CS 6v81-05
Game Hacking II: Preventing map hacks

Kevin Hulin
Department of Computer Science
The University of Texas at Dallas

November 18th, 2011
Outline

1. Introduction
2. RTS - Overview
   - Objective and Terminology
   - Types of Cheats
   - Vulnerability
3. Hacking with Kartograph
   - Map Hacking
   - Unit Hacking
4. Preventing Map Hacks with OpenConflict
   - Passive Attacks
   - Active Attacks
5. Conclusion
Outline

1. Introduction
2. RTS - Overview
   - Objective and Terminology
   - Types of Cheats
   - Vulnerability
3. Hacking with Kartograph
   - Map Hacking
   - Unit Hacking
4. Preventing Map Hacks with OpenConflict
   - Passive Attacks
   - Active Attacks
5. Conclusion
Have you ever had a dream that you were so sure was real?

https://www.youtube.com/watch?v=B31ubihpOhc&hd=1
Why do we care about game hacking?

Who cares if I cheat?
Introduction

Online Gaming = Big Money

According to the Entertainment Software Association...

- 67% of American Households play video games
- Computer games represent a 9.9 billion dollar market
- Online gaming includes 64% of gamers
But who would cheat?

Cheating takes the fun out of gaming!
There is money to be had!

Professional Gaming!

- 2010 Electronic Sports World Cup - 500 professional gamers play for $36,000 cash
- As of July 2011 StarCraft 2 tournaments paid over $1.4 million
Gaming for Profit

Some genre’s are more popular among professional gamers

- 35.5% - Real Time Strategy (RTS) – Most popular
- 10.1% - First Person Shooter
- Other genres fall behind (RPG, Puzzle, Sport, Adventure)
Outline

1. Introduction

2. RTS - Overview
   - Objective and Terminology
   - Types of Cheats
   - Vulnerability

3. Hacking with Kartograph
   - Map Hacking
   - Unit Hacking

4. Preventing Map Hacks with OpenConflict
   - Passive Attacks
   - Active Attacks

5. Conclusion
1. Introduction

2. RTS - Overview
   · Objective and Terminology
   · Types of Cheats
   · Vulnerability

3. Hacking with Kartograph
   · Map Hacking
   · Unit Hacking

4. Preventing Map Hacks with OpenConflict
   · Passive Attacks
   · Active Attacks

5. Conclusion
Real-time Strategy Games

Objective
- Gather Resources
- Build units and structures
- Wipe out your opponents
Real-time Strategy Games

Terminology

- **Structures**
- **Units**
- **Resources**
- **Mini-map**
Real-time Strategy Games

Types of Cheats

- **Resources**
  - Increasing the number of resources you have
  - Reducing the cost of your units

- **Units**
  - Increasing your units’ stats
  - "God-mode" - Units do not take damage

- **Map**
  - Lift the "Fog of War"
Map Hacks

Lifting the Fog of War
What makes RTS games vulnerable to Map Hacking?

Peer-to-peer

- RTS games are generally very fast paced
- Games are played peer-to-peer without a central server to mediate
- Broadcast your unit locations and actions to your opponent
- Let the game client handle making sure you can only see what you should see
The information you seek is within your grasp!

Exploiting the Peer-to-Peer nature of RTS

- Since everything is being broadcast, each client has a complete view of the game state
- A client could gain an unfair advantage if he/she could view the parts of this information beyond what they should be able to see
- Just a matter of tricking the game into showing it to you...
The Visibility Map

- In many RTS games, the Fog of War is implemented as a 2D array called the Visibility Map.
- Each index in the map corresponds to a unit of area on the map.
- Whether a location is rendered as viewable or as "behind the fog" depends on this value.

Composite Maps

- The visibility map could also be represented as the composition of multiple structures in memory.
- In this talk, we will assume we are working with a 2D array, however the same techniques could be used in either case.
Needle in a haystack

The Visibility Map could be anywhere!

- Today’s games have a large memory footprint (> 1 GB for Supreme Commander)
- The information we seek is comparatively very small (< 1 MB)
- Use a process of elimination to sift through memory and locate the information we’re interested in
Finding the needle

Four easy steps

- Launch the game - Make a copy of the game’s memory space
- Let the game play, triggering as many changes as possible without changing the viewable area
  - Eliminate memory blocks that changed
- Move some units into the fog so that the viewable area is changed
  - Eliminate memory blocks that remained constant
- Repeat.
Understanding the Map

Before we can use the map, we need to understand it

- **Differential Analysis**
  - Move a unit and observe how the map changes

- **Frequency Analysis**
  - Observe what values are expected in the visibility map
  - These are the values that we will be using to make other parts of the map visible
Exploiting the Map

