Addressing Automated Adversaries of Network Applications

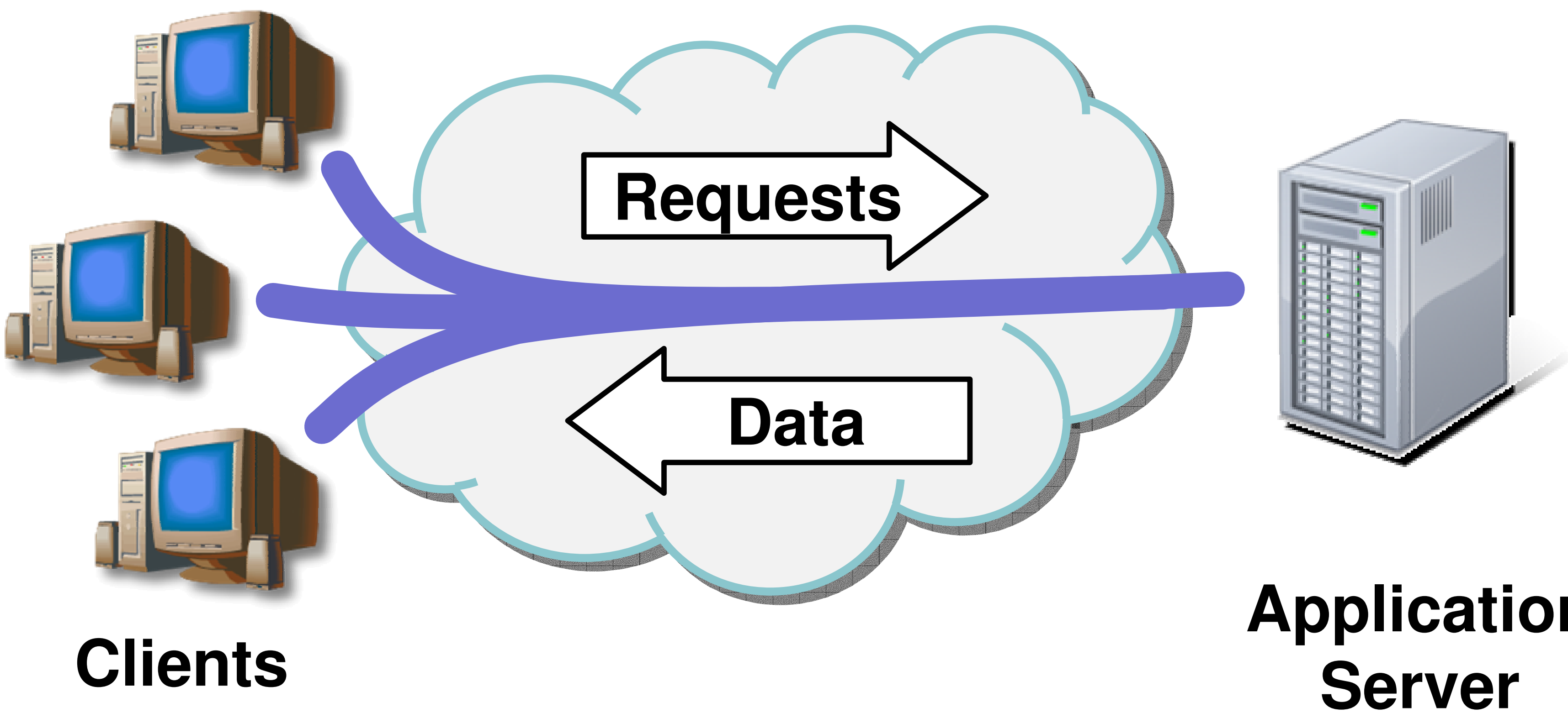
Ed Kaiser

Computer Science Department

Portland State University

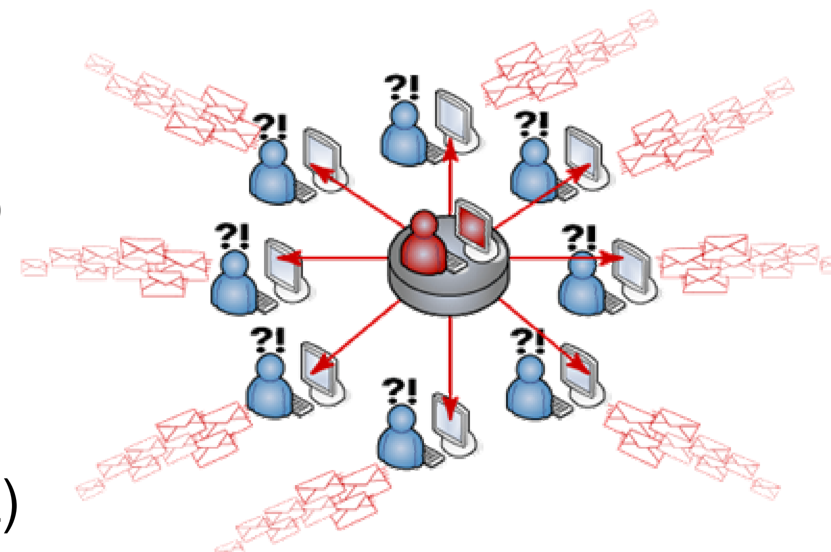
Network Applications

- Internet is patchwork of diverse applications
  - **WWW, email, multimedia, video games**



The Automation Problem

- Adversarial clients employ automation
  - to subvert the service
- Examples include
  - Port Scans
  - Worms (Slammer, Conficker)
  - Denial-of-Service (Georgia)
  - Spam, Comment Spam
  - Click Fraud (Auction Experts)
  - Ticket Bots (Hannah Montana)
  - **Video Game Bots** (WoW Glider)



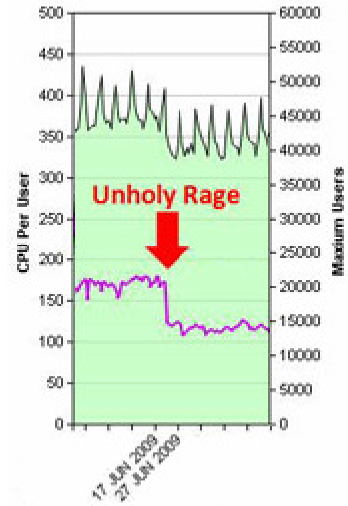
"*Gold Farming*" Example

- Automation software
  - endless repetition
- Cheap foreign labor
  - manage the software
  - respond to moderators

$200 / month expense — **virtual** — **$60k / month profit**

Automation Harms Applications

- **Increased Cost** (e.g., resources)
  - EVE Online anti-farming campaign ban of 2% users led to 30% CPU drop



- **Decreased Efficiency**
  - lower request throughput
  - lower content to noise ratio
- **Denied Accessibility**
  - legitimate users cannot transact with service
- **Tarnished Reputation**
  - rampant cheating in Diablo II
  - online poker cheaters

Thesis Statement

We have new methods to ***detect*** automated behaviors with which an application's service provider can ***identify*** and then ***disincentivize*** automated adversaries.

