Fides: Remote Anomaly-Based Cheat Detection Using Client Emulation

Ed Kaiser
Wu-chang Feng

Portland State University

Travis Schluessler

intel

The Cheating Problem

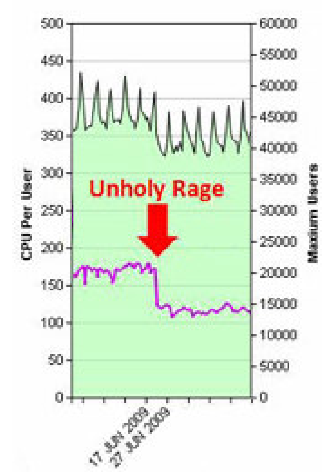
- Networked games simulate complex worlds
  - limited server computation
  - player sensitivity to network latency
- Client is ***trusted*** to run simulation locally
  - follow game rules
  - keep secrets from player
- **Cheats are software that abuse the trust**
  - accomplish feats a cheater is ***unable***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

Cheating Affects Online Games

- Frustrates legitimate players
  - distorts in-game economy
  - not fun to play against cheaters
  - cannot tell if good opponents are legitimate
    - good players get accused of cheating

Game Experience May Change During Online Play

- Impacts profitability of game developer
  - existing players may quit in frustration
  - bad reputation dissuades new players
  - increases operating costs
    - EVE online: 2% bots → 30% of load



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***
  - thread hijacking (e.g., *detour*, *function hooking*)
  - thread injection via **`CreateRemoteThread()`**
  - as debugger via **`DebugActiveProcess()`**

Nemesis Warcraft III MapHack

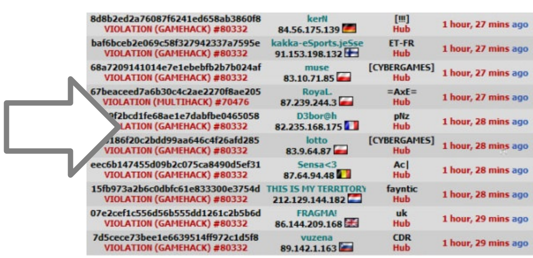
- Reveals map and secret opponent locations
  - inject DLL containing cheat code
  - detours rendering functions
  - hooks I/O handlers for toggling operation
  - calls game message functions to display status

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 ***<u>every</u>*** 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
- 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

