A Case Study in Helping Students to Covertly Eat Their Classmates

Roya Ensafi, Mike Jacobi, and Jedidiah R. Crandall
University of New Mexico,
Dept. of Computer Science
{*royaen, crandall*}*@cs.unm.edu; mikerjacobi@gmail.com*

Abstract

Werewolves is an online version of the game *Werewolves of Miller's Hollow* that we developed in 2012 to help teach information flow in a computer security and privacy class. The game pits werewolves against townspeople in a shared Linux system, where students must use the command line environment to find information flow leaks in the form of side channels that reveal the werewolves' identities.

Werewolves has many desirable traits, such as the ability to make learning about information flow fun and the fact that the kinds of attacks students can carry out to gain an advantage in the game are open ended, which leads to self-guided learning. However, these benefits quickly deteriorate if one or two students dominate the game. In this paper, we discuss instances where this has occurred through several uses of the game, and propose ways to ameliorate this problem.

1 Introduction

In 1961 programmers at Bell Labs created Darwin [14], a programming game where competing programs were loaded into an arena and dueled in a survival of the fittest type scenario. The game is no longer played because several programs were developed which were too dominant. A similar game called Core War [8], released in 1984, attracts players of all skill levels to compete in tournaments and develop their skills to this day. Why do some games devolve into one-sided, non-competitive exercises in futility while others evolve into competitive games where players learn together and stay engaged for long periods of time?

In Ensafi *et al.* [9] we described a game called Werewolves, an online version of *The Werewolves of Miller's Hollow* (a variant of *Mafia*). From that paper: "Werewolves is a party game in which participants are divided into two groups: werewolves and townspeople. The goal of the game is to eliminate members of the opposing group. Werewolves can eat townspeople at night. They are outnumbered but know each others' identities. Townspeople vote for who to hang as a suspected Werewolf each day. Werewolves act as normal townspeople during the day, so they are part of this discussion and voting process. While townspeople are given an advantage in their numbers, they do not know the identities of the werewolves among them so they must rely on inference."

Since the publication of that paper we have gained more experience with using Werewolves in two classes and two series' of club meetings[1]. While Werewolves has been a positive and educational experience for students in general, vulnerabilities discovered in the server and decisive tactics developed by students have led to situations where one or two students (or one or two groups) dominate the game. This paper describes the different cases where this has occurred and discusses future plans and proposed strategies for avoiding these scenarios in the future via careful development of Werewolves.

The learning objective of Werewolves depends on the group of students. For younger groups (*e.g.*, high school) and more general groups of students, the objective is for the students to become adept at using the Linux command line and to understand a wide variety of hacking and vulnerability concepts, including side channels. For upper-level and graduate courses on Cybersecurity the objective is for students to be able to understand concepts in the research literature about information flow and side channels and be able to explore the system in a systematic way to discover new side channels.

In this paper, we describe our experiences using Werewolves for a variety of learning tasks. Specifically, the learning objectives above have not been fully realized because a typical Werewolves game consists of one student (or group) dominating the game by easily identifying all of the werewolves. This makes the game less fun for the other players, and disincentivizes them from developing new tactics (*e.g.*, searching for new side channels

or developing mitigations to hide their activities when they are werewolves) because they do not know how the dominant player or group is so readily identifying the werewolves.

These scenarios arise because of two different issues: server vulnerabilities and decisive tactics. Server vulnerabilities are software bugs in the Werewolves server that reveal the werewolves' identities in ways that the werewolves cannot counter. For example, if the server logs the werewolves' identities in a file that is supposed to be non-world-readable and a vulnerability in the way the server handles that file makes it possible for townspeople to read the file, then no strategy adopted by the werewolves can stop their identities from being revealed until the server bug is fixed. Decisive tactics, on the other hand, can be countered by the werewolves but only if the werewolves understand them. For example, if a group is able to observe network socket activity, then other groups could possibly counter by using different TCP socket options when they log in or developing a custom UDP-based client—but his can only happen once they realize that TCP socket information is how their classmates are able to identify them.

These two separate issues share some characteristics, but they are distinct in some ways, as well. In both cases, mitigations are necessary until the problem can be addressed. For example, students who are dominating the game because they can easily find the werewolves can be forced to be werewolves for several games. However, while server vulnerabilities can be fixed by the moderators to level the playing field, decisive tactics may require significant effort by other students to develop an effective counter strategy. Our hope is that through reading this paper that describes our experiences with Werewolves the Cybersecurity gaming community can help us make progress towards understanding what can make a game like Werewolves have continued engagement from all students.

