What Can Games Learn From Security Research?

Wu-chang Feng

Portland State University
On-line Games

• A multi-billion dollar industry
  – Some cash cows

Warcraft/Starcraft
Half-Life/Counter-Strike
World of Warcraft
Age of Empires
Battlefield
Lineage
Cheating

• Achilles heel of the gaming platform
  – Causes legitimate players to quit
  – Creates bad word-of-mouth to discourage new players
  – Wrecks virtual economies
Hacking

• Achilles heel of the Internet platform
  – Causes legitimate users to lose productivity
  – Creates bad word-of-mouth to discourage new uses
  – Wrecks real economies and business models
Observation

- A lot of research and effort has been expended solving the hacking problem
- R&D from the computer security field may be helpful in preventing cheating in on-line games
Goal

• Find and apply security research in novel ways to games
• 3 projects
  – Using cryptographic protocols to prevent cheating
    • Mitigating information exposure in RTS games
  – Applying platform integrity tools to detect cheating
    • Tamper-resistant, hardware-based game measurements
  – Creating new game architectures using cryptographic primitives
    • Public-server MMORPGs
Goal

• Find and apply security research in novel ways to games
• 3 projects
  – Using cryptographic protocols to prevent cheating
    • Mitigating information exposure in RTS games
  – Applying platform integrity tools to detect cheating
    • Tamper-resistant, hardware-based game measurements
  – Creating new game architectures using cryptographic primitives
    • Public-server MMORPGs
Background

• Information exposure cheats
  – Display information player should not see
  – Server or peer sends complete information to client
    • Other player locations
    • Chest contents
    • Often due to performance reasons
  – Client code expected to hide information that is not supposed to be seen by player
  – Cheat reveals secret information to player
Examples of information exposure

- Wallhack in FPS games
  - Quake 4 – released 10/18/2005
  - Call of Duty 2 – released 10/25/2005
  - Cheats for both in 11/2005
Examples of information exposure

- Maphack/Chesthack in MMORPGs
  - ShowEQ for EverQuest
  - Display all entities and equipment on a map
Examples of information exposure

• Maphack in RTS games
  – Warcraft3
Examples of information exposure

- Maphack in RTS games
  - Warcraft3 with Maphack
  - Reveal map and enemy units
Solving information exposure

• Remote rendering
  – Expensive, slow

• Data culling
  – Still expensive, but possible (Cheating-Death)

• Problem with both approaches
  – Requires a trusted game authority
  – Does not work for peer-to-peer games
    • e.g. Real-time Strategy games such as Warcraft 3
    • No trusted third party
    • Players do not trust each other and game performs identical and complete simulations on both ends
  – Must hide information while ensuring cheat-proof play
    • Security protocols to the rescue!
Trick #1: Zero-knowledge proofs

- Determine if two units share the same location without revealing their location
  - Perform cryptographic exchange to determine if a unit is in the same region as an enemy while leaking no information about your position
  - Information revealed to both entities only when it should be

Trick #1: Zero-knowledge proofs

- A, B have two units located at positions $p_a, p_b$
- Zero-knowledge proof to determine if $p_a == p_b$ without revealing $p_a$ or $p_b$ to opponent
  - A, B create secret keys $k_a, k_b$
  - A ↔ B agree on a random number $r$
  - A ↔ B exchange encrypted positions
    - A→ B: $(r + p_a)k_a$
    - B→ A: $(r + p_b)k_b$
  - A ↔ B encrypt and exchange each other’s messages again
    - A→ B : $((r + p_b)k_b)k_a$
    - B→ A : $((r + p_a)k_a)k_b$
  - A, B compare what they sent with what they received
    - If equal, $p_a == p_b$
Trick #1: Zero-knowledge proofs

• Problems
  – Scaling as number of secrets increases
  – Latency (multiple round-trip times per proof)
Trick #2: Bit-commitment

• “Sign” and send hashes of moves using secret key that is disclosed after game
  – Reveal game information and moves in the clear when other client is entitled to it
  – Replay game to ensure cheating did not occur
Bit commitment example

