Students Who Don't Understand Information Flow Should be Eaten: An Experience Paper

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

Abstract

Information flow is still relevant, from browser privacy to side-channel attacks on cryptography. However, many of the seminal ideas come from an era when multi-level secure systems were the main subject of study. Students have a hard time relating the material to today's familiar commodity systems.

We describe our experiences developing and utilizing an online version of the game *Werewolves of Miller's Hollow* (a variant of *Mafia*). To avoid being eaten, students must exploit inference channels on a Linux system to discover "werewolves" among a population of "townspeople." Because the werewolves must secretly discuss and vote about who they want to eat at night, they are forced to have some amount of keystroke and network activity in their remote shells at this time. In each instance of the game the werewolves are chosen at random from among the townspeople, creating an interesting dynamic where students must think about information flow from both perspectives and keep adapting their techniques and strategies throughout the semester.

This game has engendered a great deal of enthusiasm among our students, and we have witnessed many interesting attacks that we did not anticipate. We plan to release the game under an open source software license.

1 Introduction

Computer security is becoming an instrumental field of study in any computer science curriculum. A computer security course can reinforce other fundamental computer science topics in everything from systems curricula (*e.g.*, operating systems, architecture, or networks) to theory and data structures [24]. Although traditional lecture-based teaching can be valuable, with respect to computer security active learning using hands-on, practical labs has been suggested by many instructors and researchers [18, 10, 5].

Games, by their very nature, can provide an active learning environment. As is shown by Oblinger [21], there are many attributes of games which make them pedagogically effective learning environments.

In this paper we discuss our experience in designing, programming, and playing an online version of *The Werewolves of Miller's Hollow* (a variant of *Mafia*), which we hereafter refer to simply as "Werewolves." 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.

We selected this game for its complex social interactions and the fact that it necessitates privacy and confidentiality for the werewolves. We decided to implement the game within a UNIX-based operating system, so that there are countless opportunities for the townspeople to exploit *covert inference channels.* In our case, the students must make use of these inference channels to identify the werewolves because there is no fortune teller roll (known as the detective in *Mafia*) who can investigate the identities of other players directly.

A covert channel attack is a type of side-channel attack that exploits communication channels that were not intended for communication by the system's designers, and violate an explicit or implied security policy [25]. Based on our past experiences of teaching this topic from a lecture-based approach, students have struggled with not only understanding information flow through covert channels, but also with the importance of exhaustively enumerating channels in a systematic way.

The benefits of lab-based teaching of computer security through games are well-known and this technique is applied regularly. For example, Fanelli *et al.* [10] discuss their successful experience in using practical approaches in teaching an information assurance course. Also, Du [8] has developed a set of hands-on labs for teaching a variety of computer security topics which is freely available. Our main contribution is in developing a game that conveys information flow in a fun way and challenges students to continually evolve their techniques and strategies.

It is important to mention that, based on the constantly evolving nature of computer security, a substantial amount of effort is required in creating and maintaining labs that are pertinent to the real world. We believe that such an effort is worthwhile in improving a student's experience in computer security courses. All information and code for our game will be released under an open source license. We hope that the game can continue to evolve in a constant battle between townspeople and werewolves where no technique or strategy exists which precludes a response from the other side. As an example of such a game, Core Wars [7] continues to challenge beginners and experts alike to take their knowledge and skill one step further. We discuss the challenges we have identified for maintaining this property for our Werewolves game and our plans for addressing them.

This paper is organized as follows. First, we give the basic rules of Werewolves in Section 2. This is followed by Section 3 which gives a brief background of information flow research. We developed ways to prepare students for the game, which we detail in Section 4. We explain our implementation of Werewolves and our server setup in Section 5. In this section, we also offer some examples of information leaks and describe some of the attacks our students employed. The lessons we learned from our experiences so far this semester are enumerated in Section 6. This is followed by discussion and future work in Section 7.

2 The rules of Werewolves

The game Werewolves (also known as Mafia) is a popular game that is not only studied theoretically [6, 20], but also is used for researching other topics such as video conferencing [2]. There are several variants of this game which differ only in minor details. In the Werewolves version as we have implemented it, players are divided into three groups: townspeople, werewolves, and the witch. Typically, there is also a fortune teller who can view the identity of one other player every night. However, we omitted this role because we want to encourage the townspeople to exploit covert inference channels rather than rely on the fortune teller. Initially, players only know their own identity. The werewolves know of each other and can infer that all other players are therefore townspeople or the witch, but the werewolves do not know which townsperson is the witch. At night, werewolves open their eyes and decide, through discussion and voting, whom to eat from among the townspeople. At the end of their discussion, which should be hidden from the townspeople, they eat the chosen player and then close their eyes. The witch then opens his/her eyes, and can then select a player to kill with a poison, save the player who the werewolves had eaten with a potion, or decide to not perform either of these actions (pass). The poison and the potion can each only be used once by the witch in the game. When the next day starts, all still-living players open their eyes and debate amongst themselves which players that are still in the game could be werewolves, and potentially kill one of the accused players via hanging. The days and nights pass until either all the townspeople or all the werewolves are dead. The group with players still remaining alive in the end wins.

