2 Game Hacking 101

Software piracy has long been a problem in the computer games business—ever since games moved from stand-alone machines in the 1970s to PCs in the 1980s. Game makers, justifiably, have gone to great lengths to thwart piracy. In the past, game makers added various countermeasures to their software to make games harder to crack. The main purpose was to prevent rampant copying so that people who wanted to play the game had to buy it. In the end, these games were always cracked—but in some cases, the countermeasures delayed the release of a cracked version by days or even weeks. This delay earned real revenue for the game companies because delaying a crack for even a week translated into hundreds of thousands of dollars.

Antipiracy countermeasures made some economic sense in the over-the-counter paradigm, in which a gamer purchased a copy of the game from a retailer and installed the copy locally on his or her PC. But things have changed. Many modern games have moved online, and with the advent of game consoles connected to the Internet, this trend is likely to accelerate.[1] That means companies now have two revenue sources to protect: the

1. The hugely popular Nintendo Wii, which debuted in late 2006, will certainly accelerate this trend.

original game price in the retail channel, and a monthly subscription revenue stream for online access.

In this chapter, we'll describe a number of cheating techniques that have become mainstream and discuss new techniques that have emerged to prevent piracy and cheating. Unfortunately, some of the new security monitoring approaches have grave privacy implications that require vigilance on the part of gamers.

Defeating Piracy by Going Online

One easy way to prevent simple piracy like copying is not to distribute anything to copy. That is, if a majority of your game resides on a central server, it can't be easily copied. By and large, game companies have adopted this strategy to prevent trivial game cracking (recall the client-server model from Chapter 1). Modern games almost all require gamers to play the game online using only supported servers. These online servers, at the very least, can check a local copy of the game client (running on the gamer's PC) for a legitimate serial number or some other key.

Of course, online games also require an online account, implying that some kind of user or gamer authentication is required to play the game. Note that this is a much clearer way to tie a game to a particular gamer than existed in the previous paradigm. Tracking gamer behavior is an important tactic in the fight against cheating.

As we briefly describe in Chapter 1, gaming is big business. For example, Blizzard Entertainment, the developers of World of Warcraft, not only charge over $30 for the game client but also require a gamer to pay $14 per month to log into the online servers. WoW has over 8 million users all paying these fees. You do the math.

Or Not . . .

Of course, the server model is not completely foolproof. A number of clever developers realized some time ago that it is possible to create new, possibly free, servers for gamers to connect to, thus sidestepping the subscription model. The question is, is this piracy?

When three programmers wrote an open source version of Blizzard's server software called BnetD, Blizzard sued—and won. See Chapter 4 for more.

Tricks and Techniques for Cheating

There are many ways to cheat in an online game. Some of them don't require much in the way of computer programming skills at all. Colluding as a group in an online poker game against an unsuspecting fellow player is an example from the "just takes a telephone" camp. On the other hand, some cheats require deep programming skills.

In this chapter, we'll introduce you to some basic cheating concepts:

- Building a bot
- Using the user interface (UI)
- Operating a proxy
- Manipulating memory
- Drawing on a debugger
- Finding the future

The end results of many of these approaches are now available for purchase online at a number of spurious Web sites. One example is the Pimp My Game Web site at <http://www.pimpmygame.org/>. The Web site, similar to many others like it, boasts the following:

> We give our users the chance to get Exploits, Bots, Hacks, Macros, Patches, Cheats and Guides for all usual MMORPGs and FPS Games that we support. Get them from our own downloads section and forums where you can discuss and debate. You will become more successful in your Game!

Of course, we're more interested in understanding what goes on behind the curtain of these "Exploits, Bots, Hacks, Macros, Patches, Cheats, and Guides" than we are in buying them.

Building a Bot: Automated Gaming

If you Google "online game bots," you'll amass impressive millions of hits. Most of the hits are for sites that offer to sell you a bot. But what is a bot really?

Bots are stand-alone programs that play a game (or part of a game) for you. The term originates from first-person shooter (FPS) games developed for the PC. The term derives from a ro*bot* that simulates another player in the game. You might play a game of chess against a bot, or you might battle a bot in an FPS game like DOOM.

Today, the term *bot* is applied widely to a range of programs, from those as simple as a keyboard mapping that allows you to script together several common actions to those as complex as a player based on artificial intelligence (AI) that plays the game by following simple reasoning rules. In the FPS world, people use bots to perform superhuman actions (e.g., perfect aim). In the MMORPG realm, players use bots to automate the boring parts of play. We provide an example of a macro later in the chapter that controls a character in WoW, thus making that character a bot (temporarily at least).

In all cases, bots perform certain tasks better than humans. Maybe their understanding of chess logic is superior, or maybe they outplay human characters by knowing more about game state than a human can track, or maybe they just do repetitive tasks without getting bored. But whatever they're programmed to do, bots give cheaters an unscrupulous advantage.

Bots have even been used to rob other characters in a game. According to an article in the *New Scientist*:[2]

> A man has been arrested in Japan on suspicion of carrying out a virtual mugging spree by using software "bots" to beat up and rob characters in the online computer game Lineage II. The stolen virtual possessions were then exchanged for real cash. The Chinese exchange student was arrested by police in Kagawa prefecture, southern Japan.

In a slightly less obvious fashion, online poker bots have been used to win poker games for their masters. Though professional-level play is not yet possible (because solving the problem involves creating legitimate AI that can pass the Turing test[3]), poker bots are good enough to win on basic tables with some regularity.[4]

In final analysis, bots have a mixed reputation. Some serious gamers deride them as a cancer ruining games and the gaming industry for everyone. Others see bots as extremely useful tools for delegating the boring aspects of play to a computer program. Still others see bots as a great way to make a living.

Game companies often deploy technical and legal countermeasures to detect and stop bot activity. Sometimes they keep play statistics about characters and notice when certain values go out of range (e.g., flagging

---

2. "Computer Characters Mugged in Virtual Crime Spree," by Will Knight (August 18, 2005; see <http://www.newscientist.com/article.ns?id=dn7865>).
3. For more on the Turing test, see <http://en.wikipedia.org/wiki/Turing_test>.
4. You can find an article from MSNBC about poker bots at <http://www.msnbc.msn.com/id/6002298/>.

things when a character quadruples its wealth in one hour). Another common countermeasure is to ask a character questions to see how humanlike its responses are.[5] The *Korea Times* reports that in the MMORPG Lineage, at least 150 game minders monitor the game for use of bots and then ban players using them. The report states that 500,000 accounts had been suspended between 2004 and April 2006 because of bot activity.[6]

Using the User Interface: Keys, Clicks, and Colors

Games have outstanding UIs these days. Consider the UI from WoW shown in Figure 2–1. For an impressive and diverse collection of UIs for MMORPGs, see <http://xune-gamers.tripod.com/id3.html>.

Figure 2–1 A WoW screenshot, demonstrating the state of the art in online game user interfaces.

5. In this case, the perfect MMORPG bot would need to be able to pass the Turing test.
6. See <http://times.hankooki.com/lpage/culture/200605/kt2006052116201765520.htm>.

As you can see, UIs include parts of the screen that a user can interact with by using standard input devices. There are buttons, text windows, and pictures. You play the game by interacting with the UI—it's your window on what's going on.

Cheaters use the UI to cheat. Let's say a game has three buttons, A, B, and C, that you're only allowed to click manually yourself. By some game companies' definition, if you were to install a software automation tool (such as a quality assurance testing tool) that automatically clicks the mouse on x- and y-coordinates to drive these buttons, you would be cheating.

In many cases, EULA allowances and their associated enforcement mechanisms *restrict how you use the software*. That is, you're allowed to click on buttons yourself, but a program that you write is not. You can learn much more about EULAs in Chapter 4.

Controlling someone's use of the game like this seems rather extreme until you consider the economic impact of automated game play. In most cases, automated game play is realized by using special tools and scripts typically referred to as *macros*. For example, in WoW, monsters appear at specific locations on a periodic basis. You can easily write a macro that causes the in-game character to stand in that location and automatically kill the monster every time it appears (thus gaining experience points and virtual gold). Of course, you can do this manually yourself, waiting around all day for the monster to appear, but given that the monster appears only once every 10 minutes, that plan will commit you to a very long and boring night. Why not write a macro to wait around for you? Ultimately the question is, how can automating such a boring and repetitive activity be considered cheating?

WoW, and many MMORPGs like it, are so afflicted with repetitious game play that the players have invented a term to describe it: *grinding*. That is, doing awful, repetitive things all day with your character just to gain experience is likened to a mule going around and around on a treadmill, grinding grain into flour day in and day out. For some reason, players *enjoy* this self-inflicted misery and will pay $14 a month for the privilege of doing it. Why?

As it turns out, there is deep-seated human psychology at play here, and it has to do with living a double life, as well as the fact that grinding away like this brings economic reward. Whenever you kill that monster, it drops in-game play money and gives you other rewards, such as more experience, skills, and ultimately levels.