- Peer-to-peer Battleship

1. A knows B’s location

2. A fires and hits
   - Shoot(1,2) → hit
   - Shoot(2,2) → hit
   - Shoot(3,2) → hit

3. A cheats, A wins
   “Just lucky I guess”

1. A doesn’t know B’s loc

2. A fires, B lies
   - Shoot() → miss
   - Shoot() → miss
   - Shoot() → miss
   - Shoot() → miss

3. B cheats, A loses
   “Just unlucky I guess”
Bit commitment example

• Securing peer-to-peer Battleship
  – Pre-game
    • Exchange keyed cryptographic hashes of ship location using secret key
    • Commits players to a specific location without revealing it (bit commitment)
  – In-game
    • Send and receive shot coordinates in the clear along with whether opponent’s last shot hit or missed
    • Opponents can lie about hits/misses here
  – Post-game
    • Exchange secrets and initial ship location
    • Verify opponent’s integrity by checking all the evidence of shots vs. replies
Our work

• Apply bit-commitment to address information exposure cheats in real-time strategy (RTS) games
  – Warcraft 3
Background

• RTS games
  – You and your adversary build and control opposing armies
  – Army units balanced like rock, paper, scissors…
    • Knowing opponent’s armies makes it easy to win
  – Rely on “fog of war” to make game interesting
  – Have hundreds of secrets that change every moment
    • Unit type and unit location
    • Zero-knowledge proofs not feasible
    • Contrast to Battleship with one secret per player per ship!
Current RTS network protocol

- Exchange initial game state and all subsequent mouse clicks
- Each player simulates identical copies of game
  - PRO: no one can lie about what units they have
  - CON: each player knows state of the entire game
  - Just like Battleship
How it should work

Green unit, and its vision radius

Green shouldn’t see these enemies

Green should see these enemies
How it should work

Blue doesn’t send these units (they’re already known about by green)

Blue only has to send these units

Blue doesn’t send these units (they’re not visible)
Applying bit commitment to RTS

Key idea: You and your opponent only know each others “view area” not each others units

if (<click> is in oppView)
    send <click>
else
    send hash(<click>, secret)

1. myView
2. myUnitsViewable
3. <click> or h(<click>, s)
Modified RTS network protocol

- **Pre-game**
  - Create your secret $s$
  - Generate initial game state $igs$, send $h(s,igs)$

- **In-game**
  - Each time slice, send (and receive)
    - Your viewable area
    - Either your move $m$, or, if it’s invisible to him, $h(s,m)$
    - If one of your units just entered his area, send that unit

- **Post-game**
  - Exchange your secret, initial conditions, and all hidden moves throughout the game
  - Verify opponent’s integrity by simulating the game rapidly with the (now known) hidden moves
Issues

• Not all information is concealed
  – Old way: know everything
  – New way: know only viewable areas
  – *How much* information does the new way conceal?
    • Use Shannon’s uncertainty to measure

• Increased network requirements
  – Old way: bandwidth = number of clicks
  – New way: bandwidth = clicks or hash of clicks, viewable areas
    • Use Warcraft 3 tournament replays to measure

Adding incremental verification

- One-way hash chains with delayed key disclosure
  - Each player creates hash chain
    - $h_n, h_{n-1}, \ldots, h_2, h_1, h_0$
    - $h_n=$random secret and $h_{n-1} = \text{H}(h_n)$
    - Exchange $h_0$
    - Round i, commit moves with $h_i$, reveal $h_{i-1}$
  - Each unit given a random ID = N
    - While unit “hidden”, unit moves are committed by sending $\text{H}(N), \text{H}(<\text{click}>, N, h_i)$
    - Reveal N when unit appears
    - Opponent verifies units incrementally
Adding non-repudiation

• How to prove cheating to a 3rd party?
  – Use public-key cryptography: message signatures
    • Digest each message and encrypt the digest with private key
    • 3rd parties digest each message and compare with decrypted digest
  – Ideally public keys for this stored at game’s authentication server
Adding to other games

• Easily applicable if game is tolerant to
  – Added bandwidth overhead
  – Added CPU overhead

• Examples
  – Board games
  – Card games
Goal

• Find and apply security research in novel ways to games
• 3 projects
  – Using cryptographic protocols to prevent cheating
    • Mitigating information exposure in RTS games
  – Applying platform integrity tools to detect cheating
    • Tamper-resistant, hardware-based game measurements
  – Creating new game architectures using cryptographic primitives
    • Public-server MMORPGs
Cheats are prevalent