We explain in further sections how this game has become a great pedagogical tool for teaching the concepts of information flow and covert channels.

3 Background on information flow and covert channels

The term "covert channel" was coined by Lampson [15], who posed the confinement problem. The confinement problem is: can we design and implement a service that takes inputs from a customer but provably does not save the inputs or transmit them to the service's owner against the customer's wishes. A related idea is Fenton's memoryless subsystems [11]. Lipner [16] commented that covert channels would be extremely difficult and costly to close.

Covert channels are channels of potential information flow in a system that were not intended by the system designers for communication, but can be used to communicate information, especially in a way that violates an explicit or implied security policy. Covert channels can be divided into two types: storage and timing. Storage channels are based on the storage of information, such as a shared semaphore. Timing channels are channels where the information is represented as timing, such as the position of the head with respect to certain tracks on a hard disk drive. Note that these examples are a bit ambiguous in that exploitation of either implies both storage and timing elements (the state of the lock is inferred by processes via the *timing* of when they are able to obtain it, and the position of the hard disk drive head implies information and therefore is a form of *storage* itself).

Goguen and Meseguer [12] describe non-interference, which is a property that says that high-security inputs should never affect low-security outputs. Kemmerer [14] describes a shared resource matrix methodology for enumerating storage and timing channels based on a transitive closure operation on a matrix that describes information storage objects by their resource attributes and primitives. Wray [26] gives some clarification on the nature of timing channels, and describes a method for enumerating channels with timing aspects by considering pairs of "clocks." "Clocks" are an abstract notion and correspond with any observable event in the system. By enumerating all "clocks" and considering all pairs of "clocks" it is possible to enumerate all covert channels with timing aspects.

To get students started so that they could systematically reason about how to play the Werewolves game and relate it to existing research and practice in information flow, we assigned and discussed three papers: Kemmerer [14], Wray [26], and a paper that applies non-interference to an intuitive and simple problem in network security [9]. We also gave students a link to Goguen and Meseguer [12], the paper that defined the notion of non-interference, but since our students were a mix of undergraduates and graduates from a variety of major areas of study, we did not make Goguen and Meseguer required reading. These readings were in addition to the material in Chapters 8, 16, and 17 of the course textbook, Bishop's *Computer Security: Art and Practice* [4].

Students typically have little trouble understanding the simple version of non-interference and how to apply it to a practical problem such as keystroke inference among UNIX processes. Kemmerer's and Wray's methodologies are also straightforward to understand, but students struggle to some degree when trying to apply these latter two methodologies to practical problems. This struggle is very fruitful and helps them to better understand Kemmerer's and Wray's methodologies in terms of their power and limitations.

One issue that often comes up when applying these methodologies to practical problems in the classroom is that most of the early research on information flow and covert channels was driven by the desire to build multilevel secure systems. Thus, this early research either explicitly or implicitly makes the assumption of collusion, that is, it is assumed that the sender and receiver of the information are both intending for the illicit flow of information to occur. In inference channels, the sender of the information is not always intending to send the information. For example, in Percival's attack on SSL RSA encryption via cache timings [22] the RSA encryption software does not intend to modulate its memory usage timings based on the secret RSA key. Instead, this happens by accident as a property of performance optimization and the Chinese remainder theorem. By asking students to apply classical research on covert channels to the more contemporary problem of inference channels, they gain a better understanding of both the classical research and what it is that distinguishes inference channels from other covert channels that are based on collusion.

Although we did not do so in our class, for a graduate class it might be appropriate to also discuss recent papers that apply ideas about timing and inference channels from the perspective of contemporary research on information flow, such as Ensafi *et al.* [9], Askarov *et al.* [1], Qian and Mao [23], or Jana and Shmatikov [13].

4 Lab preparation

At the beginning of the semester, we spent three weeks on ethical training. The goal of the training was to make students familiar with the current cyber-related laws and policies in practice and give them some notion of the ethics of security such as ethical disclosure. Students were assigned to imagine themselves in three different environments (*e.g.,* on campus, at a coffee shop, at home) performing actions that most people would consider benign, but that might violate laws or polices. Then, students exchanged their reports and acted as lawyers hired by a client to determine the legality of these actions. For the ethics part, we discussed the impact of attacks and ethical disclosure issues.