If you write a macro to do this grinding for you by manipulating the UI, you can go away to work, or sleep, and come back later and have the sum of all the gold pieces and experience for all the repetitive monster kills waiting for you. Thus, the macro earns you in-game money and simultaneously increases your character's power—but without the associated boredom of actually paying attention. What a great idea! It's so great, in fact, that thousands of players do it all the time. There is even a special term used to describe players who play this way—they are called *farmers*.

The simple bot that we include later in the chapter uses UI manipulation to control a grinding character.

Operating a Proxy: Intercepting Packets

Interacting with a game through the client software by going through the UI is a straightforward cheating technique that is not hard to code. There are many more sophisticated methods, of course. One method involves operating a proxy between the game client and the game server. This proxy can intercept packets and alter them in transit. In other words, a proxy-based cheating scheme carries out what is in security circles known as an attacker-in-the-middle attack[7] (Figure 2–2).

There are many ways to carry out an attack like this. Monitoring the network wire is one way. Getting between a program and the system dynamic link libraries (DLLs) it is using is another. Basically, any place where messages are passed around by the target program is susceptible to this kind of interpositioning.

Proxy attacks have a long history. Some of the first network-based proxy attacks were devised and used against FPS games. In these games, a fair amount of data about game state is passed around between the client software and the server. Sometimes these data are not displayed for the player to see, but they are available to the software the player is using. A proxy cheat sniffs the network packets, analyzes them, and adjusts various parameters that should not be known by the player. A classic example comes from the FPS game Counter-Strike, where proxy cheats have been used to improve aim drastically (an essential characteristic in the shoot-'em-up world).

Proxy-based cheats in FPS games are usually held very close to the vest. That's because those who use them are interested in evading detection even

7. This kind of attack is most often called a man-in-the-middle attack, but we find that terminology sexist.

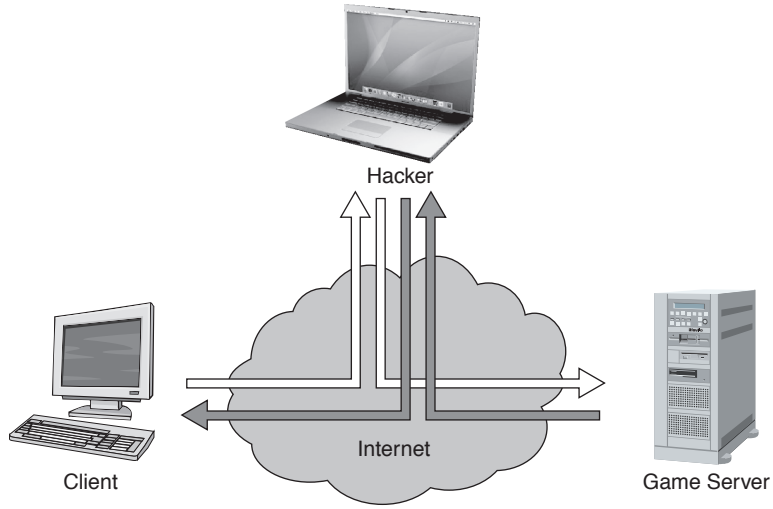

Figure 2–2 A picture of an attacker-in-the-middle attack. In this picture, the hacker interposes between the client and the server and can both monitor and manipulate traffic as it goes either direction.

in complicated social situations. Being outed at a LAN party for a visually detectable cheat could lead to bodily harm of the meat-space variety! Proxy-based aimbots are thus carefully designed to be effectively used in social situations and may involve statistical fuzzing of calculations to simulate not-quite-perfect aim.[8]

Sophisticated proxy-based attacks attempt to change data as they move between the server and the client. An obvious countermeasure is to encrypt the data so that unintelligible gibberish is all that can be seen going by. However, encryption costs cycles, and the balance is payable in a less-realistic gaming experience that is a major resource hog. Security always trades off against something; it never comes for free.

To use a proxy attack, cheaters configure the proxy with the address of the server they are using. In most cases, the FPS client runs on another machine entirely and connects through the proxy to the server. This situation is very similar in nature to a standard network sniffing situation, the only difference being that packets can be manipulated as they go by.

The real trick is being able to construct a useful model of the game from the traffic going by. The model will do things like track player locations.

8. See the history of aimbot cheating in the game Counter Strike on the Wikipedia site at <http://en.wikipedia.org/wiki/Cheating_in_Counter-Strike>.

When a packet with a marked command such as "fire the truly impressive blunderbuss of flatulence" goes by, the proxy software determines who is where and who must die and makes it so. It may need to do some weapon movement to cause this to happen so that the blunderbuss is properly aimed. (Remember, the weapon and/or player movement needs to seem natural and reasonable—a complete back flip with a triple gainer followed by a shot between the eyes may be a tip-off that something is fishy.)[9]

Proxy attacks are just as useful for non-FPS games as well. In fact, the most sophisticated attacks against MMORPGs use a very similar idea, interposing on machine state through manipulation of interrupts. You can read more about those attacks in Chapter 6.

Manipulating Memory: Reading and Writing Data

Manipulating game state through interposition as described in the previous subsection is a hands-off cheating technique. The game software is not directly manipulated, only its input and output are. More hands-on techniques get into the game program itself, reading and writing memory, changing values, and generally messing around with game state directly in the software.

One obvious target for manipulation is the drivers that control graphics rendering on the PC. Once again, FPS games led the way in cheating. The Counter-Strike game is a classic FPS. Like many games, it makes extensive use of drivers to control graphics, sound, and networking. Cheaters target the OpenGL and Direct3D drivers to do their dirty work. The Counter-Strike hack called XQZ was one of the first to alter drivers. The creators of Counter-Strike built an anticheat engine called Cheating-Death that was able to detect when OpenGL drivers had been replaced. The arms race was on.

Historically, the reason that drivers are easy to manipulate is that Microsoft Windows doesn't include an easy way to determine whether a driver is legitimate or nefarious. All of this is changing with the release of Vista and its driver signing capability (though don't believe for a minute that this is a security silver bullet). Prior to Vista, finding these cheats on any given computer was difficult even when full access was provided to the machine (and, in fact, even when rootkit technology was involved).

Another common form of client-side manipulation involves the use of classic hooking techniques to interpose on DLLs and other system libraries.

---

9. For more on proxy attacks in FPS games, see <http://www.gamasutra.com/features/20000724/ pritchard_pfv.htm>.

Many of these techniques are in common use in software exploits (see our book *Exploiting Software* for more). All modern operating systems use runtime-loadable libraries (Win32 uses DLLs; Linux and Mac OS X use standard UNIX-shared object libraries such as `glibc.so`), and as such, all are susceptible to hooking attacks.

In some games, client-side communication is accomplished through DLL calls, all of which are easily intercepted. Modern software almost always makes extensive use of outside libraries, which in this particular case leads directly to the risk of rampant cheating.

Once again, an arms race of sorts has emerged, with classic hacks (such as the `loaddll` library for Counter-Strike[10]) leading to hook-proof anticheats leading to sneakier hooks leading to hook detection mechanisms.

Drawing on the Debugger: Breakpoints

Debuggers are another common attack tool used by those who exploit software. There's nothing quite as powerful as a kernel-level debugger with the ability to stop processes in mid stride. Many cheaters use debugger techniques to set breakpoints or other triggers. These triggers can look for certain messages (e.g., a secret key press) or the use of particular functions (e.g., an easily thwarted DLL) and then carry out various nefarious activities.

Modern attacks of the sort that we describe in Chapter 6 often involve the use of sophisticated interrupt-driven state manipulation. More on that later.

Finding the Future: Predictability and Randomness, or How to Cheat in Online Poker

Many games involve an element of chance, poker being among the most obvious. The problem is that producing unpredictable randomness is nontrivial given the way computers work. Much work in software security has gone into improving randomness (which as you might imagine is also necessary for good cryptography).

Here's a good example of how poor pseudorandom number generators (PRNGs) can be broken in practice.[11] In 1999, the Software Security Group at Cigital discovered a serious flaw in the implementation of Texas hold 'em

10. See <http://www.answers.com/topic/cheating-in-counter-strike>.
11. This attack was first described in *Building Secure Software* by John Viega and Gary McGraw (Addison-Wesley, 2001). The text here is adapted from that account.

poker distributed by ASF Software, Inc. The exploit allowed a cheating player to calculate the exact deck being used for each hand in real time. That means a player who used the exploit knew the cards in every opponent's hand as well as the cards that made up the flop (cards placed face up on the table after rounds of betting). A cheater could "know when to hold 'em and know when to fold 'em" every time. A malicious attacker could use the exploit to bilk innocent players of actual money without ever being caught.

The flaw exists in the shuffling algorithm used to generate each deck. Ironically, the shuffling code was publicly displayed in an online FAQ with the idea of showing how fair the game is to interested players (the page has since been taken down), so it did not need to be reverse engineered or guessed.

In the code, a call to `randomize()` is included to reseed the random number generator with the current time before each deck is generated. The implementation, built with Delphi 4 (a Pascal IDE), seeds the random number generator with the number of milliseconds since midnight according to the system clock. That means the output of the random number generator is easily predicted. As it turns out, a predictable random number generator is a very serious security problem.