Research Challenges

1) **<u>Detection</u>**

Advantageous automation must exhibit distinguishing characteristics. What application-specific methods can detect automated behaviors?

2) **<u>Identification</u>**

Detection methods may not be individually conclusive. How can detection methods be combined to most accurately identify automated adversaries?

3) **<u>Dissuasion</u>**

Adversaries react to deterrents. How can we best disincentivize automated adversaries?

Thesis Contributions

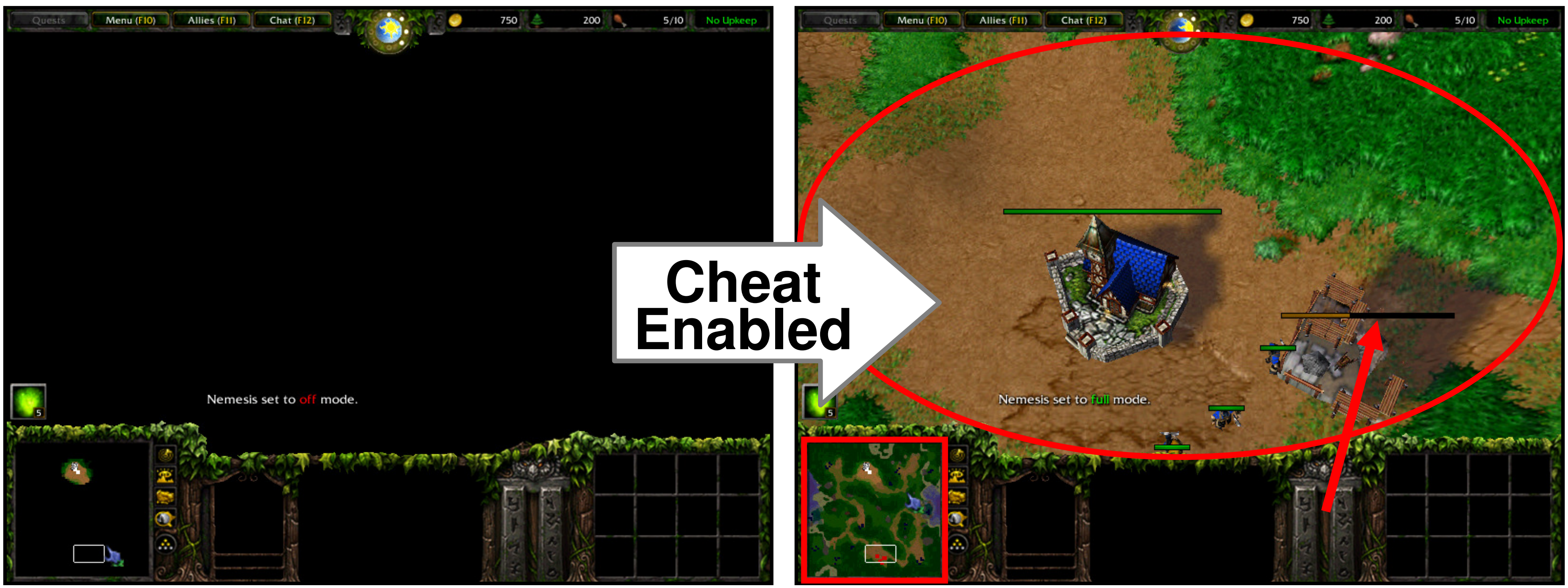
	Detection	Identification	Dissuasion
Fides anomaly-based cheat detection • online video games	★	✓	✓
PlayerRating reputation system • online video games	✓	☆	✓
kaPOW Proof-of-Work system • web-based services	✓	✓	☆

The Cheating Problem

- Networked games simulate complex worlds
  - would like trust only the server but
  - limited server computation
  - player sensitivity to network latency
- Client is ***trusted*** to run simulation locally
  - follow game rules
  - keep secrets from player
- Cheats are software that abuse the trust
  - automate actions a cheater is ***unwilling*** to do
  - accomplish feats a cheater is ***unable*** to do

Nemesis Warcraft III MapHack

- Reveals map and secret opponent locations



What Cheats Modify [NG08]

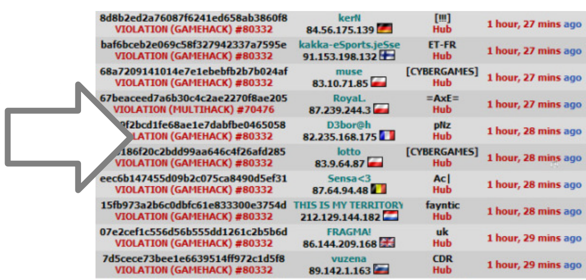
- Game ***memory*** via `WriteProcessMemory()`
  - static data (e.g., gravity constants)
  - dynamic data (e.g., location, health, team)
  - altering existing code (e.g., *hot patch*)
  - injecting new code (e.g., *DLL injection*)
- Game ***execution*** (e.g. automation)
  - thread hijacking (e.g., *detour*, *function hooking*)
  - thread injection via `CreateRemoteThread()`
  - as debugger via `DebugActiveProcess()`

State-of-the-Art in Defense

- Signature-based cheat detection
  - generate cheat-specific signatures
    - must obtain working cheats
    - continual developer effort
    - state grows as new cheats are cataloged
    - does not deal well with polymorphism
  - search ***every*** process for known signatures
    - indiscriminately reads private data (e.g., Blizzard's Warden)
    - prone to false positives:

"**Rifle Aim Prediction:**"

into IRC channel



Tricking Punkbuster

Tricking VAC

Similar to other Security Problems

- Similarity to rootkits
  - adversary controls the machine
    - has administrator privileges
    - runs before anti-cheat software
    - can modify the operating system and other tools
  - uses advanced techniques
    - cloak itself just-in-time (e.g., Hoglund's Supervisor)
    - spoof anti-cheat software results
- Similarity to viruses
  - obfuscation to prevent reversing
  - polymorphism to thwart signature detectors

... yet is Distinct Security Problem

- Adversary is <u>owner</u> of machine
  - yet four mitigating factors…
- Always connected, for long time periods
  - cannot disable server-initiated security (unlike Windows Update)
  - server can perform arbitrary checks on-demand
- Typically targets game code
  - limited places to attack
  - can do anomaly-based detection (à la kernel integrity approach)
- Presence is not immediately catastrophic
  - can wait to take action (to prevent cheater from learning)
  - machine is not used to attack network hosts
  - damage can be rolled back easily
    - unlike reissuing stolen credit card numbers, SSNs, etc.
- Monetary penalty for being caught
  - $50 game copy, $10/month, plus time lost (opposed to botnet)

The Fides Approach [CCS09]

- Approach leverages properties of problem
- ***Anomaly-based*** cheat detection
  - know what game looks like
    - finite state
    - readily available
  - search the game client for deviations
    - cheater targets game code
    - cheat agnostic
  - detection is sufficient
    - cheat is not immediately catastrophic