A security based game helps students to grasp the security concepts intuitively. Werewolves is an interesting party game, but it can be difficult to understand if you have not played it before. Therefore, in one class session we had students play the actual game with cards, to become familiar with the rules.

As explained in Section 3, we assigned and discussed three papers: Kemmerer [14], Wray [26], and a simpler paper about non-interference [9] that is easier for students to understand than Goguen and Meseguer [12].

After preparing students for the game, we divided them into groups of three and gave them a username and password to be able to log in remotely to the game server. All of the game code was made available to students on the course website, which also included a sample client script (more details in Section 5).

4.1 Werewolves on Mondays

The introductory Cybersecurity course that we developed this game for is a three-credit course, which met three times a week for an hour each session. We dedicated a month to this lab, where every Monday students were required to meet with their groups someplace away from the classroom to remotely log into the game server.

As the instructors, we initiated a game session for students to connect to during class time and play the game. On Wednesdays and Fridays, we met in class to discuss techniques and help groups to improve their inference methods.

4.2 Proposed improvements for preparation

During game play, we noticed that poor strategies that were not related to information flow caused the werewolves to be detectable even without inference attacks. This detracted from the desired aspect of the game that students should learn about information flow. For example, werewolves would often vote against their accusers during the day as townspeople, rather than killing them anonymously at night, making it clear who the werewolves were based solely on their voting patterns. Note that killing an accuser always draws suspicion to a werewolf, but voting openly for an accuser is an even worse strategy.

When students submitted their writeups about this lab at the end of the semester, most groups tended to only scratch the surface of overarching concepts such as Kemmerer's shared resource matrix methodology, noninterference, or game theory. For example, some groups found inference channels using informal methods and then described how such a channel would look in a Kemmerer-style matrix, rather than using the Kemmerer-style matrix to enumerate possible inference channels.

To improve preparation for this lab in the future, we recommend the following:

- More time should be dedicated to class discussions about strategies for basic gameplay.
- Students should be given very specific instructions for applying overarching concepts. For example, students could be given a subset of Linux system calls and told to create the full Kemmerer-style matrix for any resource attributes referred to in those system calls (typically these would be referenced by pointers into userspace to data structures that the kernel should read or modify).

5 Game design

The game architecture resembles a client-server architecture where the client and server are on the same host and communicate through named pipes. Gameplay is hosted on a single machine that players remotely log into via SSH (Figure 1). Initially, the Werewolves server was a small laptop with an Intel Atom processor and 1GB of RAM. Later the server was moved to a dual quad-core Intel Xeon with 16GB of RAM. The laptop had been specifically selected to increase the probability of discovering side-channel attacks because of its limited resources, but this turned out to not be necessary.

Players were assigned a username and password that is linked to a user account which is set up for Werewolves. The server, written in Python, is an automated moderator that manages the game. All communication between the server and clients is handled through Linux named pipes. Security is enabled through UNIX permissions and server-side controls. Players were required to write their own clients to obscure their communication, though they were given a default client to start from. All communication is recorded to provide real-time updates and detailed post-game analysis.

Each user owns two pipes, a `read` and a `write` pipe, and the moderator is in the group associated with these pipes. The `write` pipe is used for user-to-server communication and has permissions `-wxr-x---`. The `read` pipe facilitates server-to-user communication and has permissions `r-x-wx---`. Further, each pipe is stored in its own directory with the same permissions as its pipe, to prevent `stat()`s from other users. All individual pipe directories are stored in a `pipes` directory owned by the moderator with `rwx-----x` permissions. The strict file architecture and permissions ensure that each pipe is being used only by the appropriate user, and that information does not flow through the nodes for the pipes since this would make it too easy to detect the werewolves.

There is a communications Python script that provides generic helper functions to the clients and the server. The most basic of these functions are `send` and `receive`. These functions simply provide parsing and error handling for `cat`s and `echo`s on pipes (read and write, respectively), and are important because other functions are built on top of them. This implementation is significant because it means all communication is handled via system calls (which are voluntary context switches) and influences the types of information inference techniques. Other significant functions include `broadcast`, `groupChat`, and `poll`. The `broadcast` function sends the same message to every user, and is used for public messages. The `groupChat` function allows a group of users to all communicate with one another. This enables, for example, werewolves to communicate amongst each other at night, or the townspeople to communicate during the day. Finally, `poll` is used to handle the voting aspects of the game, *i.e.*, whom the werewolves vote to eat, or how the witch chooses to play.