- Wallhacks
- Maphacks
- Macros
- Item duping
- Disc hacks
- Aimbots
- Collusion
- Speed hacks
- Chest hacks
- Auto looters
- Gamebots
- Rejoin hacks
Cheats are complex

- How they cheat
  - Read memory to expose information
  - Modify display path to add visual aids
  - Inject protocol messages
  - Modify game textures and models on disk or in memory
  - Programmatically play game on behalf of player
Cheats are complex

• How they hide
  – Polymorphism
    • Disassemble signatures being checked to thwart file and memory signatures
  – Privileged mode execution
    • Run in kernel to prevent anti-cheat detection
  – Run-time code patching
  – Direct kernel object modification
  – Selective disabling
    • When anti-cheat code is about to run
    • When new anti-cheat distributed
A daunting task…

Where can we look for help?
Malware is prevalent

- Trojans
- Viruses
- Worms
- Rootkits
- Keystroke logging
- Backdoors
- Malware
- Spyware
- DLL injection
- Bots
- Hijacking
- Hooks
Cheats and malware

• What do cheats and malware have in common?
• Software mechanisms used to cheat
  – Binary modifications
  – DLL injection and user-mode hooks
    • Import Address Table (IAT) hooks
      – e.g. DirectX hooks
    • Inline function hooking
  – Code caves
  – Kernel modules and kernel-mode hooks
    • See-through graphics drivers
    • Layered drivers
    • I/O, System Service Dispatch Table (SSDT), Interrupt Descriptor Table (IDT), Structured exception handler hooks
    • Packet editing
• An entire course could be devoted to this…
  – http://thefengs.com/wuchang/work/courses/cs592
Cheats and malware

• What else do they have in common?
• Software mechanisms used by counter-measures
  – File system integrity checks
  – Memory scanning
  – Process monitoring
  – Remote measurement (e.g. PunkBuster screenshots)

  – Examples
    • Tripwire, Symantec, chkrootkit, Snort, etc. (anti-malware)
    • HLGuard, VAC, PunkBuster, Warden (anti-cheat)
Cheats and malware

• What else do they have in common?
• Fatal flaws in counter-measures
  – Easily subverted or disabled by adversary
  – Adversary finds a way to run at “Ring 0” via privilege escalation or in-kernel drivers
  – Adversary modifies
    • Operating system
    • Any memory location
    • Drivers
    • Even the anti-cheat system itself!
  – Warden vs. WoW Glider
• Towards a common approach for securing counter-measures
  – Get “beneath” the cheat/malware
  – Rely on an “Angel in the Box”
Angel in the Box

- A trusted, tamper-resistant processor that is hidden from the applications and operating system running on the host
  - Ring “–1”
  - Only runs signed code
  - Has access to key components of running system

- Paradigm
  - Run any cheat you want, but the angel is watching
Example Angel

- Intel’s Active Management Technology platform
Using the Angel

• Currently being used to secure hosts
  – Tamper-proof detection of rootkits, spyware, viruses, and malware via memory and file system integrity checks
  – Tamper-proof monitoring of critical processes
  – Network quarantine

• Applied to games
  – Tamper-proof detection of cheats via memory and file system integrity checks
  – Tamper-proof monitoring of anti-cheats and peripherals
  – Game protocol integrity
Goal

• Find and apply security research in novel ways to games
• 3 projects
  – Using cryptographic protocols to prevent cheating
    • Mitigating information exposure in RTS games
  – Applying platform integrity tools to detect cheating
    • Tamper-resistant, hardware-based game measurements
  – Creating new game architectures using cryptographic primitives
    • Public-server MMORPGs
MMORPG

- Massively Multiplayer Online Role-Playing Game
  - One of the most popular game genres
  - Popular MMOs
    - World of Warcraft, Lineage
MMORPG problems

• Centralized hosting expensive
  – Hosting costs: 20% of subscription revenue
  – Support costs: 20% of subscription revenue

• Content generation expensive
  – Dominate cost of MMOs after launch
  – Costs growing faster than game revenue

• What are games without those problems?
Public server games

• What’s a public server game?
  – Game company distributes the server code
  – Users run the code on their own servers (available publicly)
  – Users can modify the server
  – Examples
    • Half-life, Counter-Strike, Neverwinter Nights