- 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

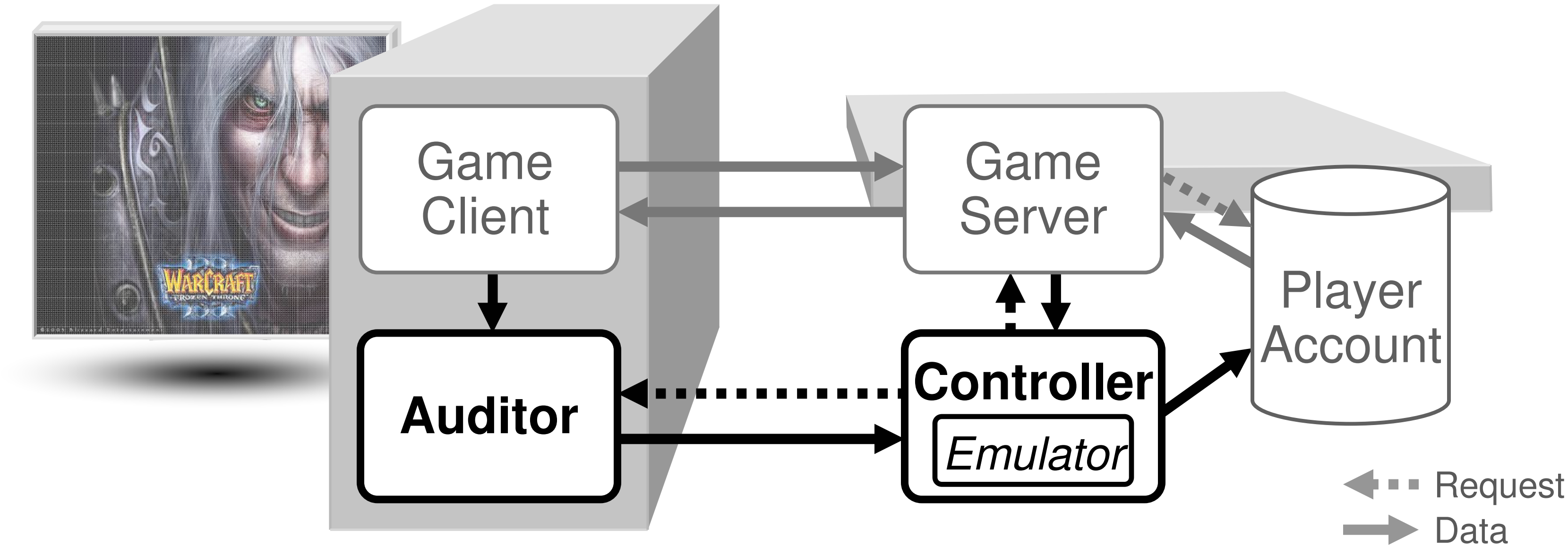
Fides Approach cont'd

- Via ***<u>continued random remote measurement</u>***
  - unpredictable
    - conceals what will next be measured and when, instilling "fear-of-the-unknown"
  - probabilistic
    - no need for complete integrity check
    - server has indefinite contact with the client
  - continued
    - can audit until absolutely confident in detection

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 ***what*** & ***when*** (and eventually ***how***) to measure the client
  - compares measurement to emulated state