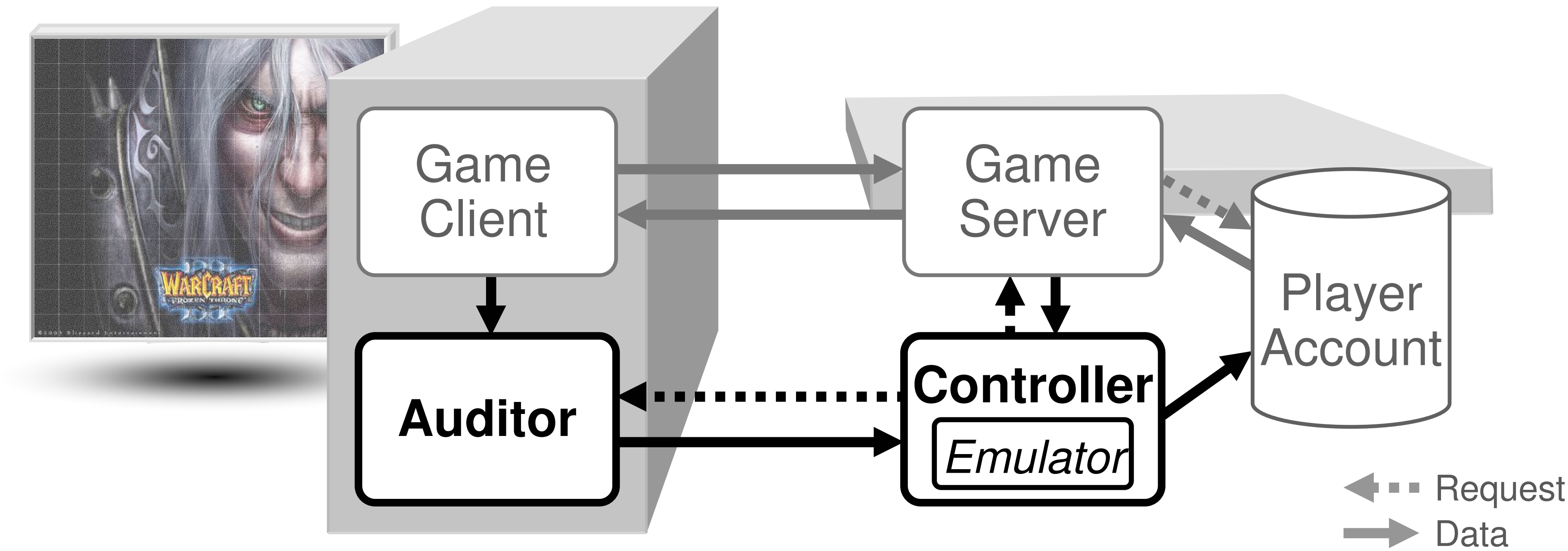
Fides Approach cont'd

- Via ***continued random remote measurement***
  - continued
    - not done only once at startup
    - server has indefinite contact with the client
    - can audit until absolutely confident in results
  - random
    - no need for complete integrity check
    - conceals what will next be measured and when, instilling "fear-of-the-unknown"
  - remote
    - don't trust the client to make judge its own integrity

Fides Approach cont’d

- Using ***partial client emulation***
  - to accommodate client system variation
    - between players
    - between sessions (e.g., desktop vs. laptop)
  - regarding libraries, versions, and locations
  - always connected, for long time periods

Fides Architecture



- Controller decides ***how***, ***what***, and ***when*** to measure the client
  - compares measurement to emulated state
  - alters player account when caught cheating
- Auditor ***only measures*** game client process

Limitations to Approach

- Software Auditor will be target of attack
  - supply it legitimate client process-state data
    - statically generated ahead of time
    - dynamically generated by second unmodified client
  - hook it to know when to unload cheat

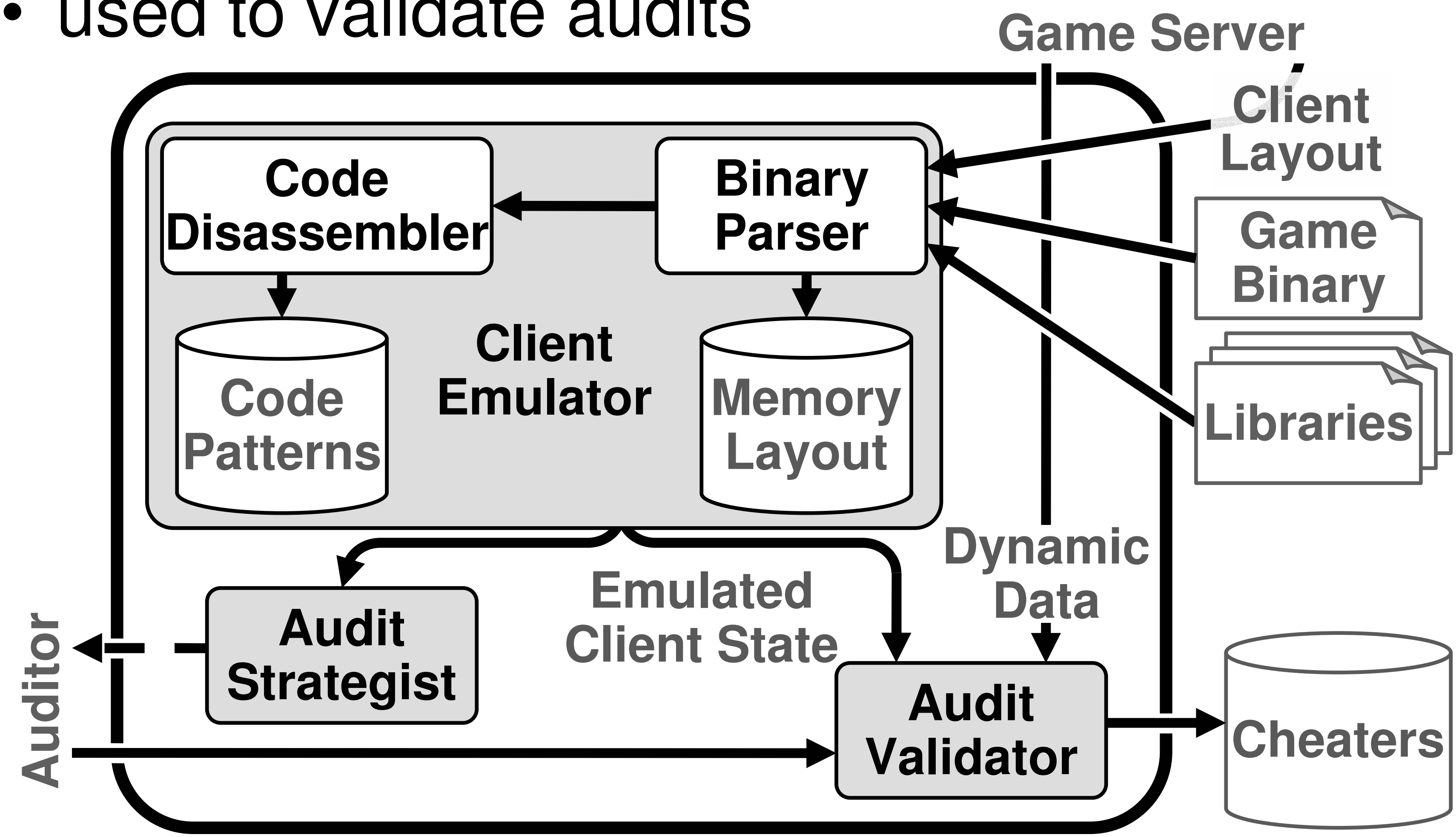
- Cannot catch cheats external to game client
  - collusion cheats
    (e.g., online poker cheaters)
  - robotic cheats
    (e.g., Guitar Hero robot,
    in-network cheats)