• Drawbacks
  – No subscription model
  – Not “massive”
  – No persistent world

• Benefits
  – Leverage user hosting and content generation resources
User resources

• How much is there much to harness?
  – Server resources
  – Content generation resources
Server resources are plentiful

70% of all servers are empty

CDF of fullness of Counter-Strike servers
User content generation plentiful

- **Half-life**
  - 6 official mods
  - 492 user-developed mods (on Wikipedia alone)
    - Counter-Strike originally a user mod
- **Neverwinter Nights**
  - 7 studio-developed modules (expansions)
  - 4372 user-developed modules
- **Second Life**
  - 80k player-hours / day spent playing
  - 25% is user content creation!
  - 10 user-years/day in content development
- **User content as popular or more popular than studio content**
Our Goal: Public Server MMORPG

- Decrease hosting and support costs by letting users host gameplay
- Decrease development costs by letting users generate content
- Allow for a persistent world built with public servers
Categorizing existing games

<table>
<thead>
<tr>
<th>Game</th>
<th>User content</th>
<th>Architecture</th>
<th>Persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical RTS</td>
<td></td>
<td>P2P</td>
<td></td>
</tr>
<tr>
<td>Typical MMO</td>
<td>X</td>
<td>Client-server</td>
<td>X</td>
</tr>
<tr>
<td>Typical FPS</td>
<td>X</td>
<td>Public server</td>
<td></td>
</tr>
<tr>
<td>Neverwinter Nights</td>
<td>X</td>
<td>Public server</td>
<td></td>
</tr>
<tr>
<td>Second Life</td>
<td>X</td>
<td>Client-server</td>
<td>X</td>
</tr>
<tr>
<td>PS MMO</td>
<td>X</td>
<td>Public server</td>
<td>X</td>
</tr>
</tbody>
</table>
Incentive-based design

• Player incentives unchanged
  – Fun
  – Persistent loot/advancement for time invested

• Hosting and content generation incentives
  – Pride
    • e.g. Second Life
  – Control of game rules and maps
    • e.g. Half-Life, Counter-Strike
  – Control of loot
    • Public servers awarded loot (virtual and real) based on player-minutes
    • Player-minutes drive PSMMO’s virtual economy
    • Public servers must provide compelling content to keep players on their server
PSMMO Overview

• Public servers host gameplay
• Publisher hosts authentication, billing, loot server
• Players store persistent data (i.e. loot) themselves
• Key challenges
  – Network all those servers together (we don’t do this)
  – Provide great uptime with user-run servers (or this)
  – Authentication
  – Issuing, verifying, managing the persistent data
Example: CSMMO

• Imagine Counter-Strike the MMO
• Player picks a public server to play on from many
• Player advances a persistent state via gameplay tasks
  – Earns tokens via kills and victories
  – Trades tokens in eventually for better (persistent) loot
• Loot examples
  – Better weapons
  – Faster running speed
  – New outfits
  – Trophy rack
  – Bank account
Why focus on loot?

• Loot is very important
  – Primary motivator for gameplay in MMOs
  – Loot is the persistent reward
  – Acquiring abilities, possessions, statistics, money can cost players hundreds of hours

• Keeps gameplay open to publisher/user innovation
  – Public servers decide game rules/content and how to issue loot
  – Can vary from public server to public server
  – CS, fishing, chess, driving, killing monsters
  – Competitive vs. cooperative games
Design pitfalls (incomplete)

• Anyone can put up a server!
  – Can they play on their own server and win all the time?
  – What do they get?
  – Can they write a bot to play on their own server, away from prying eyes?

• Anyone can design loot!
  – Can they design the most powerful loot?
  – Can they issue it to themselves?

• Anyone can join a server, and clients store their own data!
  – How can a server trust a client’s loot?
Design goals to address cheating

• Ensuring player-minutes are not fabricated and are not being played by bots
• Ensuring loot is not fabricated or duplicated
• Ensuring loot is balanced and fair
Ensuring player-minutes are authentic

- Players periodically authenticated by authentication server to determine which public server is accumulating their player-minutes
- Players periodically tested with CAPTCHAs by the authentication server
- Each completion of a CAPTCHA grants authenticated player minutes to that player’s server
- Prefer game-specific tests to avoid generic CAPTCHA farms
Ensuring loot is authentic