  - alters player account when caught cheating
- Auditor ***only reads*** game process state

Limitations to Approach

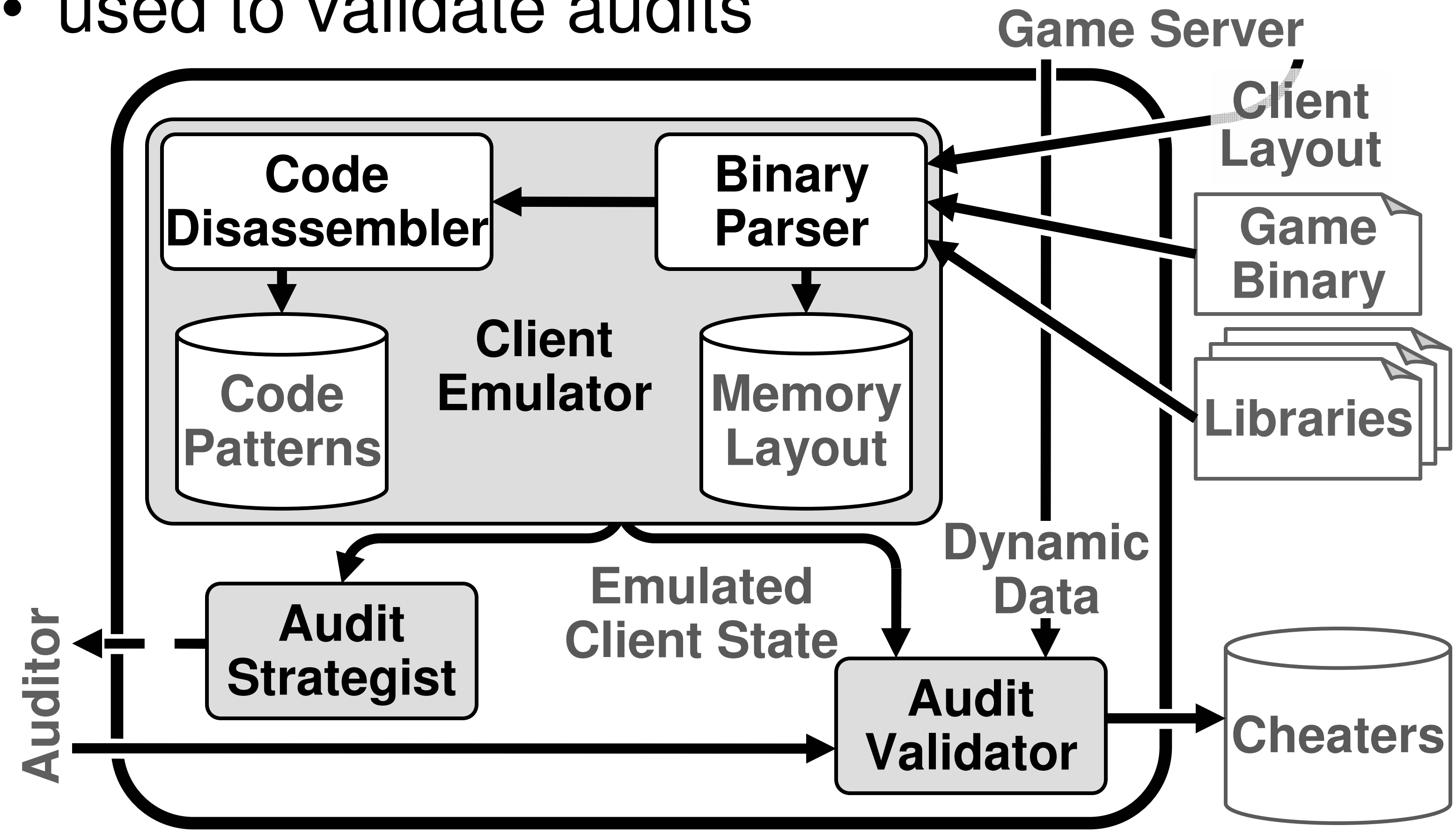
- As a software implementation
  - Auditor will become target of attack
    - feed it known legitimate client process-state data
      - statically generated (pre-sampled and stored in database)
      - dynamically generated (running second unmodified client)
    - hook it to know when to unload cheat
- Cannot catch cheats external to game client
  - collusion cheats
    (e.g., online poker cheaters)
  - robotic cheats
    (e.g., Guitar Hero robot)