The shuffling algorithm used in the ASF software always starts with an ordered deck of cards and then generates a sequence of random numbers used to reorder the deck. In a real deck of cards, there are 52! (approximately 2^{226}) possible unique shuffles. The seed for a 32-bit random number generator must be a 32-bit number, meaning that there are just over 4 billion possible seeds. Since the deck is reinitialized and the generator reseeded before each shuffle, only 4 billion possible shuffles can result from this algorithm, even if the seed had more entropy than the clock. Yet 4 billion possible shuffles is alarmingly less than 52!.

The flawed algorithm chooses the seed for the random number generator using the Pascal function `randomize()`. This particular `randomize()` function chooses a seed based on the number of milliseconds since midnight. There are a mere 86,400,000 milliseconds in a day. Since this number was being used as the seed for the random number generator, the number of possible decks now reduces to 86,400,000—alarmingly less than 4 billion.

In short, there were three major problems, any one of which would have been enough to break the system.

- The PRNG algorithm used a small seed (32 bits).
- The PRNG algorithm used was noncryptographic.
- The code was seeded with a poor source of randomness (and, in fact, reseeded often).

The system clock seed gave the Cigital group members an idea that reduced the number of possible shuffles even further. By synchronizing the program with the system clock on the server generating the pseudorandom number, they were able to reduce the possible combinations down to a number on the order of 200,000 possibilities. After that move, the system is broken, since searching through this tiny set of shuffles is trivial and can be done on a PC in real time.

The tool Cigital developed to exploit this vulnerability requires five cards from the deck to be known. Based on the five known cards, the program searches through the few hundred thousand possible shuffles and deduces which one is a perfect match. In the case of Texas hold 'em poker, this means the program takes as input the two cards that the cheating player is dealt, plus the first three community cards that are dealt face up (the flop). These five cards are known after the first of four rounds of betting and are enough to determine (in real time, during play) the exact shuffle.

Figure 2–3 shows the GUI for the exploit. The Site Parameters box in the upper left is used to synchronize the clocks. The Game Parameters box in the upper right is used to enter the five cards and initiate the search. The figure is a screenshot taken after all cards have been determined by the program. The cheating attacker knows who holds what cards, what the rest of the flop looks like, and who is going to win in advance.

Once the program knows the five cards, it generates shuffles until it discovers the shuffle that contains the five known cards in the proper order. Since the `randomize()` function is based on the server's system time, it is not very difficult to guess a starting seed with a reasonable degree of accuracy. (The closer you get, the fewer possible shuffles you have to look through.) After finding a correct seed once, it is possible to synchronize the exploit program with the server to within a few seconds. This *post facto* synchronization allows the program to determine the seed being used by the random number generator and to identify the shuffle being used during all future games in under one second.

The ASF poker software was particularly easy to attack. However, most uses of linear congruential PRNGs (such as `rand()`)are susceptible to this kind of attack. These attacks always boil down to how many outputs an

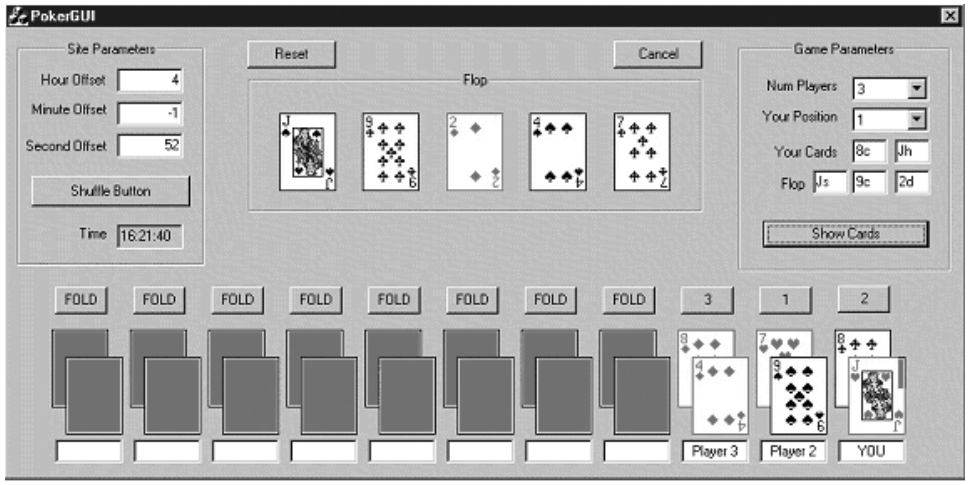

Figure 2–3 Cigital's Internet poker exploit can predict the outcome of a poker hand by taking advantage of a broken shuffling algorithm.

attacker must observe before enough information is revealed for a complete compromise. Even if you only select random numbers in a fairly small range, thus throwing away a large portion of the output of each call to `rand()`, it usually doesn't take very many outputs to give away the entire internal state.

Many games other than poker make use of randomness. Any game that is completely predictable can be won by a simple program that knows how to predict what will happen next! As more and more money is poured into gaming, these kinds of randomness problems turn into cash cows for attackers.

The Bot Parade

Since bots are such pervasive cheaters' tools, it is worth spending some time pondering a few examples. Three large-scale categories of bots are combat macro bots (used to cheat in MMORPG games), aimbots (a particular type of combat macro bot used commonly in FPS games), and poker bots.

Combat Macro Bots

Many games, especially MMORPGs, have simple scripting languages that can be used to interact with the game. These languages allow a number of basic activities to be strung together into a macro. Sometimes games don't

include a scripting language off the shelf, but third-party programs that run with the game provide scripting capability.

Scripts are very useful for automating simple tasks. Here is an example of a macro that works with WoW.[12] Web sites filled with macros like this one are easy to find on the Internet.

Macro
TargetFreeEnemy **Last Updated: June 22nd, 2006**

Updated since last WoW Patch

This macro cycles through all enemies around you and stops when it hits an untapped enemy (non-faction, i.e., non-horde and non-alliance). Very useful while grinding undead currently! (Stops after cycling through 30 enemies as to not hang your computer when none are available.) (This macro assumes you're Alliance, if you're not, replace "`Horde`" with "`Alliance`" to make it work for you as well.)

Code Start:

```
/script TargetNearestEnemy(); local i = 1;
if(UnitExists('target')) then while(UnitExists('target') and
(UnitFactionGroup('target') == 'Horde' or
UnitIsTapped('target')) and i < 30) do TargetNearestEnemy();
i = i+1; end;end;
```

Code End:

Here is another example. This macro works with Star Wars Galaxies.[13]

Date Added: August 2, 2004

Ok, this is an AFK combat macro designed for the in-game macro system. If you want to take it a step further, you can run a HAM Checker macro (checks your HAM and heals you when needed) with AC Tool.

To run this macro, set up a droid running a patrol pattern in a square pattern around a spawn. Angle your camera in so it faces the spawn (and go watch tv or sleep.

The macro is:

```
/ui action targetGroup0;
/follow;
```

12. The WoW macro is from <http://ui.worldofwar.net/listmacros.php?type=2>.
13. The Star Wars Galaxies macro is from <http://rpgexpert.com/2789.html>.

```
/ui action cycleTargetOutward;
(if you're a Brawler, insert /follow here)
/attack;
(insert specials here)
/pause XXX;
(amount of time to kill it)
/follow;
/pause 2;
/loot;
/harvest hide;
/macro AFKCombat;
```

This macro is slightly more interesting than the first example because it works unattended. In fact, its simple description makes this clear. Some gamers draw a big distinction between attended use of macros and unattended use. They believe that unattended use is cheating.

Some games go so far as to outlaw the use of macros (see Chapter 4 for more on legal moves to prevent cheating). The game Asheron's Call includes the following legalese:

> Use of unauthorized third-party software or macros with the Software may be prohibited in the sole discretion of Turbine. Specifically, you may not use third-party software which allows your character to gain experience points or items by engaging in combat without being at the keyboard, ready to respond to Turbine staff on demand (this activity is commonly called a "Combat Macro"). Logging off as soon as an admin appears (visible or invisible) or when an admin tries to speak with you will be taken into consideration in determining the use of Combat Macros.[14]

One of the most well-known and widely used macro systems in gaming is Lin2Rich <http://lin2rich.com/>. Lin2Rich is actually an advanced bot environment. It works alongside your client (i.e., you use it in tandem with the regular game client), sending packets directly to the game server according to the way it is configured. Lin2Rich can be configured to send a set of commands to the server when a certain event happens. For example, if you click certain buttons at particular times or begin to rest at a particular time and place, you can program the cheating client to send particular commands to the game. Because of the way it works, detecting Lin2Rich by monitoring player behavior is difficult.

---

14. You can find more on the Asheron's Call legalese at <http://turbine.fuzeqna.com/asheronscall/consumer/kbdetail.asp?kbid=305>.

Aimbots

Aimbots are bots used in FPS games. The idea is simple: Aim better than your opponent and win. Superhuman aim achieved through programming is generally considered cheating (though some FPS games have built-in aimbots that can be used).