The focus of Werewolves is *side channels.* In our online version (please refer to Ensafi *et al.* 2012 [9] for more details) the role of the fortune teller (*i.e.*, the player in the traditional Werewolves game that can gain partial information each night about which players are Werewolves) is removed so that only through side channels in the Linux operating system can the townspeople infer the identities of the werewolves. A covert channel is a flow of information that violates a written or implied information flow policy. Covert channels exploit communication channels that were not intended for communication by the system's designers. Typically, with covert channels it is assumed that the sender and receiver of the information are in collusion. Specifically, the source of the information is a subject that wants to transmit the information through the channel. Side channels [16, 23, 10, 19, 24, 2, 20, 6, 18, 12] are different in that the secret information is accidentally modulated into timing, storage, or resource usage of the system in a way that the receiver of the information can *infer* information that they should not have access to.

This paper is organized as follows. Section 2 discusses vulnerabilities in the Werewolves server, followed by Section 3 that discusses decisive tactics. Discussion and potential future work are in Section 4, followed by the conclusion in Section 5.

2 Vulnerabilities in Server

In this section, we describe vulnerabilities in the server that made it possible to discover the werewolves in a way that the werewolves could not counter[2]. This happens because the server reveals the werewolves' identities. The descriptions of all vulnerabilities found in the Werewolves source code so far are in Table 1. We note that this source code was developed in a matter of weeks, mid-semester, so we only had time to add functionality and no time for code review or testing with respect to security properties. In fact, we hoped that vulnerabilities would exist in version 1.0 of the Werewolves code because it would give students in the class something to find.

One vulnerability was identified and fixed before Werewolves was used or released: the pipe `stat()` vulnerability. This is classified as a vulnerability and not a decisive tactic because the werewolves must interact with the server through their assigned named pipe, so without fixing this issue the werewolves would have no way of hiding writes to this pipe. Another vulnerability was known, but not fixed in all cases in versions 1.0 and 1.1 of the server: the pipe transitions vulnerability.

Another vulnerability was discovered independently by two students in the original semester when Werewolves was used in class: the server process creation vulnerability. This led to two groups in the class dominating every game of Werewolves, and was a major motivation for us to subsequently be more proactive about finding and fixing vulnerabilities in the server. Also, for us it underscored the need for mechanisms to mitigate vulnerabilities and decisive tactics until they can be dealt with in a more effective way.

The server race condition vulnerability was discovered when the capture-the-flag team ENOFLAG in Germany played Werewolves and gave us feedback. In this vulnerability, a moderator-only log file was created and then permissions were set to keep players from reading it. The moderator's log file contained the identities of the werewolves. By opening the file after creation but before permissions were changed, a player could gain read access to the file and know the identities of the werewolves. Be-

Vulnerability name	Description	Discovered by
Pipe `stat()`	In Linux, stat()ing a named pipe reveals the last time the pipe was accessed. If users are able to `stat()` the named pipe of all players in the game, they can know when the werewolves' read and write pipes are being used through changes in the last access time. Version 1.0 fixed this before Werewolves was used or released, by putting each named pipe in its own protective directory (that other users could not `stat()` files in).	Authors
Pipe transitions	The scheduler state of processes is visible to all users. By monitoring what processes are in the `pipe_wait` state players can know every time the server sends a message to particular users. In some cases in versions 1.0 and 1.1 the server wrote a message to the werewolves only and did not send a message to other players at the same time, revealing the identities of the werewolves. This is fixed in version 1.2.	Matthew Hall
Server process creation	To write into a player's pipe, version 1.0 of the server would fork a bash process with the command "`echo "Message" > /path/to/pipe`". Since the arguments of all processes in the system are visible to all users, this revealed all messages the server sent to any player. This was fixed in version 1.1 by opening the file and writing to it from the Python process.	Alex Woody and Brian Lott
Server race condition	Upon creating the log file for each game, the file was first `touch`ed and then `chmod`ed in version 1.0. Any user who opens the file in between these operations and keeps it open can still read the log file after the `chmod`. This was fixed in version 1.1.	Jakob Lell and the ENOFLAG team
Various command injection vulnerabilities	Because inputs were passed to a bash shell process unfiltered, characters such as `;`, `|`, `&`, and `` ` `` could cause the server process (running with moderator privileges) to execute arbitrary commands injected by players. An attempted fix was made in version 1.1 through string sanitization, but did not stop all command injections. Version 1.2 opens the pipes, writes to them, and flushes the output, rather than invoking a shell command in a child process.	Geoff Reedy, Jeffrey Knockel, Geoff Alexander, and Stephen Harding
Multiple votes	Due to a logic error, the server counts the total number of votes to make sure it does not exceed the number of players, but allows players to vote multiple times. This vulnerability will be fixed in the soon-to-be-released version 1.3.	Stephen Harding
Voting ties	Due do a logic error in the server source code, a tie that results in a vote leads to the werewolves' identities being printed in the world readable log file. This vulnerability will be fixed in the soon-to-be-released version 1.3.	Stephen Harding