Addressing the Limitations

- Auditor is the weak point
  - cryptographically entangle the Auditor
    - similar approach to Pioneer
  - use secure hardware to verify 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

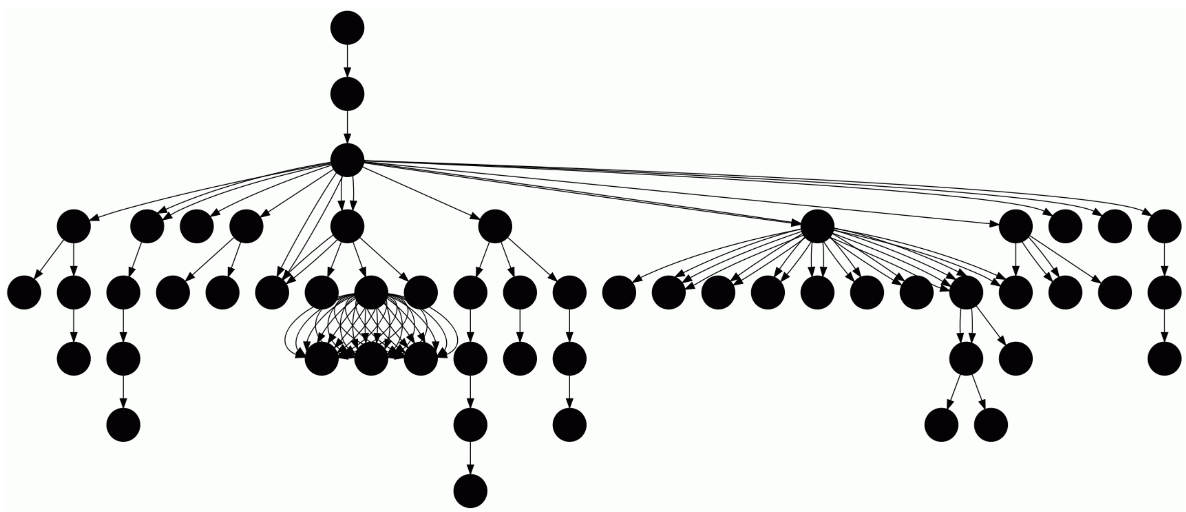


Partial Client Emulation

- Done once at client login
  - share library names, versions and locations
- Works on <u>*any*</u> commercial-off-the-shelf game
- Binary Parser recreates client layout
  - uses libraries/executable stored at Controller
  - uses client layout (base locations) to map them
  - 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)