Addressing the Limitations

- Auditor is the weak point
  - cryptographically entangle the Auditor
    - similar to Pioneer approach
  - verify correct Auditor operation
    - similar to Intel anti-virus presence detector
  - run the Auditor itself on secure hardware
    - (e.g., the Intel AMT Manageabilty Engine)
    - similar to Copilot
- Does not detect completely external cheats
  - anomaly-based detection on user behavior or statistics available to server

Controller Design

- Emulates client-process state
  - drives audit strategist (could be game-specific)
  - used to validate audits

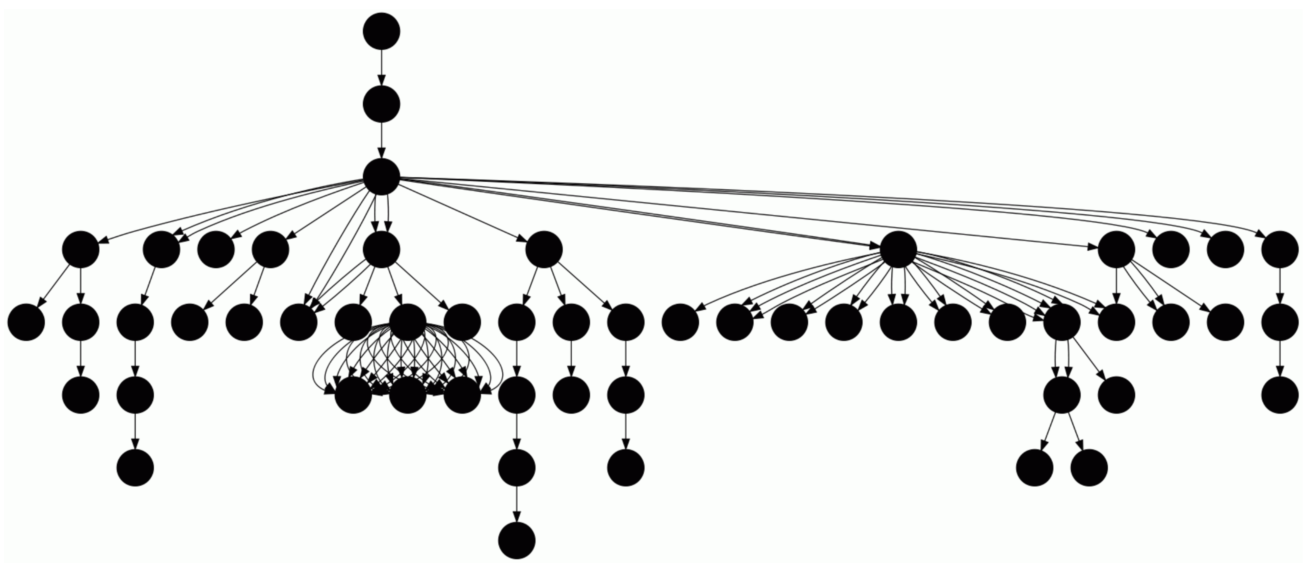


Partial Client Emulation

- Done once at client login
  - share library names, versions and locations
- Works on ***any*** commercial-off-the-shelf game
- Binary Parser recreates client layout
  - identifies and hashes static code & data
    - high confidence in client understanding
  - identifies dynamic data regions
    - more expensive (i.e., game-specific) to validate
    - high confidence in client understanding

Partial Client Emulation cont'd

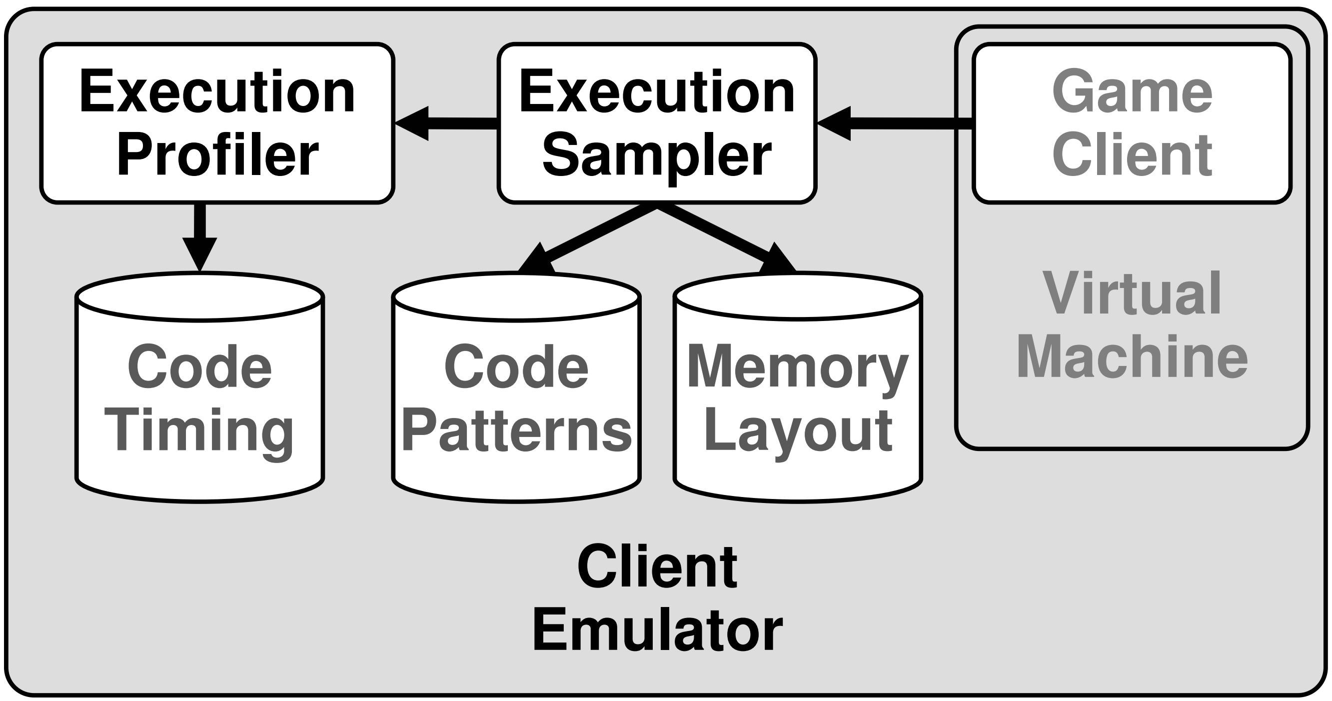
- Code Disassembler
  - creates a rough call graph
    - learns instruction range for each function
    - learns **`CALL`** addresses (i.e., relating functions)
    - lower confidence (difficult to get complete coverage)