Table 1: Server vulnerabilities.

cause this was a classic filesystem race condition vulnerability and relatively easy to exploit, we added the ability to leave the vulnerability in place into the Werewolves configuration file. Several other vulnerabilities that have been fixed can be re-enabled.

Various command injection vulnerabilities existed because the server would fork a bash shell to interpret a string as a command, such as "`echo $1 > /path/to/pipe`", where `$1` is unsanitized input from one of the players. This allowed students to run arbitrary code as the moderator user, and to then carry out attacks such as copying the moderator's private log file (which included the identities of all werewolves) into their own home directory, for example. The first attempt to fix this vulnerability was to filter out characters that could cause a command to be broken up into multiple, independent commands. This failed and students were still able to find ways to exploit the vulnerability. The vulnerability was ultimately fixed by simply opening files or pipes, writing to them, and then flushing the output buffer, without using any child processes such as bash.

Two vulnerabilities existed in versions 1.0 through 1.2 of the Werewolves source code, but will be fixed in the soon-to-be-released version 1.3. The multiple votes vulnerability exists because, when collecting the vote for which player to hang, the server counts the total votes to make sure they do not exceed the number of living players, but does not count the number of votes any one player casts. So by quickly casting all the votes a single player can control who is hanged each day. This vulnerability is actually more useful to the werewolves than to the townspeople, but in any case it led to a situation where one player dominated the game and the other players benefited less from playing Werewolves. The voting ties vulnerability is a logic error where any vote that results in a tie causes the werewolves' identities to be printed into a world-readable log file.

The vulnerabilities discovered so far in Werewolves can be divided into three classes: server information that is made visible to users other than the moderator (Pipe `stat()`, Pipe transitions, and Server process creation), classic vulnerabilities (Server race condition and Various command injection vulnerabilities), and logic errors (Multiple votes and Voting ties). Logic errors and classic vulnerabilities can be addressed in traditional ways, such as code reviews and sanitization of untrusted inputs. Server information that is made visible to users other than the moderator is a class of vulnerability that is particular to Werewolves. There is an ongoing effort to create a set of test cases that have slight differences (*e.g.*, who the werewolves are) where two traces through the game with slightly different configurations should be indistinguishable by users who are not `moderator` or `root`. By viewing the two respective system call traces it can be determined if any externally visible differences in the system state will be created based on secret information. This is based on the idea of noninterference [11], and will help to identify server vulnerabilities and fix them systematically.

3 Decisive tactics

In this section, we describe decisive tactics for revealing the identities of the Werewolves that are not due to vulnerabilities in the server. This presents a very different challenge from server vulnerabilities.

The TCP socket info tactic is discussed in our CSET paper [9] and was known to us before the first use of Werewolves in a class. Here we give more details. For SSH sessions, it is possible to infer keystrokes and other network activity in a way that is specific to individual network sockets by looking for resets in the TCP keepalive timers for each socket:

```
while : ; do netstat -an --timer | grep ":22" \
| grep "(0" | grep -v LISTEN; sleep 0.1; done
```

This bash command repeatedly lists all TCP timers for sockets connected to the SSH server on the system, and filters these to list only those that have been recently reset (*i.e.*, the most significant digit in seconds since the reset is 0 indicating that the reset happened less than one second ago). Since tying user names to IP addresses is relatively easy (*e.g.*, `who --ips`), the TCP socket info tactic allows one to fairly accurately infer all the keystrokes of all other players. It could potentially be mitigated by creating a custom UDP socket, playing with TCP socket options of the SSH connection, or obfuscating which IP belongs to a user by forking new processes and using indirect interprocess communication. However, the TCP socket tactic, as far as we know, has never been used by any group to dominate in the Werewolves game.