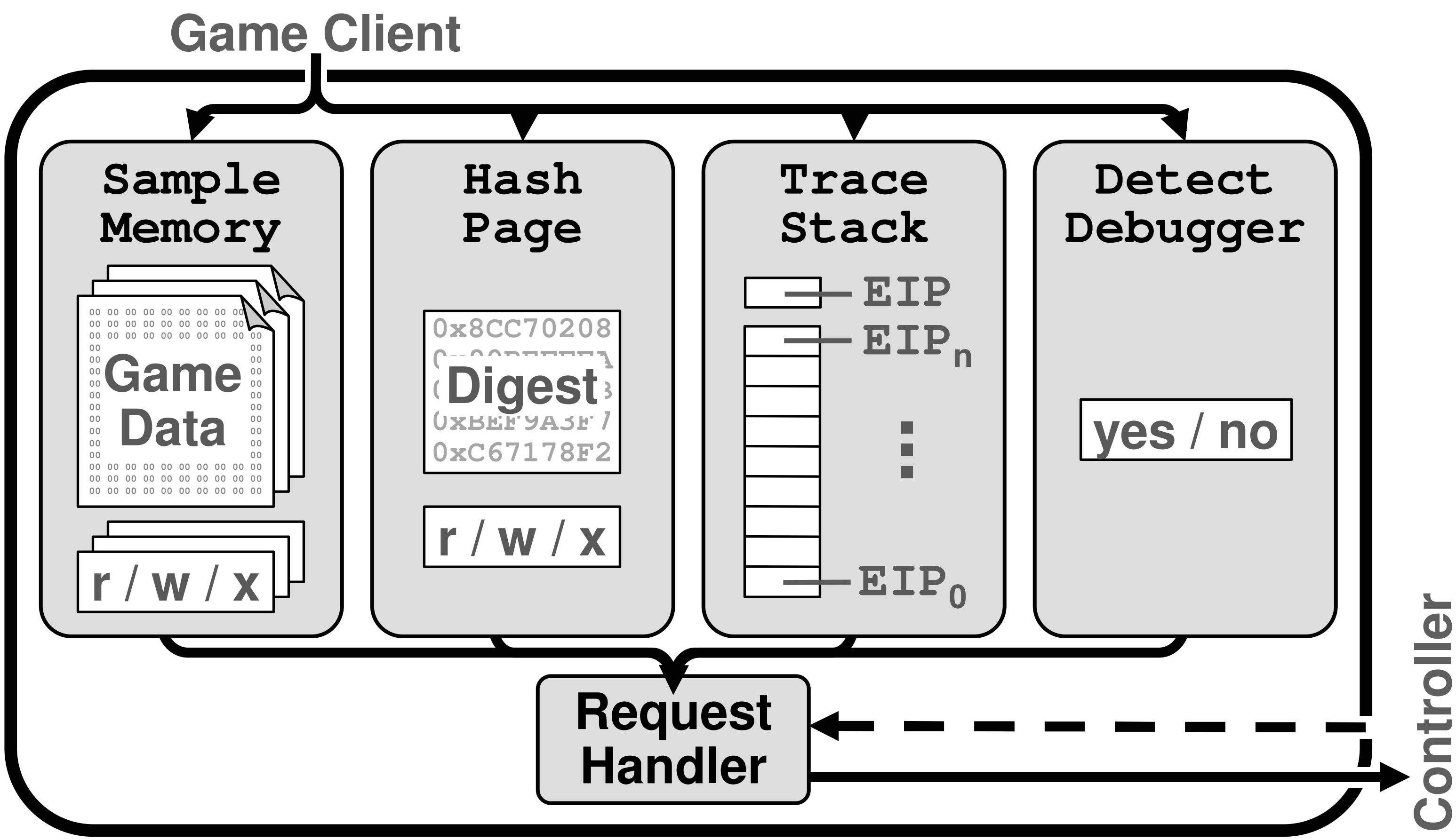
Partial Call Graph of a Homebrew game

Partial Client Emulation cont'd

- Execution Sampler and Execution Profiler
  - run game in VM with identical layout to client
  - learn dynamic calls not obtainable through static analysis
    - use hardware debugging techniques to single-step through execution

Auditor Design

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



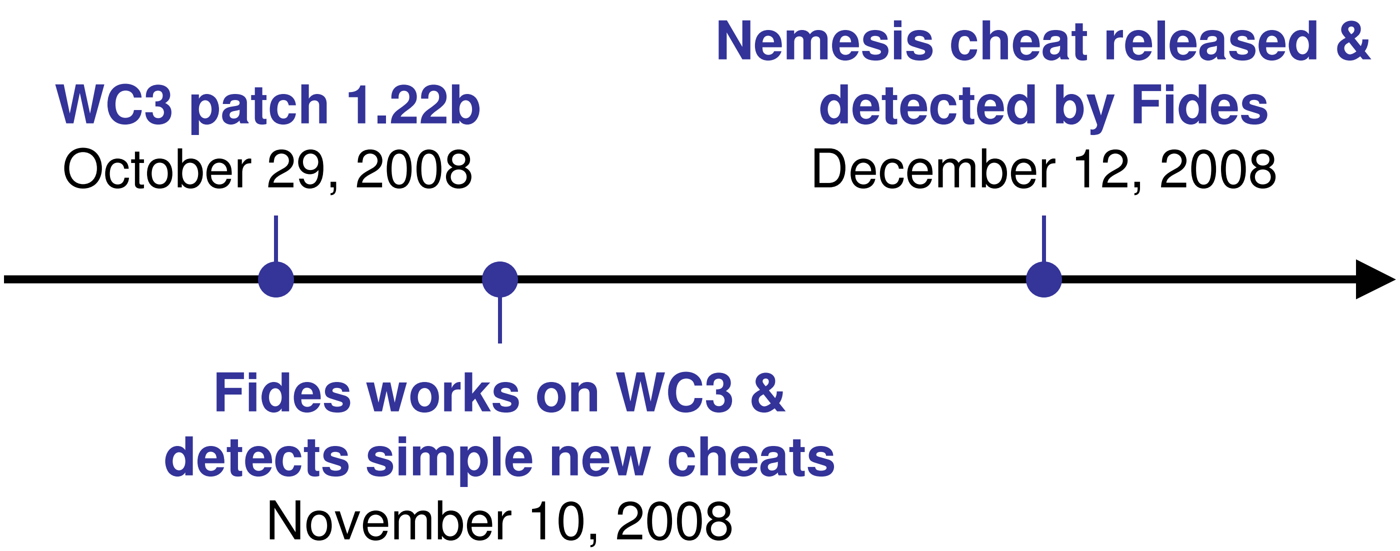
Evaluation

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

	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	Parse All Binaries	236,000,000	**98.8ms**
	Disassemble Code	205,000,000	**85.9ms**
	Validate Hash	3,170	**1.3μs**
	Validate Stack	10,800,000	**4.5ms**
	Validate Debugger	130	**52.4ns**

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!