Trick the client into telling us where the enemy is

- In many games, the visibility map stores a 0 to indicate Fog of War
- Other values indicate the number of units that can see a certain area of the map
- Forcing this map to a value such as 1 will cause the game to think we have units where we do not
- Thus, the entire map is shown to us – in real time!
Supreme Commander 2 (March 2010) - Lifting the Fog of War
https://www.youtube.com/watch?v=mFprkIAeKgM (1:22 - 11:40)
1. Introduction

2. RTS - Overview
   - Objective and Terminology
   - Types of Cheats
   - Vulnerability

3. Hacking with Kartograph
   - Map Hacking
   - Unit Hacking

4. Preventing Map Hacks with OpenConflict
   - Passive Attacks
   - Active Attacks

5. Conclusion
Unit Hacking

Similar to Map Hacking but harder

- Unit data structure is much smaller/ more complex than the visibility map
- Can’t visualize it as a bitmap – Generally stored as pointer stack
- Key statistic values are often encrypted to prevent cheating
- E.g. A unit’s health is often stored as $Z \oplus HP$ where $Z$ is a secret key
- Defeats simple memory scanning tools such as Cheat Engine
Unit Hacking

1. Reduce memory space
2. Produce units
3. Find unit structures
4. Understand unit structures

References:
- RTA - Overview
- Hacking with Kartograph
- Preventing Map Hacks with OpenConflict
- Conclusion
Modifying the game state

Values themselves are generally not important

- Only need to know where the HP of a unit is represented in memory
- Tell our program to freeze the value that represents unit health
- Effectively enabling God-Mode / Invincibility!
Demo

Age of Empires III (October 2005) - Invulnerable Units
http://vimeo.com/13972467 (9:35 - 11:15)
Outline

1. Introduction
2. RTS - Overview
   • Objective and Terminology
   • Types of Cheats
   • Vulnerability
3. Hacking with Kartograph
   • Map Hacking
   • Unit Hacking
4. Preventing Map Hacks with OpenConflict
   • Passive Attacks
   • Active Attacks
5. Conclusion
Threat Model

Passive Eavesdropping Adversaries

- Adversary has unlimited access to make copies of the game’s internal memory
- Has complete understanding of the game’s internal data structures
- Defeats our system if he can create a program $P$ to parse and display information about the opposing team that should not be visible according to the game’s rules
- If the adversary is unable to glean any additional information, we call the game secure against a passive adversary
- Note: assuming a fast paced peer-to-peer architecture (e.g. SCII) – Otherwise, it is trivial to allow the server to enforce security
Cryptographic Solution

Store only data that the client is allowed to see

- For players Alice and Bob, Alice should be able to see only Bob’s units that are near to her own.
- To ensure security against a passive adversary, Bob should only send data to Alice about units that Alice should be allowed to see.
- Problem: Bob does not and should not know which of his units Alice can see.
- Solution: Use an oblivious intersection protocol.
Oblivious Intersection Protocol

Formal Definition

- Let $M$ be the set of all cells on the map
- Let $V_A \subseteq M$ be the set of cells that Alice can see
- Let $U_B \subseteq M$ be the set of map cells containing Bob’s units
- Our goal is to compute $V_A \cap U_B$ subject to the following constraints

Oblivious Intersection Constraints

- Bob should learn nothing about $V_A$
- Alice should learn nothing about $U_B$ other than $V_A \cap U_B$

Simplicity Assumption

- Note: For simplicity we only consider the problem of calculating $V_A \cap U_B$; the reverse should be calculated similarly for sending information about Alice’s units to Bob
Oblivious Function sub-protocol

Let \( o_k(v) \) be a keyed function \( o_k : v \rightarrow D \)
- Bob holds the secret key \( k \)
- Alice is able to learn \( \{o_k(v) : v \in V_A\} \)
- Alice does not learn \( o_k(w) \) for \( w \not\in V_A \)
- Bob learns nothing about \( V_A \)
### Implementation

#### Oblivious Function Definition

- Let \( o_k(v) := H_1(v)^k \in \mathbb{G} \)
- Where
  - \( H_1 : M \rightarrow \mathbb{G} \setminus \{1\} \) is a hash function
  - \( k \in [1, q - 1] \) for large prime \( q = ||\mathbb{G}|| \), for group \( \mathbb{G} \)

#### Evaluation protocol

- Alice chooses a random integer \( r \in [1, q - 1] \) and sends \( x := H_1(v)^r \) to Bob
- Bob responds with \( y := x^k = H_1(v)^{rk} \)
- Alice computes \( o_k(v) = y^{r^{-1}} = H_1(v)^k \)
Security

Computational Diffie-Hellman

- CDH Assumption: Given $g, g^a, g^b$ it is computationally intractable to compute $g^{ab}$ for $a, b \in [1, q-1]$

Reduction to Discrete Logarithm Problem (DL)

- If one could compute the discrete logarithm base $g$ of $g^a = a$ then one could solve CDH simply by exponentiating $g^{ab} = (g^b)^a$