The context switches tactic is also discussed in the CSET paper [9], and is typically the first example of a covert inference channel attack that we show to students. It has not led to any players dominating the game because it is too noisy of a signal in practice when many players are playing. Many players are typing commands or taking measurements during the night phase even if they are not werewolves, so that all players make a lot of system calls at night.

Although the client ID was technically a server vulnerability, we classify it as a decisive tactic because it did not directly reveal the werewolves' identities. Version 1.0 of the server and client allowed clients to specify their identity, and this was not checked by the server. A group exploited this to print messages such as, "I am a werewolf" using the identities of other users. This also never led to any player dominating the game.

Vulnerability name	Description	Discovered by
TCP socket info	The timeout state of every TCP socket is visible to all users of a system in Linux. Thus every keystroke through the terminal SSH session to every player's shell is visible.	N/A
Context switches	Every write to a pipe, keystroke, *etc.* appears as a voluntary context switch in one of the player's processes.	N/A
Client ID	The server did not perform checks on the inputs from players to make sure that they had not modified their client to hide/spoof their identity.	Matthew Areno
`pty` permissions	Every keystroke updates the access and modification times of the `pty` associated with a user's shell. Since the `pty` is visible to all other users this reveals keystroke timings.	Geoff Reedy

Table 2: Decisive tactics.

The `pty` permission tactic was discovered during a code review, but the student who discovered it never had a chance to use it in a game. It would probably allow a player to dominate the game, but it is also fairly easy to fix. To hide information leaks through the `pty` a player should SSH into the server with `ssh -T` to disable pseudo-tty allocation. This creates a shell that is difficult to use, but prevents other players from inferring every keystroke. Another possibility might be for players to run a script that `touch`'s their `pty` with an exponential distribution. This particular decisive tactic is a good opportunity to introduce the ideas of Kang and Moskowitz, who describe a pump for reducing information flow that has an interesting information theoretic basis [13]. It can also be an opportunity to talk about different ideas of information flow that go beyond non-interference [21, 3, 4, 7, 15, 17], and ways to tolerate/mitigate information flow [1, 5, 22].

4 Discussion

From our experiences with both server vulnerabilities and decisive tactics, we make the following observations:

- So far, only server vulnerabilities have led to situations where one player completely dominated the game. Some decisive tactics have the potential for this to occur, but it has not occurred yet.
- For both server vulnerabilities and decisive tactics, mitigation techniques are needed to keep the game competitive until there has been enough time to fix the vulnerability or help students overcome a decisive tactic.
- In both cases, mechanisms for the moderators to discover and understand the vulnerability or tactic are key.

Next, we discuss mitigation strategies to keep the game competitive in the presence of vulnerabilities that are being exploited or decisive tactics that are allowing one player to dominate the game. Then we discuss potential surveillance and forensic mechanisms that can be used by the moderator to gain information about server vulnerabilities and decisive tactics.

4.1 Mitigation

One potential mitigation that we implemented in Werewolves version 1.1 but have not yet used is to randomize names, so that names in the game cannot easily be tied to usernames on the system. If a vulnerability reveals the usernames of the werewolves, then other players must still use inference channels to tie this username to a name in the game. This might have been useful in the first four out of the seven vulnerabilities in Table 1, and three out of four decisive tactics in Table 2. This can also make it challenging for the student dominating the game to convince the other townspeople about who the werewolves are, because their identity changes in every game and other players (especially the werewolves) may pretend to be the student who knows how to find the werewolves.

A mitigation strategy that we implemented in version 1.1 and have used extensively is the ability for the moderator to choose who the Werewolves will be at the start of each game. This was sometimes useful, because players who knew methods for finding the werewolves were forced to be the werewolves and all players were put back onto relatively equal footing. We did not employ this strategy in the long-term, so it did not become necessary to combine this with the randomized name mitigation technique from above, but in the future that might be useful. For the multiple votes vulnerability, forcing the player who was exploiting this vulnerability to be a werewolf did not help because he simply used the vul-

nerability to hang townspeople at will and win the game twice as fast (by eating one townsperson each night and hanging one each day).

