Programming Assignment 3: Hack 'n' Seek
CS 3410 Fall 2008 **Due November 25th at 11:59 p.m.**

Overview

This assignment will help you and your partner further hone your skills as elite hackers (31337 h4x0r). Your goal is to write a program (a bot) that plays the game of Hack 'n' Seek. Hack 'n' Seek simultaneously tests your hacking skills as well as your ability to protect programs from being hacked. Writing a top notch Hack 'n' Seek bot will require all the skills you have been developing so far in the course and, hopefully, be a fun and interesting challenge. At the end of the assignment, we will hold a tournament pitting all your bots against each other and award, not just a sense of m4$$1V3 l337N3$$[1] (massive leetness), but actual bonus points on the assignment. Plus we will be holding a grad-student banquet (read: pizza, cookies, and soda[2]) for all!

Overview of Hack 'n' Seek

The game of Hack 'n' Seek is played by a pair of bots at a time. Both bots are placed in a single shared memory space. The bots are executed by the same MIPS simulator we have often used, but modified to jointly execute both bots simultaneously: one instruction of bot A then one instruction of bot B, and so forth. Each program can read: it's own half of the memory space, the compiled code of the opponent, the opponent's stack and global memory, and the contents of its opponent's register file; however, a bot can only write in its own memory space (i.e. writes into the opponent's memory are ignored). At the beginning of the game, the simulator starts each program by calling the function

```
void __start(int secret)
```

handing each bot a piece of secret information (a single integer value). The goal of Hack 'n' Seek is to detect the opponent's secret and report it to the simulator before your own secret is exposed. A bot can test whether an integer is the opponent's secret by calling

```
void check(int my_secret, int opp_secret)
```

If neither player can detect the opponent's secret after a fixed number of instructions have been executed, the game has a second victory condition. In this case, the victor is the bot that has stored the most copies of its secret in a special, small "taunt" array known to both players.

Hack 'n' Seek is difficult because a bot cannot win if it forgets its secret. To win your bot must either write its secret into the taunt array or pass it to the simulator to test your opponent's secret. In either case, your bot must construct your secret in a register or memory location beforehand and, since your opponent can read these locations, they can potentially analyze your program and memory to detect it. A successful bot will use clever strategies to detect the opponent's secret while ensuring that it is not vulnerable to the same type of attacks.

Game Details

In order for the game to work, a few specifics of the game's execution must be defined.

1. When `start` is called the simulator passes the value of the secret in register $4 ($a0). In order to give every bot a fair chance to move its secret from this known location, if you try to read anything in your opponent's context (memory or registers) before the first 20 instructions have been executed, you will get zero.

[1]Yes, I know what you are thinking, this *never* stops being funny.
[2]Grad student jokes are even funnier.

2. Related to the last point when you call `check`, your program must place your secret in register $4 before jumping into the `check` function. Polling register $4 is such an obvious attack that we have changed the simulator to perform three specific operations atomically when you call `check`: the setting of register $4, the jump into `check`, and the clearing of $4 to zero.

3. To make the game more interesting, calling `check` is not free. If your check fails, the simulator will write your secret into the "taunt" array in a random location unknown to both your bot and your opponent's bot. When your guesses fail, be sure to clean up your secret quickly before your opponent finds it.

4. Memory mapping: the simulator makes it seem like your bot lives in the lower 2GB of memory using addresses less than or equal to `0x7FFFFFFF`. Your opponent is mapped into the upper 2GB starting at address `0x80000000`. Your stack starts near `0x7FFFFFFF` and grows down and your code is placed starting near address `0x00000000`. Likewise your opponent's stack starts near `0xFFFFFFFF` and their code starts near `0x80000000`. We say "near" the various addresses because the simulator needs room to map registers into memory and shifts the stacks and code slightly to accomplish this. However, to make memory access simpler we supply a header file (described below) for your bots that supplies logically named macros to generate pointers to various important memory regions, such as registers, taunt arrays, stacks and code. There are also several functions defined, such as rand(), check(), printi(), and prints(). The header also explains in detail how memory is laid out relative to your bot.

