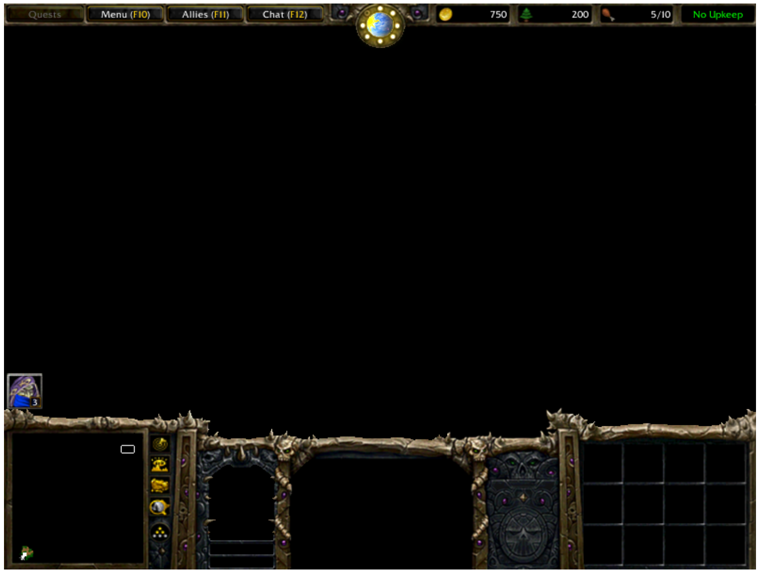
- change game files or build a new game client
  - modify opponent models to be bigger / brighter
  - build a new game client or automation robot that speaks the game's network protocol

Cheating through Hardware

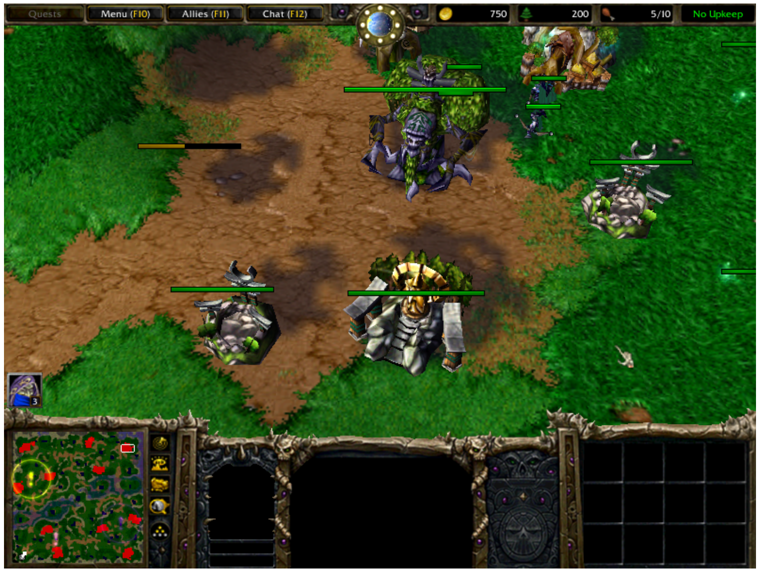
10) Exploit Hardware Facilities

- use registers, hardware debugging, or virtualization features to manipulate the game
  - modify Interrupt Descriptor Table Register (IDTR) to point to a different table of handlers
  - modify control and segment registers (CR0 through CR3) to change page-write permission
  - hardware debugging
  - run the game using hardware virtualization

eg. WarCraft III Maphack



Before



After

- unauthorized write
- code injection (*hot patch*)
  - NOP over visibility check code

Measuring Memory

1) Code Integrity

- integrity check of existing code
  - game and loaded DLL ".text" segments will reveal hot patches, detours

2) Function Pointer Validation

- integrity check function pointer tables
  - game and loaded DLL ".idata" segments will reveal IAT hooks

3) Static Game Data Validation

- integrity check invariant game data
  - reveals unauthorized static data writes (eg. gravity constant)

Measuring more Memory

4) Scan for Injected Pages

- scan for pages inexplicably marked as executable

5) Stack Validation

- check that stack represents legal call chain
  - reveals thread hijacking, thread injection, code injection, debugging and virtualization

6) Memory Watchpoints

- log changes to dynamic data
  - reveals unauthorized dynamic data writes (eg. player team)