We have also experimented with various changes to the game rules. One is to require that the werewolves vote unanimously in order to eat somebody. Coupled with hiding each werewolf's vote from the other werewolves, this can force the werewolves to have at least some discussion at night. It is not clear if similar rule changes can help the werewolves when one player knows their identities (because of a vulnerability or decisive tactic). One strategy werewolves have employed is to eat the player they know will find out their identities in the first night, but the witch usually saves the player with the potion (the witch is a special townsperson whose abilities include saving a townsperson once with a potion, refer to Ensafi *et al.* [9] for more information).

One issue that we have not addressed is out-of-band communication amongst players. Players can use ytalk, shared files, covert channels, and other aspects of the game server to communicate in secret outside the game, as well as verbal communications, cell phone text messages, and other forms of communication that do not even involve the server. Any mitigation strategy that attempted to silence in some way the student who knows the identities of the werewolves would have to address these out-of-band communications.

Another potential mitigation strategy is that the student(s) who is/are dominating the game could be offered an incentive to collude with the moderator. The goal of this collusion could be, *e.g.*, to ensure that a particular player is the last and only player left in the game at the end. This would give the dominating player an incentive to help both the townspeople and the werewolves.

4.2 Discovery

To reduce the amount of time between when one student or group discovers a vulnerability in the server or a decisive tactic and when the issue is addressed fully, the moderator needs mechanisms for discovering and understanding exploits and decisive tactic implementations.

One mechanism for this might be to make disclosure of vulnerabilities and decisive tactics before their use in a game carry a larger reward than winning games. Then the issue can be addressed before it is used in a game, the student who discovered it could be allowed to use it in one or two games to illustrate it, and then they could inform the other students about it and how to counter it.

Another mechanism is that the moderator, who also has root access on the system, can spy on all players. This can include copying their home directory and analyzing any scripts found, reading their shell histories, or attaching to their screen sessions, for example. We have employed this to some extent, but have not found it to be as effective as it could be because there is a lot of information to sort through. Automated tools to aid in this process would be helpful. Note that we always tell students that they have no expectation of privacy on the Werewolves server, not even for passwords.

5 Conclusion

In conclusion, we have described the various cases so far where a vulnerability in the Werewolves server or a decisive tactic strategy caused a student or group of students to dominate the game. Werewolves has the potential to be a powerful and fun tool for teaching computer and information security and privacy, but in order for the game to remain fun and educational it must remain competitive. Thus we proposed various mitigation strategies that we plan to implement.

In the broader context, for any security-related game where students are pitted directly against each other in a strategic and tactical way, maintaining competitiveness so that all students can learn and have fun will be a challenge. Capture-the-flag contests are often based on points and a fixed set of "puzzles." This keeps students of all skill levels engaged, but there are certain advantages to pitting students directly against each other. One key advantage is that games like Werewolves are open-ended, meaning that there are no limits on what students can explore and teach themselves in order to gain an advantage in the game. Fundamentally, for games like Werewolves the game itself and the players must evolve for the game to remain competitive, and the trick will be how to encourage this evolution and make sure it can happen in a timely manner.

We draw inspiration from the success of Core War, where continual evolution of the game and ways to engage players at all skill levels has led to a successful game with a vibrant community. It is our hope that by further developing the Werewolves game we can also allow for evolution of the game and a community of players.

Acknowledgments

We would like to thank all of the tasty people who shared vulnerabilities and other insights about Werewolves with us, all of the savory students who have played Werewolves, all of our appetizing colleagues at CSET 2012 and other venues that have given us feedback, Jörg Schneider and all the scrumptious members of the ENOFLAG team, and the delectable anonymous reviewers for 3GSE. John Montoya and Tim C'de Baca helped us to understand the logic errors in the Werewolves server and are helping to develop version 1.3. The UNM Department of Computer Science supported Mike Jacobi with a teaching assistantship during the development of Werewolves. This material is based upon work supported by the National Science Foundation under Grant Nos. #0844880, #0905177, #1017602, and #1314297.

References

[1] ASKAROV, A., ZHANG, D., AND MYERS, A. C. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM conference on Computer and communications security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 297–307.

[2] BERNSTEIN, D. J. Cache-timing attacks on AES. `http://cr.yp.to`, 2005.

[3] BROWNE, R. The Turing test and non-information flow. In *IEEE Symposium on Security and Privacy* (1991), pp. 373–388.