- Loot issued to servers based on authenticated player minutes logged at the server
- Loot “minted” via a centralized loot server using public-key cryptography
  - Loot server creates loot and binds it to player
  - Loot server uses its private key to sign loot
  - Public servers ensure loot is authentic using loot server’s public key
Ensuring loot is balanced

• Users design any sort of gameplay
• Users design any non-persistent loot for their mod
• Users design persistent loot, but such loot must be examined and balanced by publisher before being issued by loot server
  – Persistent loot must meet verifiable standards (e.g., point balanced)
  – Publisher examines and rates the power of user generated loot before allowing loot server to issue
  – Loot server ensures power of issued loot is proportional to player-minutes
Overall operation

1. Authenticate
2. Gameplay
3. Loot request
4. Bound loot
5. Bound loot
Limitations (incomplete list)

- Player stores loot
  - Has to manage backups
  - Has to manage across computers
- Loot is bound to a player
  - No trading with other player
  - No buying or selling

Item 1
attributes
Signatures
bind_key_p1
loot_key
Player 1
Challenge: Trading Items

- We would like to relax the “loot cannot be traded” restriction for game-play purposes
- Trade: Player A had item, now Player B does
  - How can player A not have it anymore?
  - We don’t want to allow item duping
  - We don’t want to keep an item revocation list (not scalable)
- Solution: periodic trading window and item re-minting
  - Publisher coordinates a player-wide swap meet or auction
  - During auction, all items of players on-line are re-minted
  - Old items are invalidated by changing minting keys
    \((Loot\_key\_pub, Loot\_key\_priv)\)
  - Loot server re-mints items based on trades using new minting keys
  - Off-line players have their items re-minted using new minting keys upon next connection
Trading Illustration

Submit items to mint
Trading Illustration

Receive new items

Player 1

Item 2
attributes
Signatures
bind_key_p1
loot_key'

Item 3
attributes
Signatures
bind_key_p1
loot_key'

Mint

Player 2

Item 1
attributes
Signatures
bind_key_p2
loot_key'
Burning questions

• How can we prevent servers from giving players loot for doing nothing?
  – We don’t
  – We do require players to be actively playing and we assume they will gravitate towards compelling content

• How can we prevent servers from running unfair gameplay rules (eg, only give loot to admin)
  – We don’t
  – In other public server games, players avoid cheating servers by reputation
  – Servers taking a cut of the loot may actually be okay with players!
Summary

• Public-server architecture for persistent MMORPG
• Key incentive
  – Public servers earn loot for authenticated player-minutes
• Advantages
  – Leverage user resources in content generation and hosting
• Disadvantages
  – Limited trading, no guarantees of fair gameplay

Conclusions

• Cross-pollination of security and games can be profitable
  – Protects existing on-line games
  – Creates new game architectures with better cost structures

http://thefengs.com/wuchang/work/cstrike

http://mshmro.com
Extra slides
Cheating is prevalent across genres

**RTS**
- Map hacks
- Chest hacks
- Macros
- Disc hacks
- Collusion

**FPS**
- Wall hacks
- Aimbots
- Macros
- Rejoin hacks

**MMO**
- Chest hacks
- Map hacks
- Auto looters
- AFK Bots
- Macros
- Item duping
- Collusion
- eBay of Virtual resources
Categorize cheats to address them

<table>
<thead>
<tr>
<th>RTS</th>
<th>FPS</th>
<th>MMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maphacks</td>
<td>Wallhacks</td>
<td>Chest hacks</td>
</tr>
<tr>
<td>Chest hacks</td>
<td></td>
<td>Maphacks</td>
</tr>
<tr>
<td>Macros</td>
<td>Aimbots</td>
<td>Auto looters</td>
</tr>
<tr>
<td></td>
<td>Macros</td>
<td>AFK Bots</td>
</tr>
<tr>
<td>Disc hacks</td>
<td>Rejoin hacks</td>
<td>Item duping</td>
</tr>
<tr>
<td>collusion</td>
<td></td>
<td>collusion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eBay of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Virtual resources</td>
</tr>
</tbody>
</table>


Categories of cheats

- Information Exposure
- Abstraction of Input or Output
- Protocol Manipulation
- Out-of-path
Examples of cheats per category

• Information exposure
  – Wallhacks (OGC), Maphacks (Warcraft 3), Chest hacks (showEQ)