Aimbots come in a variety of flavors. Some only aim. Some aim and shoot. Some move, aim, and shoot. The first aimbots were created for Quake and Counter-Strike. An interesting FAQ about Unreal Tournament and cheats can be found at <http://www.digdilem.org/ut/aimbot.php>.

Black Hat Corner: ZelliusBot Readme File

========ZelliusBot UT2004 Edition ver 1.0==========

—About—

My leet aimbot with lots of useful features :D Works on UT2004 Demo & UT2004 retail ver. 3186. My bots feature base is aimed towards ease of tracking down crybabies and huge-ego clanmembers :)

—Installation—

1. Copy ZelliusBot.u & Zelliusbot.ini to your UT's 'System' folder
2. Open up your UT2004.ini, navigate to the [XInterface.ExtendedConsole] and edit the ServerInfoMenu so its value is: ZelliusBot.ZelliusServerInfo. It should look like ServerInfoMenu=ZelliusBot.ZelliusServerInfo. If Serverinfo line aint already there, just add it.
3. Open up your User.ini and bind 'serverinfo' to a key of your choice. e.g., z=serverinfo
4. Wait until you've joined a game and press your serverinfo key, if all goes well the Zelliusbot status should appear on your HUD.

—Usage—

Use your number row keys to toggle various commands. You can change your selected toggling keys by editing your ut2004.ini statements like AutoTauntKey.

You can get the various number codes for each key by setting bShowKeyNumbers to True, joining a game and taking note of the numbers that are displayed when you press a certain key.

—Notes—

This bot will not aim while in UT2004 vehicles

Compile in the UT2003 development environment.

Experiment in game or read the uscript to figure out how everything works.

If you encounter a 'shooting at wall' problem when an enemy comes into sight increase your FireDelayTime.

Unload the bot when the match is ended, if you still get a invalid game file error on map change type 'reconnect' in the console.

—Thanks—

Thanks go to tenbucks, moonwolf, play2win, lamer, spunky and many others for lots of testing and feedback.

Thanks go to [ELF]HelioS for his help with ping code, HUD and his textures (from helios ut bot).

Thanks go to DrSiN[Epic] for leaving multiple gaping holes in UT2004.

I hope you have lots of fun!

~Zellius

One popular aimbot called ZelliusBot works by reprogramming initialization files in Unreal Tournament. The Black Hat Corner includes some text from the readme file that comes with ZelliusBot.

In most cases, aimbots come packaged with other features such as the ability to travel through walls. One early aimbot for Counter-Strike was XQZ. XQZ combined several capabilities into an easy-to-use package. It was eventually released to the public. XQZ at first replaced and then later hooked the OpenGL DLL. XQZ included features that allowed users to use it stealthily during LAN parties without detection. Counter-Strike is still filled with spinoffs from XQZ.

In its first instantiations, the XQZ aimbot was invoked when a button on the keyboard or mouse was pressed. Pressing the button resulted in the aimbot controlling the aiming crosshairs.

Care must be taken when developing and using aimbots to ensure that their superhuman potential does not become a giveaway. For example, using an aimbot that perfectly tracks a strangely moving adversary, shooting it over and over in the exact same spot, is called slaving. Decoupling the aim and shoot features is sometimes necessary. As we mentioned earlier, more advanced aimbots can add statistical noise to their actions.

Poker Bots

Now that online poker has topped $1 billion in revenue, it's not surprising that much activity has been invested in creating automated bots to play poker. Fortunately, poker is a reasonably hard game that can't easily be automated. However, poker bots do exist, and they are improving rapidly.

Of course, if poker bots get too good, they will undermine the game itself. Nobody wants to lose consistently to a bot! Especially when money is involved.

Rumors abound of a new generation of sophisticated poker bots that can beat newbies with some regularity. Some even say that above-average players can be beaten as well. Of course, just as in other kinds of gaming, the game companies want to discourage the use of bots. They make use of player monitoring and look for suspicious patterns of play, sometimes adapting their games to defeat bots.

Still, some people are making their living by selling poker bots commercially. A number of programs are available at the WinHoldEm Web site <http://www.winholdem.net/>, ranging from a $25 standard package of hand analysis software to a $200 version that includes bot play capability.

Once again we find ourselves in the midst of a classic arms race. Simple bots pop up, antibot measures out them, the bots improve, and so on. Of course, most online casinos don't allow bots, but enforcing that rule is difficult.

The entire online poker phenomenon is confusing to many security professionals. The biggest problem is collusion between players. Cheating by discussing hands in an out-of-band conversation is very difficult to detect. Banning this activity is easy from a legal perspective, but enforcing the ban may not be possible.

Some believe the inability to control poker bots and out-of-band communication will be the demise of online poker. We'll just have to wait and see.

Lurking (Data Siphoning)

Much valuable gaming data can be gleaned simply by watching other players play the game and learning how they behave. This is true for sports, of course, where watching your opponents play in order to understand their

play is an invaluable aid. The same kind of technique works for online gaming, from actions in an MMORPG to hands in online poker.

Online Statistics

Services like Thottbot help users collect and use statistics about the game.[15] For example, Thottbot can tell you exactly where to find the vorpal sword of heinosity, provide you with a map to it, and let you know what your chances are of obtaining it once you're there. Thottbot works by sucking up as much information as it can from cooperating gamers and then republishing that information after it has been properly organized. For more on Thottbot, see Chapter 3.

Poker Statistics

Online poker is just as big as if not bigger than MMORPGs, though the recently passed legislation in the United States will put a big crimp in the market.

A number of third-party vendors create and sell software packages to help analyze hand history data and build a database of information about players and their tendencies. Serious poker players use these statistics to check for weaknesses in their play and uncover weaknesses in the play of others. Using these tools is extremely common, and all serious players (including an entire class of professional online poker players) use them.

As usual, academics and mathematicians have entered the fray. One interesting paper titled "Mathematical Statistics and Online Poker" by Jason Swanson can be found at <http://www.math.wisc.edu/~swanson/instructional/stats_poker.pdf>. Be forewarned, though—this paper includes real math!

Figure 2–4 shows the GUI from a typical online poker third-party application. This application helps a player understand poker statistics and what to do next. It generates statistics, win percentages, hand probabilities, and useful tactical information for the game of Texas Hold 'em Poker.

In any game that involves money, it should be clear that your adversary will tool up. Online poker cheats and stats trackers are destined to become much better over time. Perhaps one day the bots will be good enough to beat even the best humans consistently.

---

15. Thottbot <http://www.thottbot.com> is an online database of WoW information.

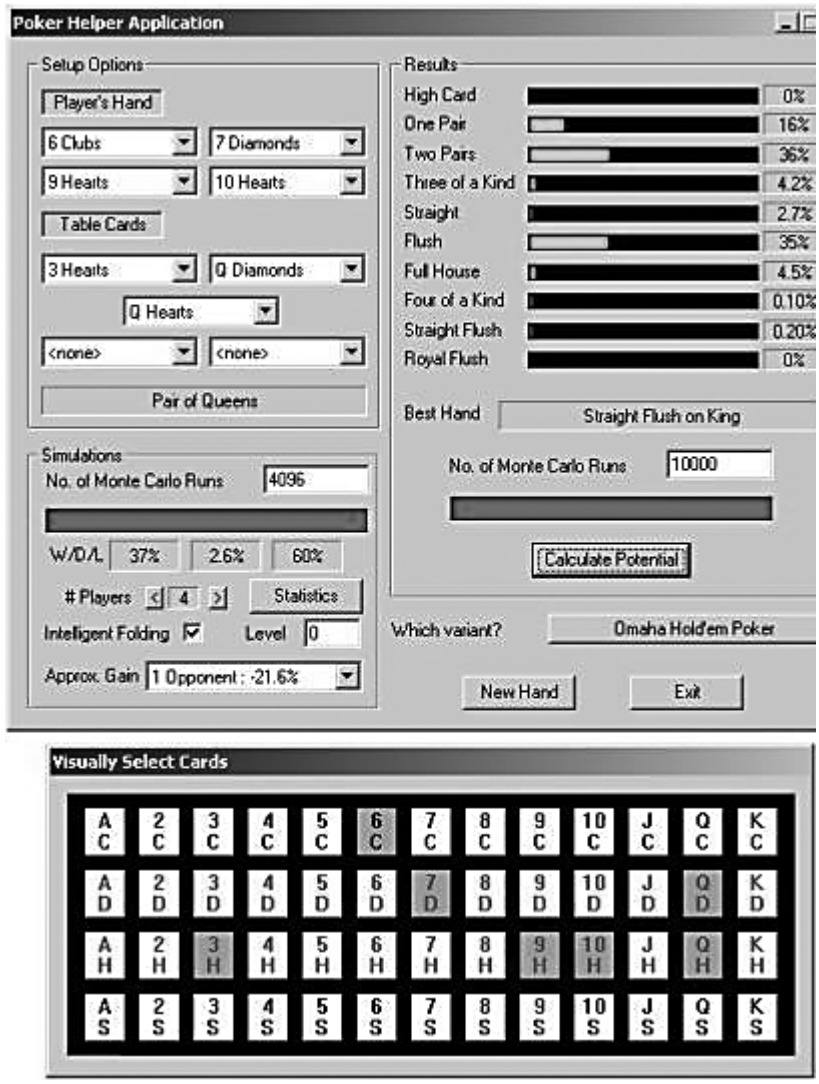

Figure 2–4 An online poker helper application. (From <http://www.frayn.net>; reproduced with permission.)