Honest but Curious Assumption

- Security only holds if Alice is honest
- A much stronger assumption than CDH is required for a dishonest Alice
# Oblivious Set Intersection Protocol

<table>
<thead>
<tr>
<th>Back to our problem of computing $V_A \cap V_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Bob chooses random key $k \in [1, q - 1]$</td>
</tr>
<tr>
<td>- For each $u \in U_B$</td>
</tr>
<tr>
<td>- Compute key $k_u \leftarrow H_2(o_k(u))$ where $H_2$ is a hash function</td>
</tr>
<tr>
<td>- Encrypt the message $f_B(u)$ using $k_u$ using an authenticated encryption (includes MAC)</td>
</tr>
<tr>
<td>- Bob sends resulting ciphertexts to Alice</td>
</tr>
<tr>
<td>- Alice engages in the oblivious function evaluation protocol (described earlier) to obtain $y_v \coloneqq o_k(v)$ for all $v \in V_A$</td>
</tr>
<tr>
<td>- For each $y_v$ Alice computes $k_v \leftarrow H_2(y_v)$ and tests if one of the ciphertexts received from Bob decrypts correctly under $y_v$</td>
</tr>
<tr>
<td>- For each correct decryption of $y_v$, Alice learns $v \in V_A \cap U_B$ and $f_B(v)$</td>
</tr>
</tbody>
</table>
Improvements

**Chaff**
- Alice leaks the number of cells in her visibility set $V_A$
- Bob also leaks the number of units he has in $U_B$
- We can reduce the information leaked in these cases by padding queries with chaff - meaningless data
- Thus, the protocol will leak an upper bound on the number of units or cells rather than the actual amounts

**Hypergrids**
- Visibility regions are generally large and continuous
- Decompose $V_A$ as a union of grids
- Alice sends # queries proportional to the perimeter of $V_A$ rather than its area
- Reduces the complexity of Alice’s queries by a factor of 6
OpenConflict - Proof of Concept

Weaker Crypto for Better Performance

- NIST standard Elliptic Curve cryptography, single exponentiation takes $\approx 1\, ms$
- With 200 visibility hypertiles and 150 units per player, this would amount to $\approx 750\, ms$ – Unacceptable
- Implement weaker and more efficient elliptic curve crypto ($q \in [1, 2^{127} - 1]$) – reduce exponentiation time to $\approx 11\, \mu s$
OpenConflict - Security

Security

- In empirical testing, it took 12 seconds to solve discrete logarithms for \( q = 2^{61} - 1 \)
- We can reason that it would take about 3200 machine-years to solve DL for \( q = 2^{127} - 1 \)
- Investing 1000 times more compute power will have a 1 in 28,000 chance of breaking the key within the duration of a game → the system is said to be computationally resistant to passive cheaters
OpenConflict - Evaluation

Evaluation

- In the worst case to be expected in online play, there could be a lag of about 25 ms
- This would be compounded by network latency
- Still working to make the system more efficient

Project Release

- Kartograph remains unreleased, but is promised to be released alongside a cryptographic security package that implements the security ideas in this paper and would defeat Kartograph.
Discussion - Active Attacks

- Near impossible to prevent in real time
- Would require all players to verify (in Zero Knowledge) the actions of all others
- Discovery of a cheater post game, however, would be simple.
  - During the game, every client signs its network traffic and periodically sends a hash of it to other players
  - After the game, all commitments + traces are uploaded to a central server
  - Server verifies that the actions made during the game are consistent with the game rules and that the network traffic agrees with the commitments
Preventing Active Attacks

Discussion - PRNG

- Another issue arises for random number generation (Manipulating the random number generator could cause a player’s units to deal extra damage (crit) every hit.
  - This can also be checked off-line if all parties agree to a randomly selected seed for their pseudorandom number generators
Outline

1. Introduction

2. RTS - Overview
   - Objective and Terminology
   - Types of Cheats
   - Vulnerability

3. Hacking with Kartograph
   - Map Hacking
   - Unit Hacking

4. Preventing Map Hacks with OpenConflict
   - Passive Attacks
   - Active Attacks

5. Conclusion
Security in Peer-to-Peer Gaming is Hard

- In gaming, performance is of utmost importance
- Any latency is considered intolerable for professional gamers
- As a result, the security of a game is often overlooked in exchange for increased performance
- The consequences are insecure data structures in memory that can be easily reverse engineered
- Passive cheating (e.g. Map Hacking) is impossible to detect in real-time since the client has not been modified and the modified data is not distributed to other players
- Active cheating is more sinister, but can be detected after a game has concluded
Conclusion

Parallels in Trusted Computing

- The problem of RTS Map security is very closely aligned with trusted client computing.
- Not (just) an excuse to play video games...
Questions