Basic Strategy

What can be said? Find your opponent's secret fast and keep yours well hidden. If you think that your search is hopeless, dump your secret into the taunt array, and hope for the best. Hopefully, Hack 'n' Seek is easy to understand but subtly challenging. You will want to think carefully about what you do with your secret and what could be potential vulnerabilities in your opponent. Your most powerful weapon is your detailed knowledge of how a program lays out memory and behaves while it is executing. You know how each stack frame is designed, you know how a program uses registers during execution, and you know how to decode compiled MIPS assembly. Using this information to its fullest is your best chance of success.

Remember there are two win conditions. The first is to guess your opponent's secret before (s)he guesses yours. The second is to have the most copies of your secret in your taunt zone. Remember that your opponent can read the contents of your memory, so do not place your secret there too early. To get started, we have supplied eight example bots that employ some basic strategies. Their source code can be found in the same directory as the Hack 'n' Seek simulator. The example bots are:

noop: The simplest bot there is; it does nothing.
stalemater: Doesn't try to win, so it plays to annoy; erases its secret and does nothing.
quickdraw: Tries to catch you passing your secret around; rapidly scans register $4.
reghider: Sneakily hides its secret in the $LO multiplication register (check the MIPS handbook).
regscanner: Submits all the values in its opponent's registers as fast as it can; always beats reghider.
stackhider: Stashes its secret onto the stack and hopes for the best.
stackscanner: Submits opponent's stack values until it gets lucky; always beats stackhider.
tauntscanner: Figures no one is perfect; submits taunt array words hoping for a mistake.

Playing the game

The Hack 'n' Seek simulator and the example bots can be found on the CSUG machines at

```
/courses/cs3410/hacknseek
```

Copy the whole directory to your home directory to get started.

```
cp -R /courses/cs3410/hacknseek ~
```

A game bot is simply a C program compiled into MIPS assembly. The `bots` directory contains a `Makefile` to do the compilation for you. The `bots` directory already contains the bots listed above. Changing to that directory and typing `make` will compile all the example bots and prepare them for simulation. When you write your own bot, you can add it to compilation list by editing the `Makefile`. Add the name of your bot (minus the .c extension) to the list of `TARGETS` and run `make` to compile.

In the main directory, you will find the Hack 'n' Seek executable `hacknseek` and a quick play script `playall` that tests its input bot against all of the bots above. The simulator `hacknseek` takes two inputs, the names of the bots in the match, for example:

```
./hacknseek bots/noop bots/quickdraw
```

pits noop against quickdraw (Player 1, quickdraw, wins.) while `playall` takes only one, the bot to pit against the all the examples:

```
./playall bots/quickdraw
```

You can see quickdraw does quite well against it peers. You will likely want to read the assembly for a particular bot. As you have seen before you can decompile a bot file with the command:

```
/courses/cs3410/mipsel-linux/bin/mipsel-linux-objdump -xdl <bot filename>
```

Finally you have two tools to make bot development easier. First, you will want to read the `helper.h` file in the `bots` directory. It is the only header file you can import into your bot programs but it is designed to be all you will need. It contains the basic functions `check` , `printi` and `prints` but, more importantly, its comment describe, in detail, how your program is laid out in memory and how your opponent's information is mapped into your address space. It also contains the essential macros for generating pointers to various memory regions. For example:

```
int* p = OPP_REG(4);
```

sets `p` to point to your opponents register 4. Your second major tool is the `hacknseek` simulator which has several useful command line options for debugging bots. Type `hacknseek` (no arguments) to see a list.

Grading

Your grade for this assignment will have three parts.

80 points for designing a bot that beats all of the example bots above every time it plays them.

20 points for designing a bot that beats several secret course bots. We are working on a system that will allow your bots to play each other and example secret bots without revealing the source assembly. This will be made available in the middle of the week of November 17th.

Bonus!! for designing a bot that wins the 3410 bot tournament. The tournament will pair your bot against others in series. Each pairing will play several games (for example, 16) with alternating start order. A player receives 1 point for winning a game, 0 for a tie, and -1 on a loss. The player with the higher score at the end of all games in the pairing wins. If tied at the end of the pairing, the victor is the player who does best against the secret 3410 bots. Periodically, a round ends, lower scoring bots are eliminated, and new pairings are made between the bots that survive. Rounds continue until only one bot remains. The overall elimination strategy will be either double elimination or all-pairs depending on the final plans for the showdown party.

What to Submit

The source code for your bot file. Good luck.