Auction Manipulation

Cheaters also like to cheat in auctions. Though only tangential to online gaming, online auctions share many of the same "instant riches" lure that online games do. Considering some of the tactics that cheaters in online auctions resort to may provide some insight on cheaters in general.

Probably the most obvious tactic in auction manipulation is shill bidding. The idea behind shill bidding is to place a bid on an item only to inflate the final value. Of course, it's also against the law and a felony in the United States (you see, shill bidding existed long before online auctions). Auction houses like eBay track IP numbers to try to defeat shill bidding. They also monitor bidding activity over time to look for suspicious patterns. Sound familiar?

Another common cheating technique in online auctions is interfering in a transaction through out-of-band communication. This can take place through e-mail or any other channel. Colluding in an auction can be just as unfair as colluding in a poker game, and just as hard to detect.

A third form of cheating involves interposing near the end of an auction to try to intercept payment. By simply dashing off a quick e-mail to the winner as if the attacker were the seller and asking for payment, the attacker can sometimes dupe the poor winner into paying the wrong person. Traceable payment systems help make this attack less prevalent than others.[16]

Finally, there's a way to cheat in head-to-head auctions, applicable when things begin to heat up at the end. The competing parties (A and B) may be bidding against each other for the last few minutes, when A carries out an attack to deny service to B. One simple technique involves A attempting to log in as B unsuccessfully several times in a row so that B is temporarily locked out of the account.

Tooling Up

By now we have described a number of common "game cheating 101" attacks. We've even shown you how simple macros work. But explaining how these things work is not quite the same as seeing how they are constructed. Toward that end, we present a simple macro for WoW. *Warning*: Use of this macro is cheating and is against the rules. Your character may be banned from the game if you use this macro.

Note that these kinds of tools are in common use every day in all kinds of games. It's not just in WoW that cheating happens.

16. For more on online auction cheating, see <http://www.dummies.com/WileyCDA/DummiesArticle/id-2679.html>.

AC Tool: Macro Construction

There are a number of popular macro construction tools on the Internet. These include macro express (which costs money), AC Tool, AutoHotKey, AutoIt3.0, LTool-0.3, and xautomation. All of the ones mentioned here have been nicely collected on a wiki at <http://wiki.atitd.net/tale2/Macros>.

AC Tool is particularly popular. You can download it for free from <http://www.actool.net>. Once you have installed AC Tool, you can create macros for WoW using its macro language. The FAQ on the AC Tool Web site describes the tool as follows:

> AC Tool is a utility that allows you to list a series of keystrokes and mouse clicks in advance and send them to Asheron's Call at a later time. The list of keystrokes and mouse clicks is called a macro or a script.

Here is a simple grinding/farming macro (called Hoglund's WoW_Agro Macro) designed to be used with AC Tool. This macro controls a character that camps out waiting for monsters to appear and kills them when they do. We'll interleave commentary with the code so you can understand what's happening.

```
// ------------------------
// hoglund's WoW_Agro Macro
// ------------------------
//RESOLUTION: 1024x768
//PUT ALL FILES IN YOUR AC TOOL\MACROS FOLDER.

SETActiveWindow World of Warcraft
delay 4 sec

// put all your 'globals' here
Constants
       gPCHealth = "NoValue"
       // the current health of the PC ( HIGH | MEDIUM | LOW |
CRITICAL )
       gMobHealth = "NoValue"
       // the current health of the current targeted mob ( HIGH |
MEDIUM | LOW | CRITICAL )
       gPCPosture = "Standing"
       // current/starting posture of the PC character
       gMachineState = "START"
       // global machine state, core of the system
       gSelectedTarget = "NoTarget"
       pc_name = xanier
```

The first part of the code sets up global variables, most importantly the currently selected target, the name of the character being automated, and the current state of the state machine. This script implements a classic architecture that many macros follow—that is, it is designed to operate as a state machine. This means that the script maintains a single state variable while automating the character. States can have mnemonic labels such as "attacking," "healing," or "running away for dear life." Only one state can be active at any one time, and there are rules for transitioning from one state to another.

```
// hot keys
keyExecAttack = 1
// set your F1 key to attack before using this MACRO
keyExecPickup = {F2}
```

The part of the script above indicates which keys need to be used in order to automate the game. When the F2 key is pressed, the character picks up items, and when the number 1 key is pressed, the character attacks. This kind of key binding is very common in macros like this. Simple macros operate the software using only keystrokes and mouse movements—there is nothing invasive that exploits the game software directly. More advanced methods of cheating are nowhere near as simple.

```
// screen coordinates
coord_MobHPMin            = 262, 50
coord_MobHPFull           = 370, 50

coord_SafeHP              = 200, 50
//if green at or above this mark, you're fine
coord_HalfHP              = 146, 50
//half hp if lower than this mark
coord_LowHP               = 116, 50
//low hp if at or lower than this mark
coord_DeadHP              = 96, 50
//you're dead :(
coord_temp                = 0,0

// colors
color_AliveGreen          = 150
color_AttackRed           = 147
```

```
        // temp stuff
        TempTarget                = NoValue
        //holds the value for target before placing into list
        tCount                    = 1
        //used to traverse the targetList index
        Total                     = 1
        //used to find the ListCount
        redDifference             = 0
        blueDifference            = 0
        greenDifference           = 0
        numAttacks                = 0
        Returned                  = 0
        aCount                    = 0

End
```

The section of the script above is quite fascinating. It designates x- and y-coordinates on the screen. Furthermore, it designates exact color thresholds for pixels. This information is used to sample pixels on the screen at precise x- and y-coordinates. The pixels in question happen to correspond to locations where health and enemy health are displayed on the screen for the game user. This information, along with the color threshold, is then used to determine whether the character is at full health, is partially damaged, or is in critical condition—all via sampling pixels on the screen and figuring out color matches while the macro runs.

```
////MAIN LOOP
While 1=1
  Processmessages    //required for AC Tool operation
  Call Lazy8_Main    //the main lazy8 machine handler
End
///END MAIN LOOP

//////////////////////////////////////////////////
// the main lazy8 machine handler
//////////////////////////////////////////////////
Procedure Lazy8_Main

        //KeyDown /whisper $pc_name Lazy8_Main {RETURN}
        //KeyDown /whisper $pc_name current state: $gMachineState
{RETURN}
```

You see this sort of thing all over the place in the macro; the `whisper` command prints a message on the screen and is used to output debug

messages while the macro is under development. You can see the results of a `whisper` call in the screenshot in Figure 2–5.

```
        // perform effective switch() on machine state
        //
        // START
        if $gMachineState = "START"
        Call LazyStart
        end

        // IDLE
        // bored, we need to do something
        if $gMachineState = "DRAWING"
        Call LazyDraw
        end

        // END
        // game over, man
        if $gMachineState = "END"
        call LazyEnd
        end
End //end Lazy8_Main

Procedure LazyStart
        KeyDown /stand {RETURN}
        delay 100
        // go directly to draw agro mode
        Set gMachineState = "DRAWING"
End

Procedure LazyDraw
        KeyDown /whisper $pc_name LazyDraw {RETURN}

        //Call LureIfNoMonster

        //target last agro, this targets nearest also
        KeyDown {TAB}
        Delay 400

        Call EnsureAttackMode

        // hero strike
        KeyDown 2
        Delay 400

End
```

In the above section of the macro, we perform a pixel test to determine whether the character is currently in "agro" mode—that is, attacking a monster. If not, we call a routine to target the nearest monster and begin attacking. The script presses the 2 key in order to perform a hero strike—a powerful form of attack in the game.

```
// run forward and back to lure agro
Procedure Lure
       KeyDown {UP} 3 sec
       KeyDown {DOWN} 3 sec
End

// put the PC into attack mode
Procedure EnsureAttackMode
       LoadRGB 34,64
       Compute $redDifference = Abs ( $color_AttackRed -
{RGBRED} )
       if $redDifference < 80
                KeyDown /whisper $pc_name im currently in attack
mode {RETURN}
       else
                KeyDown /whisper $pc_name im attempting to start
attack mode {RETURN}
              KeyDown $keyExecAttack 150
       end
End

// perform a random move if no monster is targeted
Procedure LureIfNoMonster
       Call util_GetMobHealth
       if $gMobHealth = "DEAD"
                KeyDown /whisper $pc_name Rest in Peace! .. luring
{RETURN}
                Call Lure
       else
                KeyDown /whisper $pc_name still going... {RETURN}
       end
End
```

The section of the macro above is used to "draw agro" from nearby monsters. Essentially this amounts to getting the monster's attention by moving nearby. Such movement causes the monster to attack the character. Once the attack occurs, the character can attack the monster and kill it.

```
////////////////////////////////////////////////////
// END - kill the engine
////////////////////////////////////////////////////
Procedure LazyEnd
       KeyDown /whisper $pc_name LazyEnd {RETURN}

       //TODO
End

////////////////////////////////////////////////////
// Utility Function
// Get target mob health
////////////////////////////////////////////////////
Procedure util_GetMobHealth
       //KeyDown /whisper $pc_name util_GetMobHealth {RETURN}

       LoadRGB $coord_MobHPMin
       Compute $greenDifference = ABS ( $color_AliveGreen -
{RGBGREEN} )

       //KeyDown /whisper $pc_name sample monster {RGBRED}
{RGBBLUE} {RGBGREEN} {RETURN}

       if {RGBRED} = 0 AND {RGBBLUE} = 0 AND $greenDifference < 80
              Set gMobHealth = "ALIVE"
              //KeyDown /whisper $pc_name monster is still alive
{RGBGREEN} {RETURN}
       else
              // we never found any green
              Set gMobHealth = "DEAD"
              KeyDown /whisper $pc_name monster is dead {RETURN}
      end

End
```

This section of the macro detects whether or not the target monster is dead yet. Like monitoring character health, this is done through the simple technique of sampling pixels. In this case we pull a sample at the location that displays the monster's health bar.