Measuring Execution Behavior

7) Instruction Counts

- check distribution of opcode type

8) Check Execution Range

- observe range of execution through EIP register

9) Code Timing

- count cycles to go through game's event loop

10) System Call Behavior

- check distribution and sequence of system calls

Measuring I/O, Registers, and File

11) I/O Path Validation

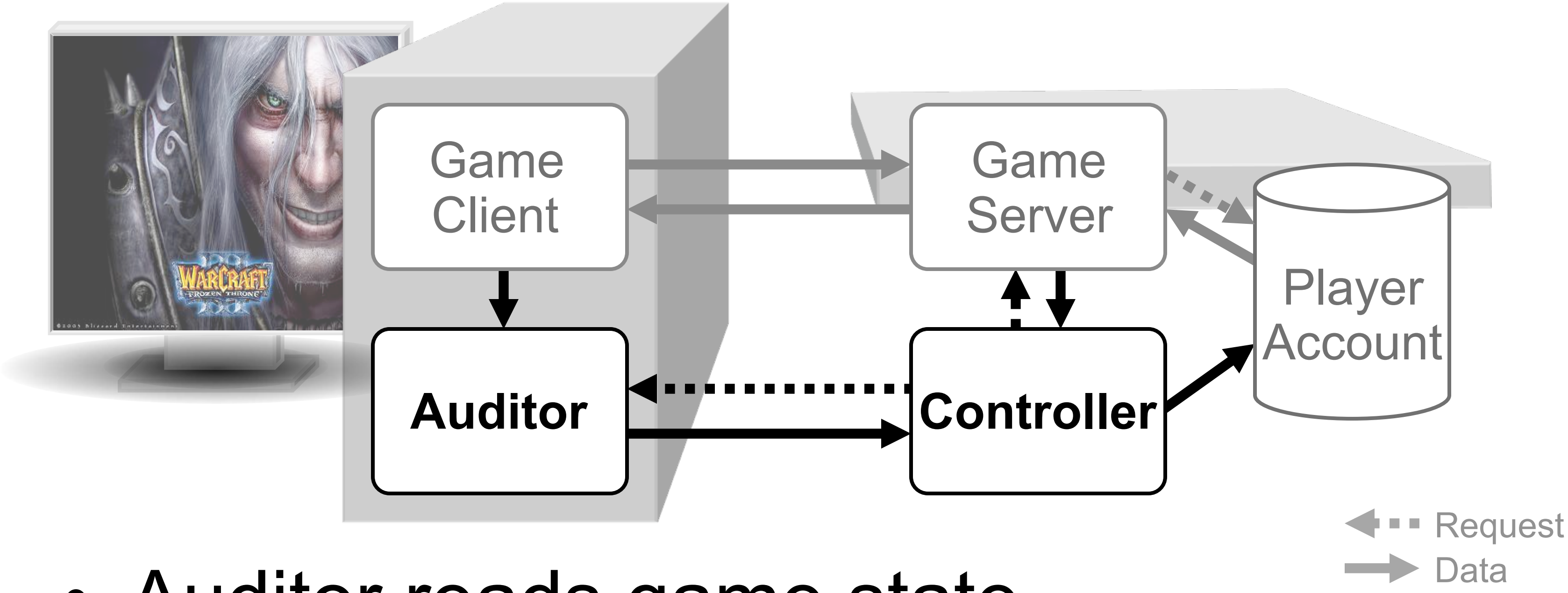
- verify a raw I/O signal corresponds to every mouse and keyboard event

12) Register Monitoring

13) File Integrity

14) Environment Validation

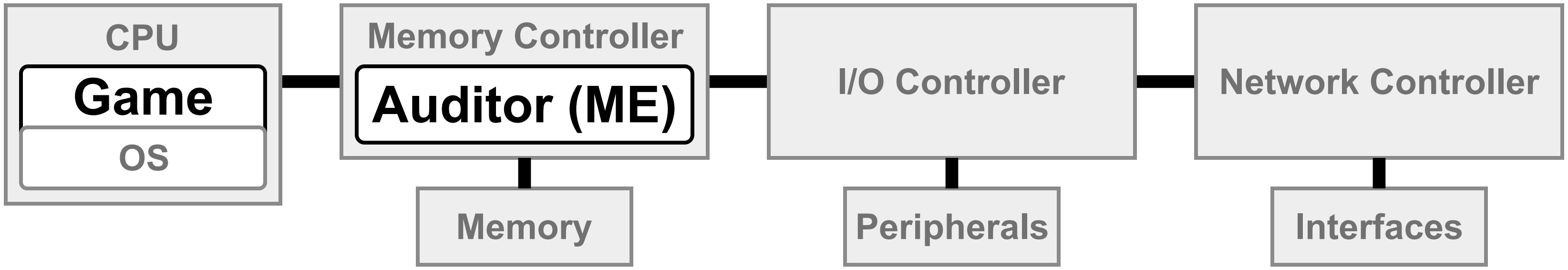
System Architecture



- Auditor reads game state
- Controller decides what & when to sample
  - compares measurement to expected state
  - alters player account when caught cheating

Stealth Measurements

- Run Auditor on Management Engine (ME)
  - safe from software or OS interference
  - can observe memory, I/O, and network traffic
  - secure (authenticated and encrypted) network channel to Controller



Possible Stealth Measurements

Measurement	Cheats Caught
Code Integrity Code Timing	Code Injection Thread Hijacking File Modification
Static Data Integrity	Unauthorized Data Writes File Modification
Scan of Executable Pages Check Execution Range	DLL Injection Code Caves
Stack Validation	Thread Hijacking Thread Injection Direct Game Function Calls Function Pointer Hooks IDTR Tampering
I/O Path Validation	Direct Function Calls

What Could Improve the System?

- Better CPU and memory monitoring

Measurement	**Cheats Caught**
Instruction Counts Interrupt Counts	Code Injection Thread Manipulation File Modification
Register Monitoring	Hardware Facilities
Memory Watch Points	Authorized Data Reads Unauthorized Data Reads Unauthorized Data Writes

Related Work

- Classification
  - J. Yan and B. Randell. A Systematic Classification of Cheating in Online Games. *NetGames* 2005.
  - Y. Lyhyaoui, A. Lyhyaoui, and S. Natkin. Online Games: Categorization of Attacks. *Eurocon* 2005.

- Hardware-based Cheat Detection
  - T. Schluessler, E. Johnson, and S. Goglin. Is a Bot at the Controls – Detecting Input Data Attacks. *NetGames* 2007.

Conclusions

- Cheating is bad
- Cheats are advanced
  - large range of cheat methods
- Detection can catch them
  - many detection methods
- Hardware supports stealth measurements
  - can catch most cheats
- Additional support could catch more cheats

Thanks