Partial Call Graph of a Homebrew game

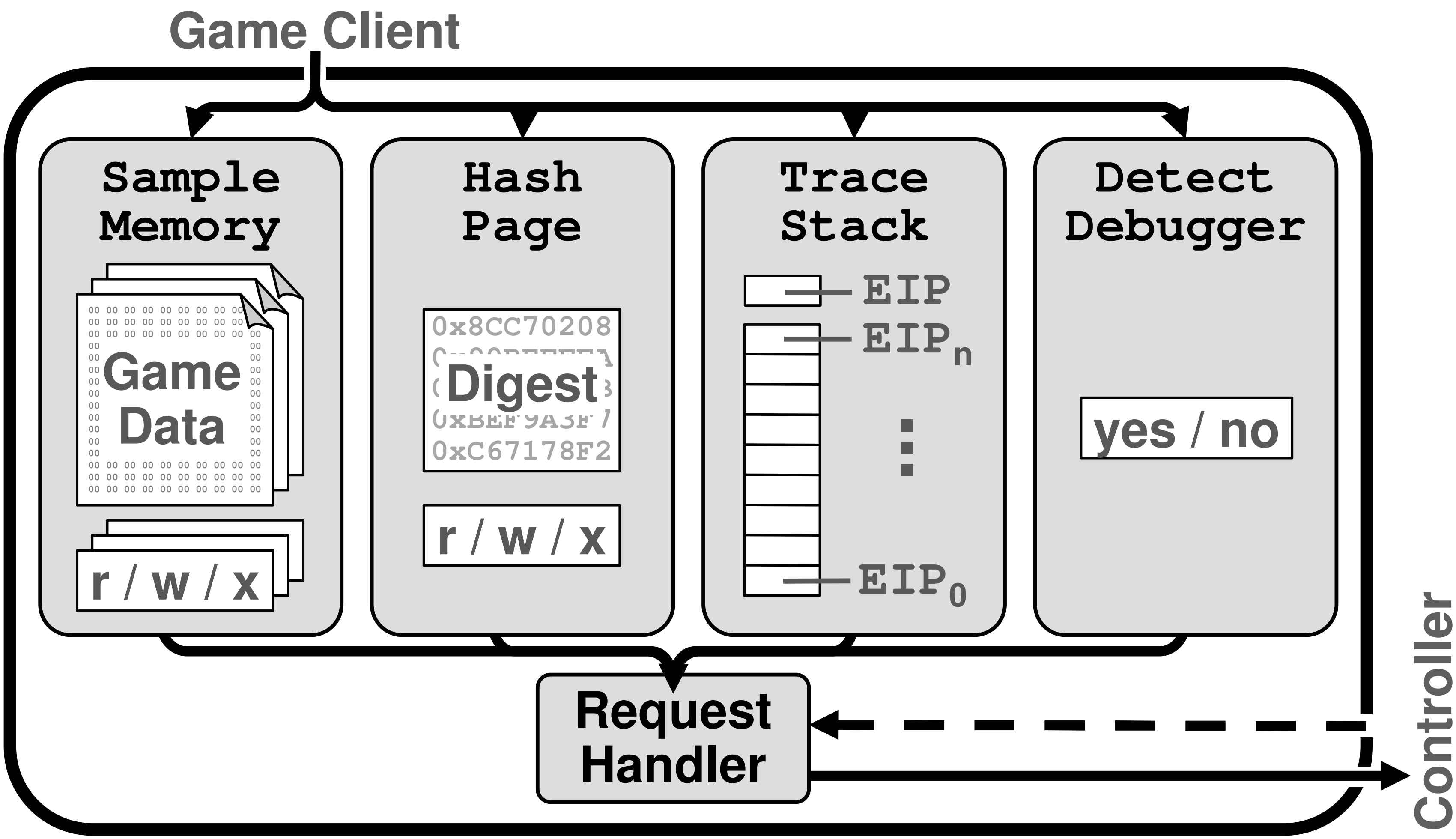
Partial Client Emulation cont'd

- Execution Sampler and Execution Profiler
  - run game with identical layout to client
  - learn dynamic calls not obtainable through static analysis



Auditor Design

- Measures the client process
  - returns the data to the controller



Auditor Measurements

Cheat Methods best detected by the Auditor Measurements

Measurement	Cheat Method
`Sample Memory`	Dynamic Data Manipulation
`Hash Page`	Code Manipulation
	Code Injection
	Static Data Manipulation
	File Replacement
`Trace Stack`	Thread Injection
	Thread Hijacking
	Function Pointer Hooking
	Direct Function Calls
`Detect Debugger`	Software / Hardware Debugging

Evaluation

- Implemented Auditor & Controller in C++
  - running on separate 2.39GHz Intel Core2
  - on commercial-off-the-shelf game Warcraft III
- Experiments answer:

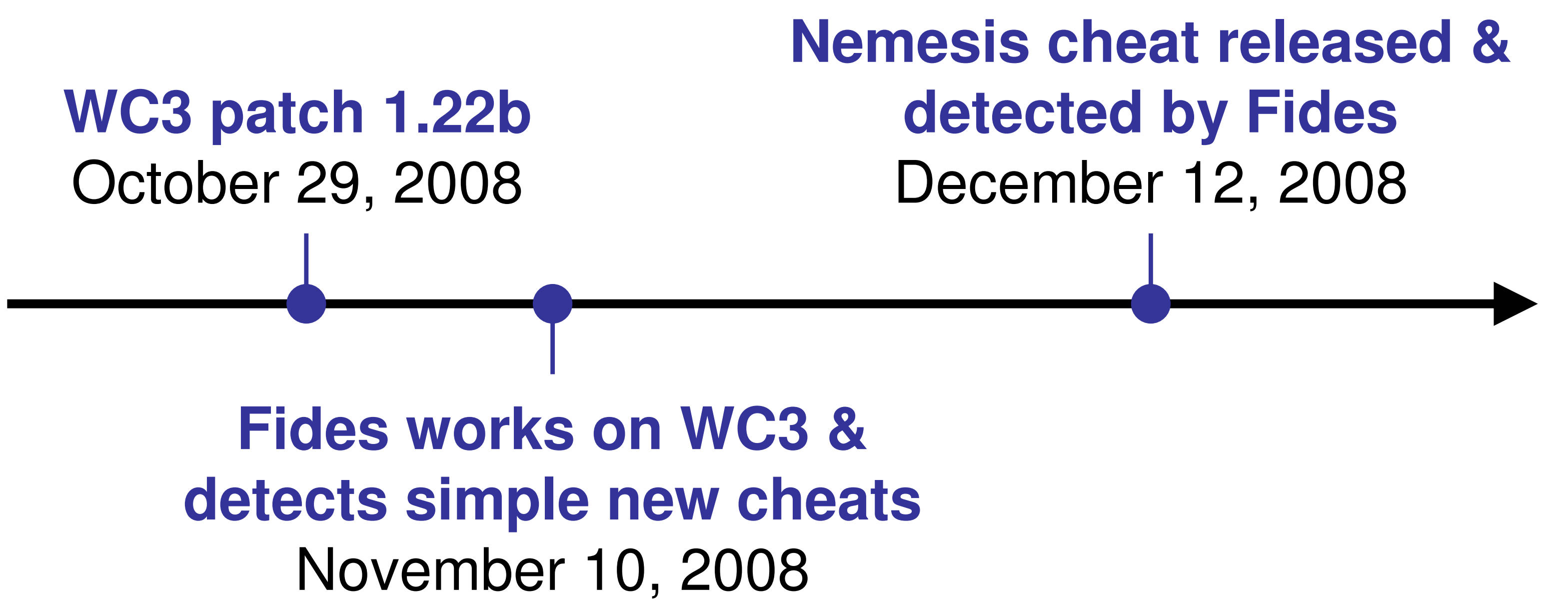
  "Does Fides work?"