And that's it, a complete macro for automating the attraction and killing of monsters in WoW. By running this macro, a character can accumulate experience and gold without human intervention.

Figure 2–5 shows a screenshot taken while the WoW_Agro macro is running. Notice the pile of dead monsters near the character. You should

Figure 2–5 A screenshot from WoW showing what happens when the WoW_Agro macro runs. Virtual bodies pile up as the macro runs all night long.

note that running a macro like this is against the rules and can cause you to be banned from the game. The WoW character's name displayed on the screen, Xanier, is no secret. Blizzard banned Hoglund's Xanier account just before the character was to reach level 60 (which at the time was the ultimate level for a WoW character). In this case, Hoglund was a little too flagrant with his cheating. Hoglund had spent over $400 on game-card registered accounts by this point. All of the accounts were banned for various reasons.

Time for another pesky question: Why is farming with a macro cheating? Some would argue that Blizzard made the game boring in the first place, so fair is fair.

Countermeasures

For obvious reasons, gaming companies are most interested in keeping bots out of games. Scientists have even begun to study the problem and publish work about it. For example, the paper "Preventing Bots from Playing

Online Games" by Philippe Golle and Nicolas Ducheneaut appears in *ACM Computers in Entertainment*.[17]

One clever idea suggested for ensuring that humans (rather than bots) are playing games has been used successfully to combat spam. It's called a reverse Turing test. The Turing test is a test from the AI field meant to determine whether the entity at one end of a conversation is a human. A reverse Turing test works by asking questions to ensure that the answering target (in this case, a gamer) is a human. In many cases, this involves displaying a convoluted picture that only a human can properly decode (Figure 2–6). This technique is often referred to as *captcha*, short for Completely Automated Public Turing Test to Tell Computers and Humans Apart.

Often, gamers self-police and try to ferret out and expose bots among themselves. This can happen through the invocation of a game's chat function (something that game masters also employ as a tactic to find bots). The chat countermeasure works in many games, from MMORPGs to online poker. Poker bots are not very good at small talk!

Monitoring in real time (possibly using chat) is often accompanied by monitoring history as well. Looking for behavior that is nonhuman in nature—maybe superhuman aim, or maybe obsessive behavior such as killing monsters over and over in the same spot—is often a tip-off that a bot may be involved. Game masters make use of history to uncover cheating as a standard tactic.

Figure 2–6 A classic reverse Turing test based on human perception. (From <http://www.brianpautsch.com/Blog/2005/12/1/Captcha>; used by permission.)

17. The paper is also available from the PARC Web site at <http://www.parc.xerox.com/research/publications/files/5445.pdf>.

Spyware

Anticheat tools have been around almost as long as game servers have. Public outcry on open servers led to the first such tools. Game companies running private servers carry on the tradition to this day.

The first service devoted to this is called PunkBuster, run by a Texas company named Even Balance, Inc. The PunkBuster service started in 2000. The first versions relied on simple techniques such as variable checking and process validation authorized through a server. This was followed by a number of server-based checks. Today, PunkBuster is integrated into a number of online games.

The PunkBuster Web site lists the following "features" of the product.[18]

- Real-time scanning of memory by PB Client on players' computers searching for known hacks/cheats
- Throttled two-tiered background auto-update system using multiple Internet Master Servers to provide end-user security ensuring that no false or corrupted updates can be installed on players' computers
- Frequent status reports (highly encrypted) are sent to the PB Server by all players and the PB Server raises a violation when necessary which causes the offending player to be removed from the game and all other players are informed of the violation
- PB Admins can also manually remove players from the game for a specified number of minutes or permanently ban if desired
- PB Servers can optionally be configured to randomly check player settings looking for known exploits of the game engine
- PB Admins can request actual screenshot samples from specific players and/or can configure the PB Server to randomly grab screenshot samples from players during gameplay
- An optional "bad name" facility is provided so that PB Admins can prevent players from using offensive player names containing unwanted profanity or racial slurs
- Search functions are provided for PB Admins who wish to search player's keybindings and scripts for anything that may be known to exploit the game

18. Quoted from the PunkBuster Web site at <http://www.evenbalance.com/index.php?page=info.php>.

- The PunkBuster Player Power facility can be configured to allow players to self-administer game servers when the Server Administrator is not present entirely without the need for passwords
- PB Servers have an optional built-in mini http web server interface that allows the game server to be remotely administered via a web browser from anywhere over the Internet

Note that some of these features are extremely invasive, such as the one that scans client PCs! This kind of scanning often involves the installation of spyware on a gamer's PC.

PunkBuster was originally conceived as a way to stop client-side hacks in Counter-Strike. But today PunkBuster concentrates on helping to protect and police other games.

As it turns out, PunkBuster was a harbinger of invasive techniques to come.

The Warden: Defeating Cheaters by Crossing the Line

Today, WoW has a two-pronged strategy to defeat cheaters. The first approach is to make rules against cheating and ban those characters caught cheating. This is carried out by publishing the terms of use, which are administered according to the legally binding EULA. Nothing is wrong with that.[19] (For more on the legal situation, see Chapter 4.) The second approach is to keep an eye on every PC running the WoW client and try to determine whether it is being used to cheat. This spying thing is a problem.

If monitoring someone's PC sounds like spying to you, that's because it is. WoW's embedded Warden reads all sorts of data from the gamer's PC. Hoglund's Warden discovery in fall 2005 set off a controversy over privacy, security, and just what lengths security measures should go to in order to stop cheating. And it's part of the reason that we decided to write *Exploiting Online Games*.

Besides monitoring the WoW process space and keeping track of DLLs running in that space, the Warden pokes around into other processes, doing things like reading the window text in the title bar of *every window* and doing a scan of the code loaded for *every process running* on your computer (which it then compares against known cheat code). Blizzard claims not to

19. Note that this is not a law, it is a contract, and there is no due process. Game companies can ban your character at any time.

have any designs to use the data it digs up for purposes other than security, but nothing is really stopping the company from doing whatever it wants on a gamer's PC, and it has already crossed the invisible line by poking around outside the game's process area. We don't trust them.

This is a clear invasion of privacy. So much so that the Electronic Frontier Foundation (EFF) has even weighed in with an opinion <http://www.eff.org/deeplinks/archives/004076.php>. Though the EULA does call out that Blizzard may monitor PC activities with the Warden (without specifying what the Warden actually does), this information is buried in the small print that almost nobody ever reads. In informal conversations with WoW players, we have found that none of them were aware that they had agreed to such monitoring. Some of them were concerned enough to stop playing.

Worldwide, 8 million people play WoW (usually about 500,000 at any one time). And *every single one* of these players has granted Blizzard the right to scan their computer programs and memory—which Blizzard does.

The Warden process scans for an arbitrary list of things such as open processes, URLs, and so on, controlled at Blizzard's discretion, and mails the information it finds back to Blizzard any time the game program is running. Figure 2–7 shows an example of content being sent back to Blizzard by the Warden as determined by a program we wrote called the

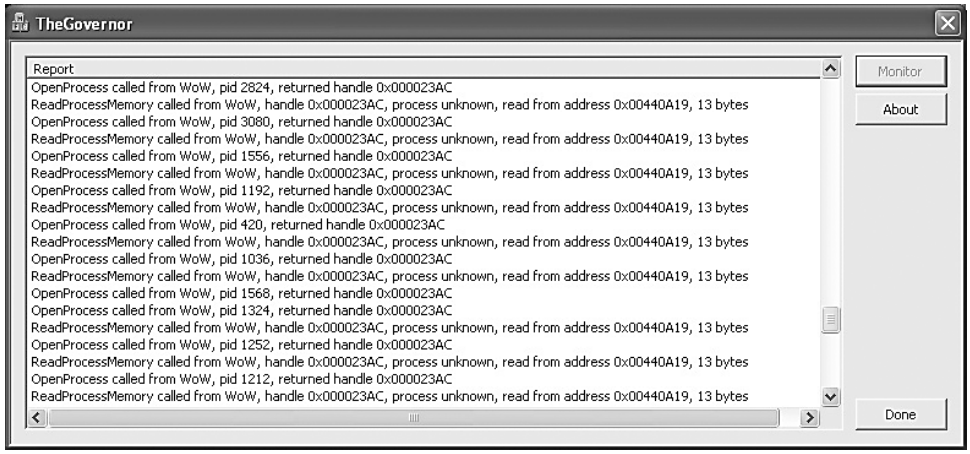

Figure 2–7 The Governor program, which we created, reports on the activities of the WoW Warden. The Warden acts as a spy for Blizzard and keeps an eye on your PC by calling into open processes and reading memory to see what it finds, then reporting that information back to Blizzard for analysis.