• Automation and abstraction
  – Aimbot (OGC), Troop command macros (Warcraft 3), Auto-looting (WoW QuickLoot), AFK bots

• Protocol
  – Reset cheat (Half-Life), Unit fabrication (Warcraft 3), Item duping (MMO), Speed hack (Half-Life), Hit point hack (Diablo), Disconnect cheat

• Game bugs
  – Game-specific coding errors that lead to unintended behavior
Automation cheats

- Automate game activities via Bots
- Aimbots
  - OGC
  - Automate aiming in FPS
- Macros and game bot farming
  - MacroQuest for EQ2
  - Automate wealth acquisition via programs
Abstraction

• Input Abstraction, Output Abstraction
• AI Definition: *write user inputs or gameplay messages decreasing user interactivity*
• AO Definition: *display refined information to the user to guide input*
• Examples:
  – FPS: Aimbots (AI)
  – RTS: Troop command macros (AI)
  – MMO: bots for selling buffs (AI)
  – Cards: card counting (AO)
  – FPS: color enemies red (AO)
Protocol cheats

• Hit point cheating
  – Diablo protocol messages indicating damage done to enemy
  – Inject messages with inflated damage to instantly kill opponent

• Item duping
  – Disconnect while dropping item
  – Ambiguity in whether event happened globally

• Speed hack
  – Inject movement messages to make your character move or fire “faster” than normal
Protocol Cheats

• Definition: *write messages not generable by user actions to exploit weaknesses in communication protocol*
• The Problem: server API suffers bugs and design flaws/tradeoffs like any other code
• Protocol cheats take advantage of squiggle room in protocol
• Examples:
  – FPS: player reset
  – RTS: inventing units
  – MMO: item duping
  – Speedhacks
Protocol Cheats

- **Definition:** write messages not generable by user actions to exploit weaknesses in communication protocol
- **The Problem:** server API suffers bugs and design flaws/tradeoffs like any other code
- **Protocol cheats** take advantage of squiggle room in protocol
- **Examples:**
  - FPS: player reset
  - RTS: inventing units
  - MMO: item duping
  - Speedhacks

![Diagram showing hit points comparison between 11hp and 65536hp in Diablo Client]
Game cheats

• Exploit inconsistencies and errors in game code
  – Magic “pizza” machine in The Sims On-line
  – Vending machine and pawn shop hack in Lucasfilm’s Habitat
  – Skin cheats in Counter-Strike
  – Not highly relevant to this course
Out-of-path cheats

• Important problems, not in the control path
• Rule: “If it can be achieved via normal gameplay, it’s not a control path cheat”
• Things not considered control path cheats that are still bad include…
  – Game bugs such as duping via normal gameplay
  – Map exploits
  – Vendor design exploits
• More Examples:
  – Password stealing, cd-key stealing, DDOS, remote root
  – Buying/selling virtual items on eBay
  – Collusion, two-boxing
  – Performance-enhancing drugs, robots
Bit commitment example

• Fair coin flip
  – Each participant comes up with a secret key
  – Selects and encrypts either “heads” or “tails”
  – Exchanges encrypted messages
  – Exchange secret keys
  – Whoever was the “flipper” wins if answers differ, loses if they’re the same
Cheating links

• General
  – http://rpgexploits.com
  – http://msxsecurity.com
  – http://zerogamers.com

• WoW
  – WoW Glider
    • http://wowglider.com
  – WoW radar, WoW Sharp, ByteBot, GALB
  – WardenNet, ISXWarden (anti-anti-cheats)
    • http://ismods.com/warden
    • http://edgeofnowhere.cc/viewtopic.php?t=311208
    • http://www.rootkit.com/newsread.php?newsid==360
  – ISXWoW
    • http://ismods.com/downloads.php
Cheating links

• Half-Life
  – OGC
    • http://mpcdownloads.com
    • http://www.mpcforum.com/showthread.php?t=31409

• EverQuest 2
  – MacroQuest
    • http://sourceforge.net/projects/macroquest
Anti-cheat links

- WoW Warden
  - http://www.ismods.com/warden
- PunkBuster
  - http://punkbuster.com
- Valve Anti-Cheat (VAC)
- HLGuard, Cheating-Death
  - http://unitedadmins.com
- Intel’s AMT
  - http://www.intel.com/go/iamt/