  "Does Fides operate quickly?"

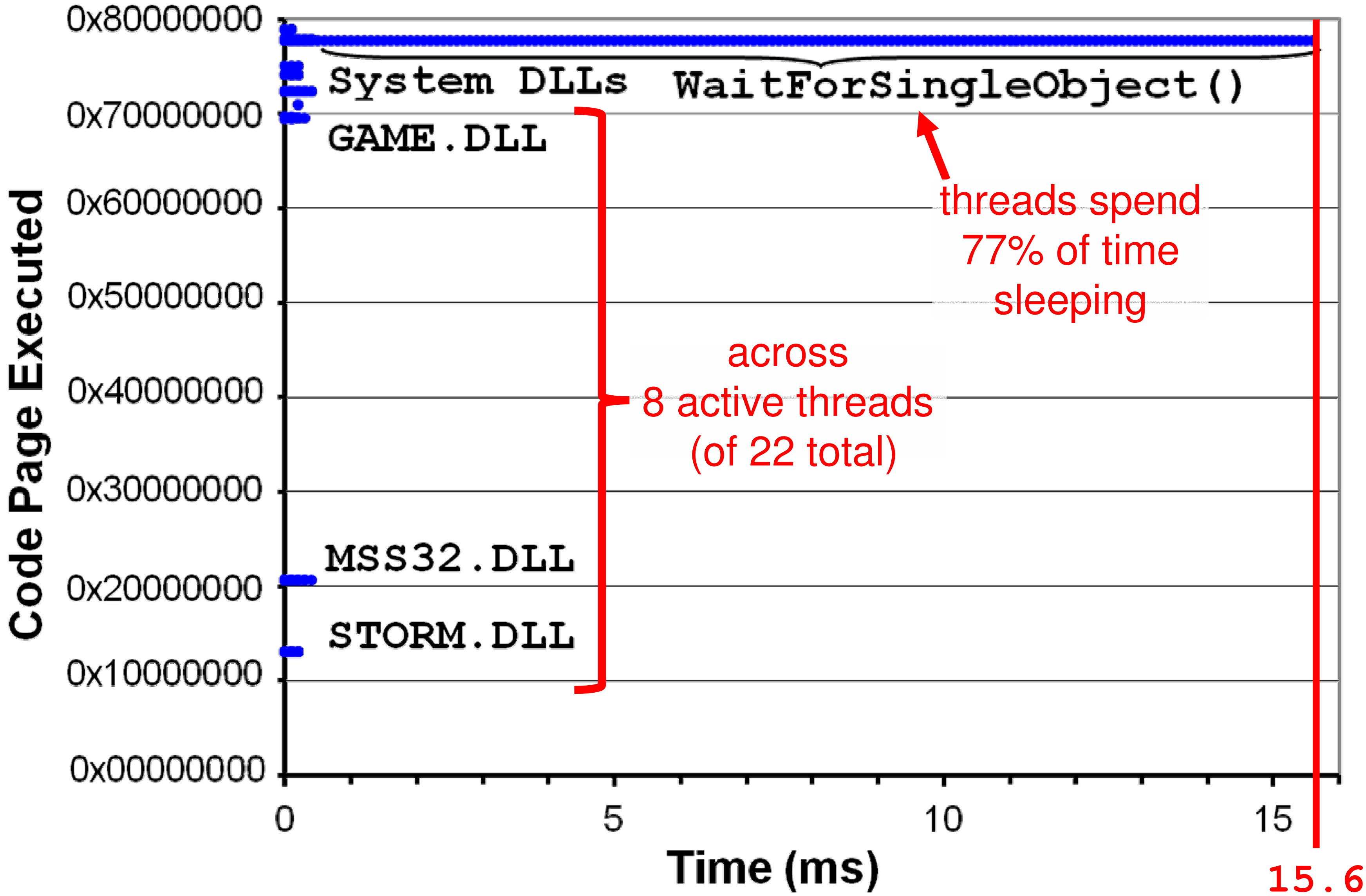
  "Will Fides scale well?"

Evaluation Timeline

- Learn newly patched game
  - account for security (i.e., obfuscation & anti-debugging)
- Acquire and run first-to-release cheats
  - verify correct operation
- Detection!



Warcraft III Execution Profile



Experiment: Functionality

- Ran the following cheats:
  - Bendik's MH, NOPs a few bytes
  - Kolkoo's MH, NOPs bytes over more pages
  - Revealer MH, NOPs and hooks input functions
  - Simple MH, NOPs bytes over many pages
  - Nemesis MH, complex and "undetectable"

complexity ↓

- Periodically audit (±5% randomness)
- Code page hash audit the game
  - hash currently executed code page
- Measure the mean audits required to detect
  - averaged over 1000 trials