Governor. You can think of the Warden as 8 million copies of what the game companies call "legitimate spyware." We think the term *spyware* is apt without a modifier.

Of course, Blizzard isn't the only company that does monitoring like this—many other game companies do it, too. Sony even tried to install a monitor when music CDs were inserted into a PC but did it so poorly that the company landed in hot water. Reaction to the unmasking of programs like the WoW Warden and Sony's rootkit has been intense.

The Black Hat Corner reprints a blog entry outing the Warden as spyware. This story spread worldwide.

Black Hat Corner: WoW Warden as Spyware

In the fall of 2005, one of us (Hoglund) discovered the WoW Warden and outed it to the technical community. Hoglund's original posting to the rootkit.com blog (available at <http://www.rootkit.com/blog.php?newsid=358> with follow-up threads and discussion) touched off a firestorm. Here's the post.

4.5 million copies of EULA-compliant spyware
Oct 05 2005, 19:07 (UTC+0)

Hoglund writes:

I recently performed a rather long reversing session on a piece of software written by Blizzard Entertainment, yes—the ones who made Warcraft, and World of Warcraft (which has 4.5 million+ players now, apparently). This software is known as the 'warden client'—it's written like shellcode in that it's position independent. It is downloaded on the fly from Blizzard's servers, and it runs about every 15 seconds. It is one of the most interesting pieces of spyware to date, because it is designed only to verify compliance with a EULA/TOS. Here is what it does, about every 15 seconds, to about 4.5 million people (500,000 of which are logged on at any given time):

The warden dumps all the DLL's using a ToolHelp API call. It reads information from every DLL loaded in the 'world of warcraft' executable process space. No big deal.

The warden then uses the GetWindowTextA function to read the window text in the titlebar of every window. These are windows that are not in the WoW process, but any program running on your computer. Now a Big Deal.

I watched the warden sniff down the email addresses of people I was communicating with on MSN, the URL of several websites that I had open at the time, and the names of all my running programs, including those that were minimized or in the toolbar. These strings can easily contain social security numbers or credit card numbers, for example, if I have Microsoft Excel or Quickbooks open w/ my personal finances at the time.

Once these strings are obtained, they are passed through a hashing function and compared against a list of 'banning hashes'—if you match something in their list, I suspect you will get banned. For example, if you have a window titled 'WoW!Inmate'—regardless of what that window really does, it could result in a ban. If you can't believe it, make a dummy window that does nothing at all and name it this, then start WoW. It certainly will result in warden reporting you as a cheater. I really believe that reading these window titles violates privacy, considering window titles contain a lot of personal data. But, we already know Blizzard Entertainment is fierce from a legal perspective. Look at what they have done to people who tried to make BNetD, freecraft, or third-party WoW servers.

Next, warden opens every process running on your computer. When each program is opened, warden then calls ReadProcessMemory and reads a series of addresses—usually in the 0x0040xxxx or 0x0041xxxx range—this is the range that most executable programs on windows will place their code. Warden reads about 10–20 bytes for each test, and again hashes this and compares against a list of banning hashes. These tests are clearly designed to detect known 3rd party programs, such as wowglider and friends. Every process is read from in this way. I watched warden open my email program, and even my PGP key manager. Again, I feel this is a fairly severe violation of privacy, but what can you do? It would be very easy to devise a test where the warden clearly reads confidential or personal information without regard.

This behavior places the warden client squarely in the category of spyware. What is interesting about this is that it might be the first use of spyware to verify compliance with a EULA. I cannot imagine that such

> practices will be legal in the future, but right now in terms of law, this is the wild wild west. You can't blame Blizz for trying, as well as any other company, but this practice will have to stop if we have any hope of privacy. Agree w/ botting or game cheaters or not, this is a much larger issue called 'privacy' and Blizz has no right to be opening my excel or PGP programs, for whatever reason.
>
> —Greg

The Governor

Hoglund was upset enough by the Warden to write a program called the Governor that gamers can use to determine exactly what the Warden is doing.

We believe that having the Governor around is useful, especially if you're interested in what WoW software might be doing on your PC. A listing of part of the Governor is included here. You may download a copy of the software and the libraries required to build it from the book's Web site or at <http://www.rootkit.com/vault/hoglund/Governor.zip>. As we did earlier, we have interspersed the code with commentary to make it easier to understand.

```
// The Governor
// Oct 16, 2005 - Greg Hoglund
// www.rootkit.com

#include "stdafx.h"
#include "GovernorDLL.h"
#include <windows.h>
#include <stdio.h>
#include <stdarg.h>
#include <process.h>

HANDLE g_hPipe = 0;
CRITICAL_SECTION g_pipe_protector;

void PatchFunctions();
```

The code here is used in a DLL—a type of code object that can be loaded directly into a program. In this case, the WoW program is *forced* to load this DLL via a process known in software security parlance as *DLL injection*. The injected DLL is actually just a normal DLL, but it is forced to load after the WoW game client has already started and logged into the WoW server.

```
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved )
{
       switch (ul_reason_for_call)
       {
       case DLL_PROCESS_ATTACH:
              InitializeCriticalSection(&g_pipe_protector);
              g_hPipe = StartPipe("\\\\.\\pipe\\wow_hooker");
              SendText(g_hPipe, "GovernorDLL Loaded.");
              PatchFunctions();
              break;
       case DLL_THREAD_ATTACH:
              break;
       case DLL_THREAD_DETACH:
              break;
       case DLL_PROCESS_DETACH:
              SendText(g_hPipe, "GovernorDLL Unloaded.");
              ShutdownPipe(g_hPipe);
              break;
       }
       return TRUE;
}
```

The DLL opens a named pipe—a form of communication port—when it is loaded. Later, the named pipe will be used to report on the Warden's activity.

```
//
// Send text down the pipe
//
void
SendText (HANDLE hPipe, char *szText)
{
        if(!hPipe) return;
```

```
 char *c;
 DWORD dwWritten;
 DWORD len = strlen(szText);
 DWORD lenh = 4;

 EnterCriticalSection(&g_pipe_protector);

 // send length first
 c = (char *)&len;
 while(lenh)
 {
       //char _g[255];
       //_snprintf(_g, 252, "sending header %d", lenh);
       //OutputDebugString(_g);

       if (!WriteFile (hPipe, c, lenh, &dwWritten, NULL))
       {
              LeaveCriticalSection(&g_pipe_protector);
              ShutdownPipe(hPipe);
              return;
       }
       lenh -= dwWritten;
       c += dwWritten;
}

// then string
c = szText;
while(len)
{
       //char _g[255];
       //_snprintf(_g, 252, "sending string %d", len);
       //OutputDebugString(_g);


       if (!WriteFile (hPipe, c, len, &dwWritten, NULL))
       {
              LeaveCriticalSection(&g_pipe_protector);
              ShutdownPipe(hPipe);
              return;
       }
       len -= dwWritten;
       c += dwWritten;
}
```

```
        LeaveCriticalSection(&g_pipe_protector);
}

HANDLE StartPipe(char *szPipeName)
{
        HANDLE hPipe;
        TCHAR szBuffer[300];

        //
        // Open the output pipe
        //
        hPipe = CreateFile (szPipeName, GENERIC_WRITE, 0, NULL,
                        OPEN_EXISTING, FILE_FLAG_WRITE_THROUGH,
                        NULL);
        if (hPipe == INVALID_HANDLE_VALUE)
        {
                _snprintf (szBuffer, sizeof (szBuffer),
                     "Failed to open output pipe(%s): %d\n",
                     szPipeName, GetLastError ());
                     OutputDebugString(szBuffer);
                return NULL;
        }

        return hPipe;
}

void ShutdownPipe(HANDLE hPipe)
{
        //cleanup
        if (hPipe)
        {
                FlushFileBuffers (hPipe);
                CloseHandle (hPipe);
        }
        // make sure it stops being used
        g_hPipe = 0;
}
```

Some of the Warden hooking files are displayed below in order to enhance your understanding of what's going on. Remember, we're only showing you the essential parts of the Governor system here.

```
// The Governor
// Oct 16, 2005 - Greg Hoglund
// www.rootkit.com

#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <winsock2.h>
#include "GovernorDLL.h"
#include "detours.h"
#include <psapi.h>

//////////////////////////////////////////////////////////////////////
// These are functions used by the Warden to spy on other processes
//////////////////////////////////////////////////////////////////////
```