[4] BROWNE, R. An entropy conservation law for testing the completeness of covert channel analysis. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and Communications Security* (New York, NY, USA, 1994), ACM Press, pp. 270–281.

[5] BROWNE, R. Mode security: An infrastructure for covert channel suppression. In *IEEE Symposium on Security and Privacy* (1999), pp. 39–55.

[6] BRUMLEY, D., AND BONEH, D. Remote timing attacks are practical. *Comput. Netw. 48*, 5 (Aug. 2005), 701–716.

[7] CLARKSON, M. R., MYERS, A. C., AND SCHNEIDER, F. B. Belief in information flow. In *CSFW '05: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 31–45.

[8] DEWDNEY, A. Computer Recreations. Scientific American, 250, 5, 14-22, May, 1984 (and generally 1984-86).

[9] ENSAFI, R., JACOBI, M., AND CRANDALL, J. R. Students who don't understand information flow should be eaten: An experience paper. In *CSET 12: Proceedings of the 5th USENIX Workshop on Cybersecurity Experimentation and Test* (2012).

[10] ENSAFI, R., PARK, J. C., KAPUR, D., AND CRANDALL, J. R. Idle port scanning and non-interference analysis of network protocol stacks using model checking. In *Proceedings of the 19th USENIX conference on Security* (Berkeley, CA, USA, 2010), USENIX Security'10, USENIX Association, pp. 17–17.

[11] GOGUEN, J. A., AND MESEGUER, J. Unwinding and inference control. In *IEEE Symposium on Security and Privacy* (1984), pp. 75–86.

[12] JANA, S., AND SHMATIKOV, V. Memento: Learning secrets from process footprints. In *Proceedings of the 33nd IEEE Symposium on Security & Privacy* (San Francisco, CA, May 2012).

[13] KANG, M. H., AND MOSKOWITZ, I. S. A pump for rapid, reliable, secure communication. In *CCS '93: Proceedings of the 1st ACM conference on Computer and Communications Security* (New York, NY, USA, 1993), ACM Press, pp. 119–129.

[14] MCILROY, M. D., MORRIS, R., AND VYSSOTSKY, V. A. Darwin, a game of survival of the fittest among programs. Available at `http://www.cs.dartmouth.edu/~doug/darwin.pdf`.

[15] MILLEN, J. K. 20 years of covert channel modeling and analysis. In *IEEE Symposium on Security and Privacy* (1999), pp. 113–114.

[16] PERCIVAL, C. Cache missing for fun and profit, 2005. `http://www.daemonology.net/hyperthreading-considered-harmful/`.

[17] PROCTOR, N. E., AND NEUMANN, P. G. Architectural implications of covert channels. In *Fifteenth National Computer Security Conference* (October 1992), pp. 28–43.

[18] QIAN, Z., AND MAO, Z. M. Off-path TCP sequence number inference attack – how firewall middleboxes reduce security. In *Proceedings of the 33nd IEEE Symposium on Security & Privacy* (San Francisco, CA, May 2012).

[19] SHAH, G., MOLINA, A., AND BLAZE, M. Keyboards and covert channels. In *USENIX Security Symposium 2006* (2006).

[20] SONG, D. X., WAGNER, D., AND TIAN, X. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security Symposium 2001* (2001).

[21] SUTHERLAND, D. A model of information. In *Proceedings of the 9th National Computer Security Conference* (1986).

[22] YUMEREFENDI, A., MICKLE, B., AND COX, L. P. TightLip: Keeping applications from spilling the beans. In *Networked Systems Design and Implementation (NSDI)* (2007).

[23] ZALEWSKI, M. *Silence on the Wire*. No Starch Press, Inc., San Francisco, CA, 2005.

[24] ZHANG, K., AND WANG, X. Peeping Tom in the neighborhood: keystroke eavesdropping on multi-user systems. In *Proceedings of the 18th conference on USENIX security symposium* (Berkeley, CA, USA, 2009), SSYM'09, USENIX Association, pp. 17–32.

Notes

[1] CS 444/544 Introduction to Cybersecurity Spring 2013 at the University of New Mexico (UNM), CS 491/591 Computer Security and Privacy at UNM, a short-lived "Werewolves Club" at UNM, and the capture-the-flag team ENOFLAG in Germany.

[2] At this point in the paper authors Jed Crandall and Roya Ensafi would like to note that Mike Jacobi deserves all of the credit for developing versions 1.0 and 1.1 of the Werewolves source code.