Results

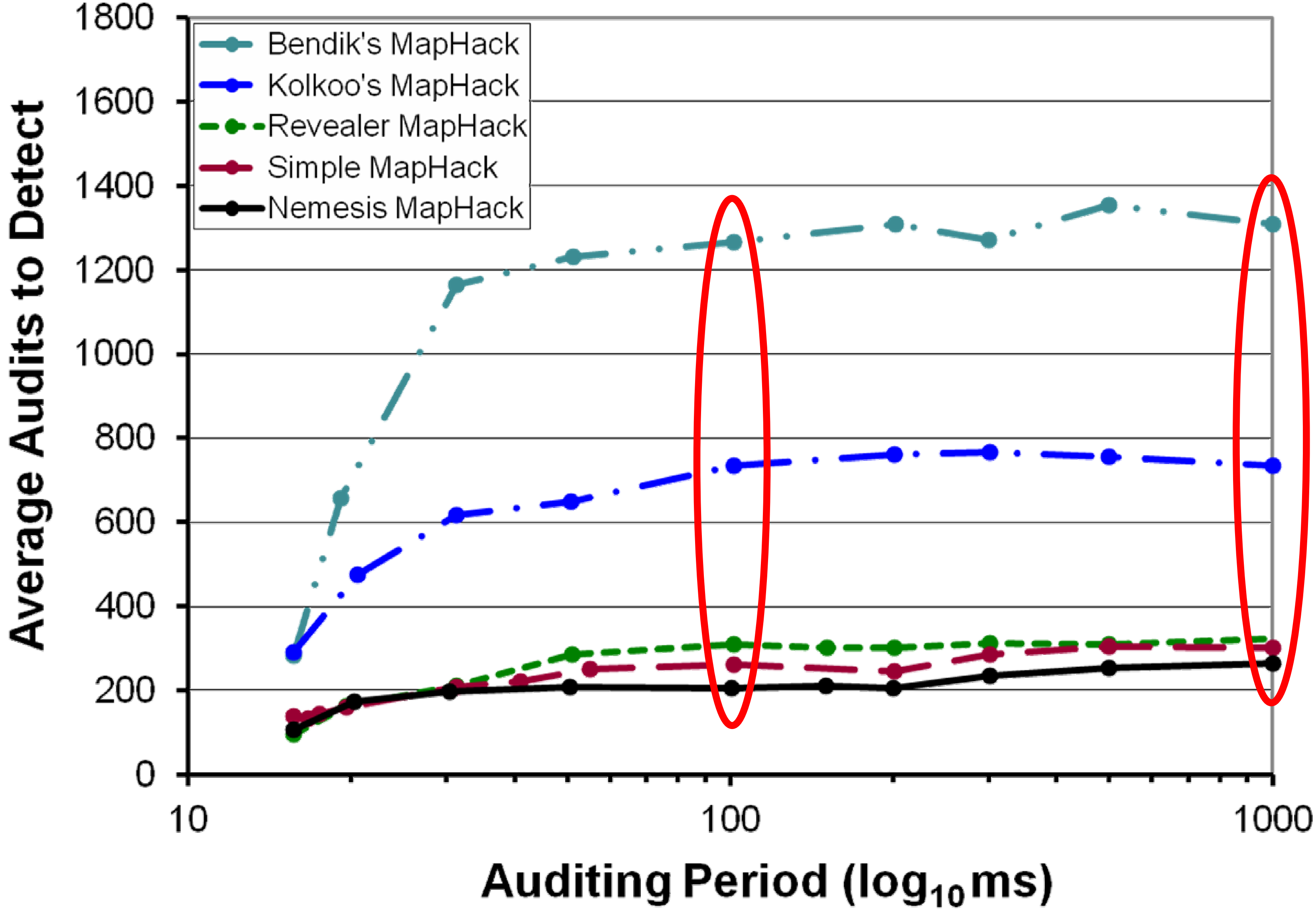
- Auditing roughly once every 100ms

Cheat	Avg Audits Required	Wall-clock Time
Bendik's MH	1265.3	2min 08.4s
Kolkoo's MH	733.4	1min 14.5s
Revealer MH	309.8	31.4s
Simple MH	260.4	26.4s
Nemesis MH	204.1	20.5s

- Auditing roughly once every second

Cheat	Avg Audits To Detect	Wall-clock Time
Bendik's MH	1309.7	21min 49.7s
Kolkoo's MH	733.0	12min 13.0s
Revealer MH	322.2	5min 22.2s
Simple MH	301.3	5min 01.3s
Nemesis MH	264.1	4min 24.1s

Detecting Warcraft III MapHacks



Observations

- Audits required to detect
  - complex cheats require fewer audits
    - make more modifications
    - easier to detect randomly
  - generally starts low
    - when sampling faster than game input loop, audits encounter more infrequently executed pages
  - asymptotically levels off
    - when sampling much slower than game input loop, each audit becomes independent random sample

Experiment: Efficiency

- Benchmarked routines by measuring cycles
  - using **`RDTSC`** register

	Task	Cycles	Time
Auditor	`Sample Memory`	38,000	**15.9μs**
	`Hash Page`	113,000	**47.2μs**
	`Trace Stack`	64,400	**26.9μs**
	`Detect Debugger`	23,200,000	**9.7ms**
Controller	Validate Hash	3,170	**1.3μs**
	Validate Stack	10,800,000	**4.5ms**
	Validate Debugger	130	**52.4ns**
	Parse All Binaries	236,000,000	**98.8ms**
	Disassemble Code	205,000,000	**85.9ms**

Experiment: Scalability

- Can Fides exploit common memory layout to reduce emulation replication?
  - in terms of memory
  - and computation
- On non-ASLR systems, libraries are loaded to consistent pre-determined locations

Warcraft III Memory Allocation

- Ran game on two different XP machines
  - 1000 trials on each (2000 total)
  - memory section is one or more 4KB pages
    - executable → code
    - writable → dynamic data
    - only readable → static data

Type	Memory	Similarity		
		Client A	Client B	Both
Code	28.7MB (1.4%)	100.0%	96.2%	90.8%
Static Data	20.5MB (1.0%)	98.8%	94.6%	87.2%
Dynamic Data	71.7MB (3.5%)	*29.5%*	*55.6%*	*11.3%*
Reserved	69.6MB (3.5%)	*64.6%*	*93.5%*	*49.4%*
Unallocated	1.9GB (90.6%)	--	--	--

Fides Summary

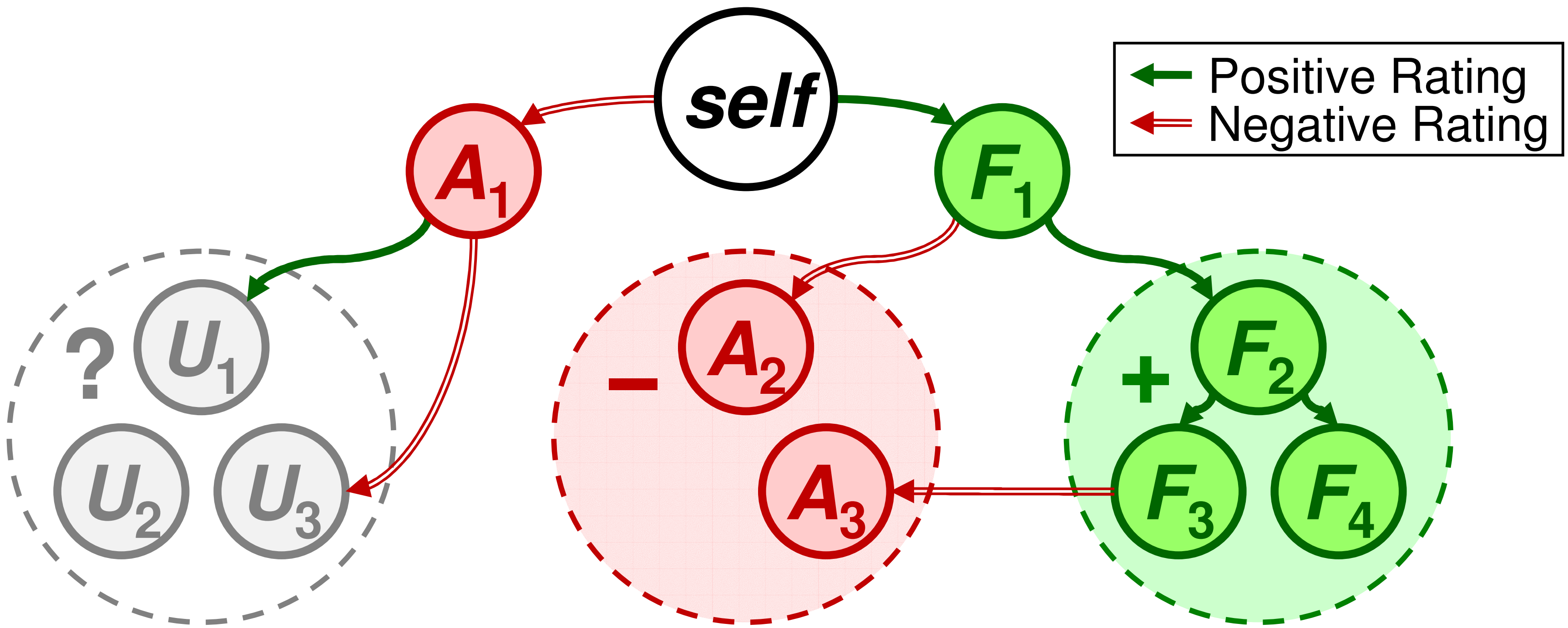
- Cheats are advanced
  - large range of cheat methods
  - present a distinct security problem
- **Fides** is specifically designed to detect them
  - ***anomaly-based*** detection
  - via ***continued random remote measurements***
  - using ***partial client emulation***