Experiment Setup

- Ran the following cheats:

complexity

  - 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”
- 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

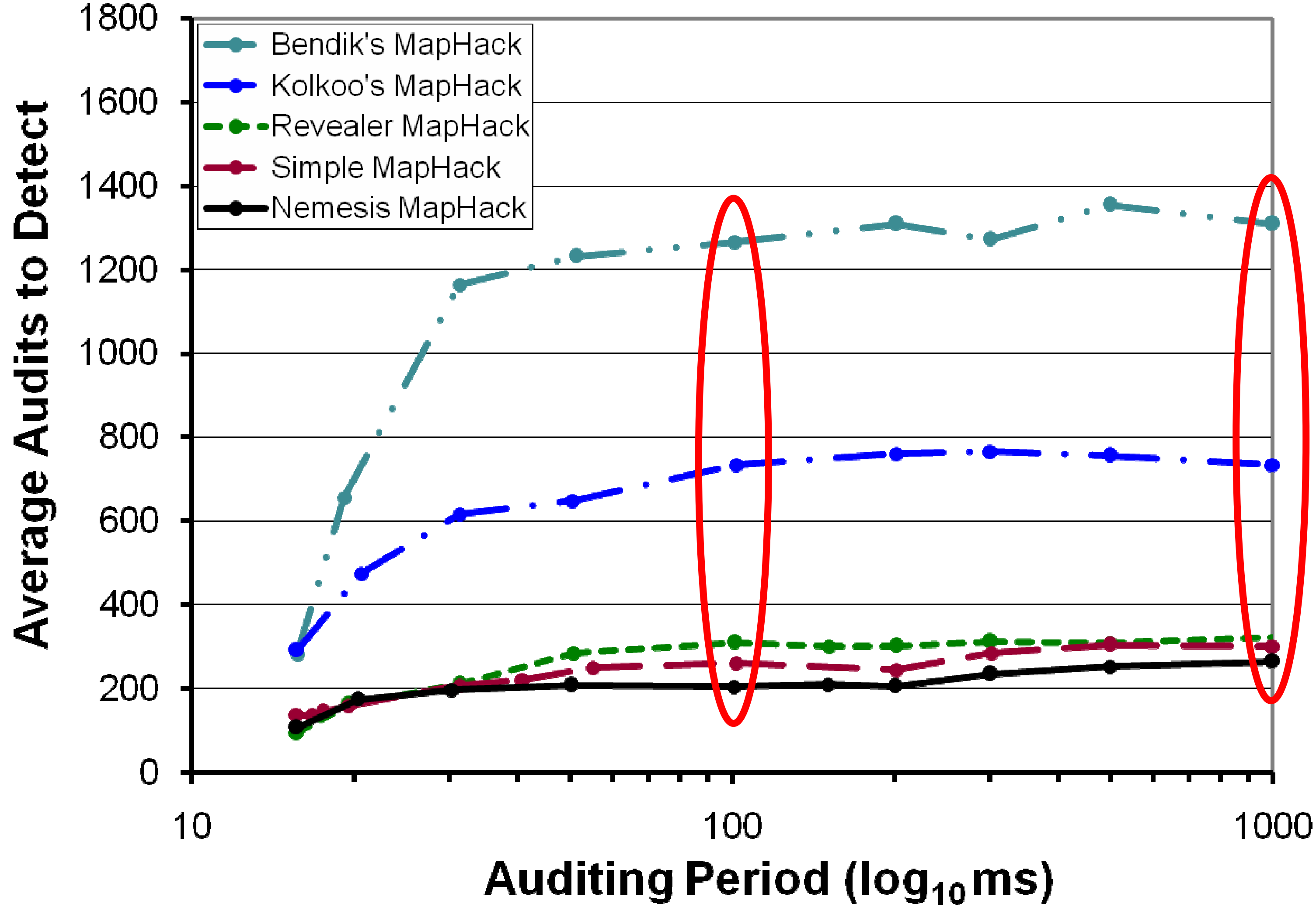
- Auditing roughly once every 100ms

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



Conclusions

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

Thanks