The functions below are all used by the WoW Warden. Here, we use the public Microsoft *Detours* library to intercept calls made to these functions.[20] This move allows us to report on any program that attempts to use these calls, including the Warden, through our named pipe.

```
DETOUR_TRAMPOLINE(BOOL __stdcall Real_GetWindowTextA( HWND hWnd,
LPSTR lpString, int nMaxCount), GetWindowTextA);
BOOL __stdcall Mine_GetWindowTextA( HWND hWnd, LPSTR lpString, int
nMaxCount)
{
	int len = Real_GetWindowTextA( hWnd, lpString, nMaxCount);
	if( len != 0)
	{
		// WoW found some window text, let's report it
		char _t[255];
		_snprintf(_t, 252, "GetWindowTextA called from WoW,
returned %d bytes, %s ", len, lpString);
		SendText(g_hPipe, _t);
	}

	return len;
}
```

20. You can find out more about the Microsoft Detours library and download the code from <http://research.microsoft.com/sn/detours>.

```
DETOUR_TRAMPOLINE(BOOL __stdcall Real_GetWindowTextW( HWND hWnd,
LPWSTR lpString, int nMaxCount), GetWindowTextW);
BOOL __stdcall Mine_GetWindowTextW( HWND hWnd, LPWSTR lpString,
int nMaxCount)
{
       int len = Real_GetWindowTextW( hWnd, lpString, nMaxCount);
       if( len != 0)
       {
              // WoW found some window text, let's report it

              char _t[255];
              _snprintf(_t, 252, "GetWindowTextW called from WoW,
returned %d bytes", len);
              SendText(g_hPipe, _t);
       }

       return len;
}
```

Now we can identify when the Warden is opening windows on the computer, even windows that belong to other programs (e.g., your instant messaging program). The Governor reports not only which window the Warden opened but also what text it read. This technique has been used to watch the Warden program read sensitive and presumably private information, including the e-mail addresses of the contacts in your instant messenger program.

```
DETOUR_TRAMPOLINE(int __stdcall Real_WSARecv(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
  LPDWORD lpNumberOfBytesRecvd,
  LPDWORD lpFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine),
WSARecv);

int __stdcall Mine_WSARecv(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
```

```
  LPDWORD lpNumberOfBytesRecvd,
  LPDWORD lpFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)
{
       int res = Real_WSARecv(s, lpBuffers, dwBufferCount,
                              lpNumberOfBytesRecvd,lpFlags,
                              lpOverlapped,lpCompletionRoutine );
       char _t[255];
       _snprintf(_t, 252, "WSARecv returned %d, %d bytes
received", res, *lpNumberOfBytesRecvd);
       SendText(g_hPipe, _t);

       return res;
}

DETOUR_TRAMPOLINE(DWORD __stdcall Real_CharUpperBuffA( LPTSTR
lpString, DWORD cchLength), CharUpperBuffA);
DWORD __stdcall Mine_CharUpperBuffA( LPTSTR lpString, DWORD
cchLength)
{
       DWORD len = Real_CharUpperBuffA( lpString, cchLength );
       if( len != 0)
       {
              // WoW is processing some text, let's report it

              char _t[255];
              _snprintf(_t, 252, "CharUpperBuffA called from WoW,
string %s", lpString);
              SendText(g_hPipe, _t);
       }

       return len;
}

DETOUR_TRAMPOLINE(HANDLE __stdcall Real_OpenProcess( DWORD
dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId ),
OpenProcess);
HANDLE __stdcall Mine_OpenProcess( DWORD dwDesiredAccess, BOOL
bInheritHandle, DWORD dwProcessId )
{
       HANDLE h = Real_OpenProcess( dwDesiredAccess,
bInheritHandle, dwProcessId );
```

```
	if( h != 0)
	{
		// WoW is opening a process, let's report it

		char _t[255];
		_snprintf(_t, 252, "OpenProcess called from WoW, pid
%d, returned handle 0x%08X", dwProcessId, (DWORD)h);
		SendText(g_hPipe, _t);
	}

	return h;
}

DETOUR_TRAMPOLINE(BOOL __stdcall Real_ReadProcessMemory(
				HANDLE hProcess,
				LPCVOID lpBaseAddress,
				LPVOID lpBuffer,
				SIZE_T nSize,
				SIZE_T* lpNumberOfBytesRead ),
ReadProcessMemory );

BOOL __stdcall Mine_ReadProcessMemory( HANDLE hProcess, LPCVOID
lpBaseAddress, LPVOID lpBuffer, SIZE_T nSize, SIZE_T
*lpNumberOfBytesRead )
{
	BOOL ret = Real_ReadProcessMemory( hProcess,
lpBaseAddress, lpBuffer, nSize, lpNumberOfBytesRead );
	if( ret && ((DWORD)hProcess != -1) )
	{
		// WoW is reading a process, let's report it
		char szProcessName[MAX_PATH] = "unknown";


		GetProcessImageFileName(hProcess, szProcessName,
MAX_PATH);


		char _t[255];
		_snprintf(_t, 252, "ReadProcessMemory called from
WoW, handle 0x%08X, process %s, read from address 0x%08X,
%d bytes",
				hProcess,
				szProcessName,
				lpBaseAddress,
				nSize);
```

```
                SendText(g_hPipe, _t);
        }

        return ret;
}
```

The last two functions above leave no guesswork as to Warden activity. They clearly report whenever the Warden opens another process and it reads the memory of that process. Our spy versus spy system is complete.

```
void PatchFunctions()
{
        DetourFunctionWithTrampoline( (PBYTE)Real_GetWindowTextA,
(PBYTE)Mine_GetWindowTextA);
        DetourFunctionWithTrampoline( (PBYTE)Real_GetWindowTextW,
(PBYTE)Mine_GetWindowTextW);
        DetourFunctionWithTrampoline( (PBYTE)Real_CharUpperBuffA,
(PBYTE)Mine_CharUpperBuffA);
        DetourFunctionWithTrampoline( (PBYTE)Real_OpenProcess,
(PBYTE)Mine_OpenProcess);
        DetourFunctionWithTrampoline( (PBYTE)Real_ReadProcessMemory,
(PBYTE)Mine_ReadProcessMemory);

        //DetourFunctionWithTrampoline( (PBYTE)Real_WSARecv,
(PBYTE)Mine_WSARecv);
}
```

This last part of the code actually installs the "detours" we defined against the selected functions. You can easily modify the source code to monitor additional calls, and you can apply this code against any program you want, including other games. We encourage you to experiment with this functionality. Perhaps you will discover other games and applications that are spying on users.

Where Do You Stand?

Online MMORPG forums and Web sites have been abuzz with the spyware controversy ever since Hoglund described the Warden. Some people believe that complaining about such spyware is silly and that if someone is so worried about it, he or she can just choose not to play. We disagree. We've also heard the opinion that maybe the blame belongs to Microsoft since its operating system allows arbitrary programs to collect the kind of

information the Warden collects. What we worry about is the intention here. Regardless of whether you *can* invade privacy, should you?

What Blizzard is doing in the name of security is unacceptable, and it needs to stop. The tradeoff between personal liberty and security is an essential tradeoff that must be carefully negotiated. Citizens in a free society must guard their freedoms vigilantly or risk tyranny in the name of security. Historically, monitoring activities lead very quickly to abuse and present an unacceptably slippery slope. Blizzard does not need to read our e-mail, surf our URLs, or look into our non-Blizzard processes. Allowing the company to do so gives up too much ground for not enough return. Once everyone is doing this, there will be no more privacy on a "personal" computer.

Hoglund says it best in an open message to Blizzard posted at <http://www.rootkit.com/newsread.php?newsid=371>:

> Blizzard, it is within your right to attempt to make your computer game the way you wish it to be, and to attempt to catch cheaters. But, reading the memory of other processes and windows that are not part of the World of Warcraft game client is a violation of privacy. Making a violation of privacy legal in your EULA and TOS does not make it also moral. It remains a violation of privacy. Please refactor your policy in regards to scanning memory, and limit the warden to integrity checking of the game client's memory space, and please stop opening other processes and reading windows that do not belong to you.

If we stand by and let a game company poke around on our PCs in the name of security, what do you suppose they will do next?

Cheating

As we have shown in this chapter, there are many widespread ways to cheat in online games. Some of the techniques we describe are several years old. As more and more money is at stake, more gamers are likely to be tempted to cheat. Cheating is bad. Cheating technology is improving.

Big questions remain regarding anticheating technology, though. Are back door programs, rootkits, and invasive scanners really necessary for piracy detection? When does such a countermeasure cross the line between legitimate copy protection and invasion of privacy? What rights to your computer must you give up in order to play an online game?

Consider, for example, that Blizzard's Warden scanning program isn't really used to detect pirates—it's really used to detect cheaters. Blizzard, of

course, gets to define precisely what it means by cheating—it is their game, after all. Among other things, cheating in this case means any use of software automation tools like the WoW_Agro macro we showed in this chapter to play the game.

It seems that the war on cheating has led to collateral damage in the form of privacy. Yet we have only begun to describe what's happening in the gaming world.