Other Thesis Contributions

	Detection	Identification	Dissuasion
Fides anomaly-based cheat detection • online video games	★	✓	✓
PlayerRating reputation system • online video games	✓	★	✓
kaPOW Proof-of-Work system • web-based services	✓	✓	★

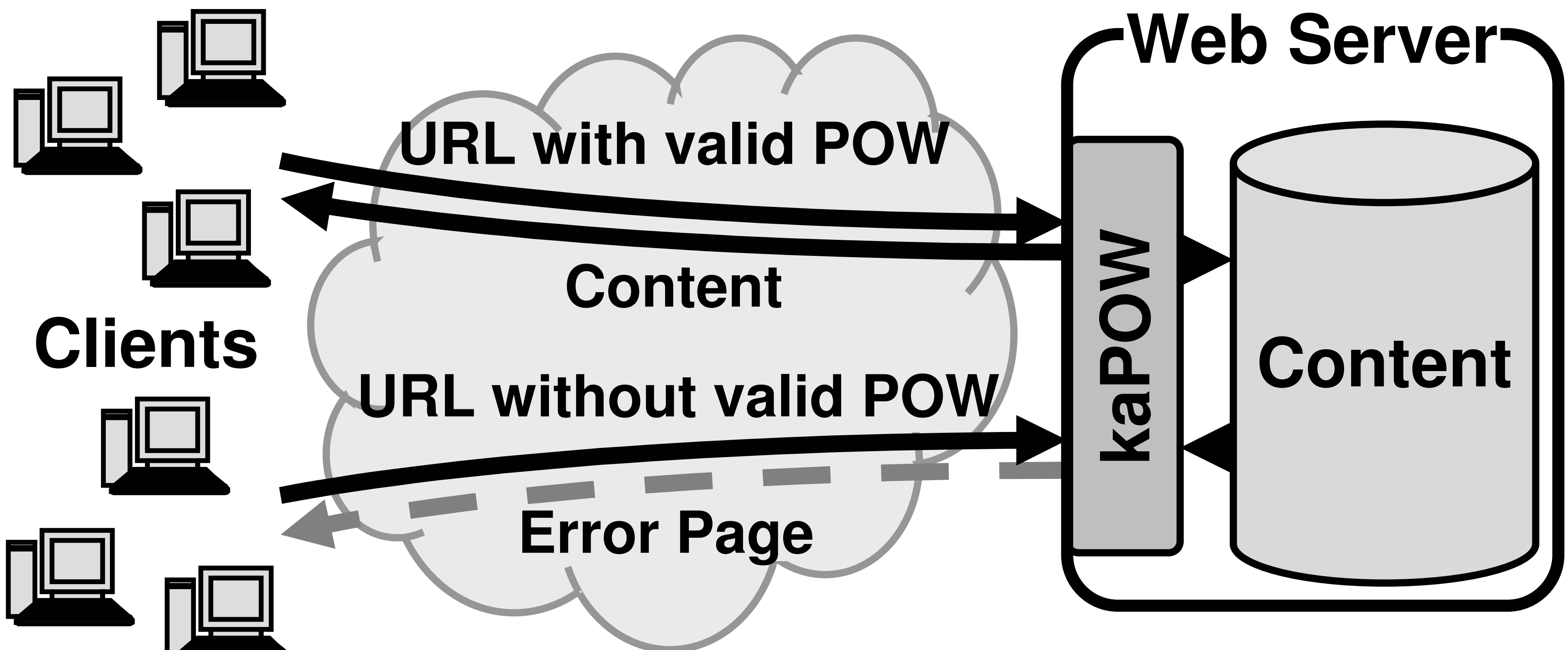
PlayerRating System [NG09]

- A peer-to-peer reputation system
  - treat peers as homogenous detectors
  - can facilitate other information sources
  - positive trust is transitive



kaPOW System [GI08, GI10]

- Transparent Proof-of-Work
  - computationally tax malicious/automated clients
  - geographic location as an automation indicator



Conclusion

- Thesis statement:

  We have new methods to ***detect*** automated behaviors with which an application's service provider can ***identify*** and then ***disincentivize*** automated adversaries.

- Thesis validated:
  - ✓ Explored detection methods
  - ✓ Built a detector aggregator
  - ✓ Penalized automated adversaries

Thanks

Associated Peer-Reviewed Publications

[GI10] E. Kaiser, W. Feng. "Helping TicketMaster: Changing the Economics of Ticket Robots with Geographic Proof-of-Work." In *Global Internet,* March 2010.

[NG09] E. Kaiser, W. Feng. "PlayerRating: A Reputation System for Multiplayer Online Games." In *NetGames,* November 2009.

[CCS09] E. Kaiser, W. Feng, and T. Schluessler. "Fides: Remote Anomaly-Based Cheat Detection." In *ACM CCS*, November 2009.

[NG08] W. Feng, E. Kaiser, and T. Schluessler. "Stealth Measurements for Cheat Detection in On-line Games." In *NetGames*, October 2008.

[GI08] E. Kaiser and W. Feng. "mod_kaPOW: Protecting the Web with Transparent Proof-of-Work." In *Global Internet,* March 2008.

[GI07] W. Feng and E. Kaiser. "The Case for Public Work." In *Global Internet*, April 2007.

[IC05] W. Feng, E. Kaiser, W. Feng and A. Luu. "The Design and Implementation of Network Puzzles." In *IEEE INFOCOM*, March 2005.