The main role of the server, which is owned by the moderator user, is to initialize a game and cycle between rounds of day and night until an end state is

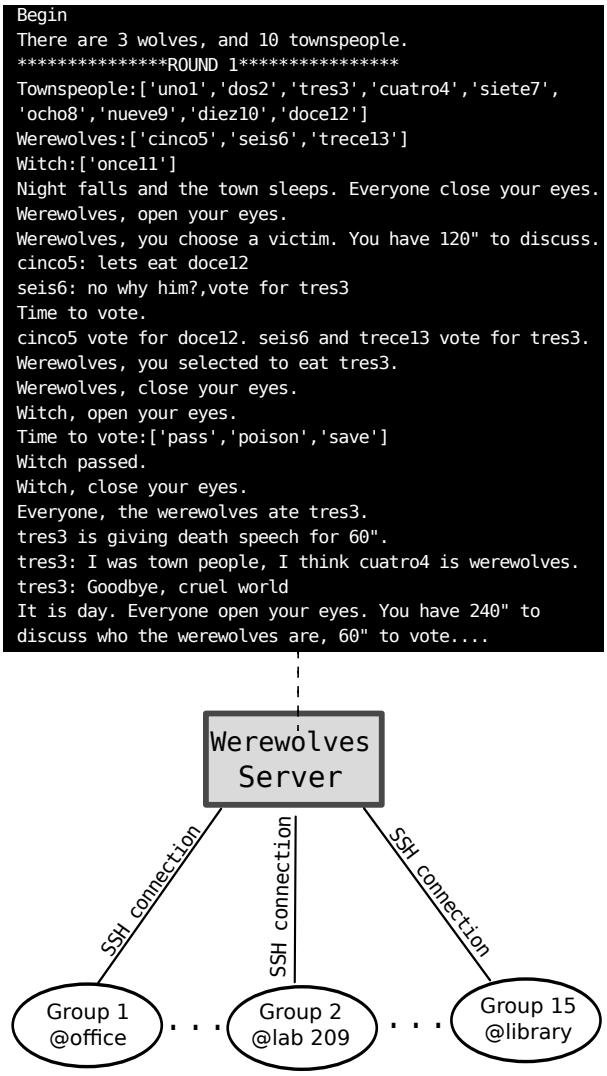

Figure 1: **Basic topology of our game.**

reached. Prior to the game starting, the moderator selects how long each chat and vote session lasts. Initializing the game consists of starting a moderator `listener` thread, starting a `groupChat` thread, listening for client connections, and randomly assigning roles to the connected players. The game starts at night, in which the werewolves group chat for `wolfchat` seconds, and then a vote session is spawned amongst them. Then, a vote session is spawned for the witch, with a dynamic list of options based upon the werewolves action. Next, the game enters day, and the townspeople group chat for `townchat` seconds. The day concludes with a town vote to accuse one player of being a werewolf and then hang them, and then the next night begins. If a player is eliminated, either through a werewolf vote, a witch poison, or a town vote, the server removes that player from the appropriate list(s) and allows them to make concluding remarks. Throughout the night and day, the server checks for end game conditions before continuing. Though the server is fully automated, the moderator has access to various commands to intervene with an active game. For example, the moderator may `broadcast` to the group to spur discussion, remove a player suspected of unauthorized play, or skip over sessions that are lasting too long.

The default client is written in Python and has two threads. The `send` thread sits in an infinite loop waiting for keyboard input to send to the server. The `listener` thread infinitely loops to receive server input and print it to the screen. This client acts in predictable ways, and may be modified to obscure its actions. Further, the only dependencies to communicate with the server are the designated pipes, so a client may be written in any language, or emulated entirely from the command line.

5.1 Examples of information leaks

In this section, we describe several attacks of the type that we expect the townspeople to employ against the werewolves, to illustrate the kinds of information flows and inferences that the game is about. All commands and stated properties of Linux systems were tested in Ubuntu 10.04. All of these inference attacks are designed for inferring information about processes that are not owned by the attacker, where neither the process nor the attacker are running with administrator privileges.

To maintain a balance between townspeople and werewolves in the game, we want for it to be possible for the townspeople to infer who the werewolves are, but also possible for the werewolves to evade detection. For this reason, we have made attempts to close information flows that make it too easy for the townspeople to detect the werewolves. For example, it is possible to tell whenever a named pipe has been written to by performing a `stat()` on the named pipe, even without ownership or any permissions on the pipe:

```
while : ; do stat /tmp/mypipe | grep Change; \
sleep 1; done
```

Since `stat()` gives both the time a file or pipe was last accessed when it was last modified, this makes it trivial for the townspeople to detect the werewolves directly through their interactions with the server via the pipe. Note that Linux systems allow a user to `stat()` a file or pipe even if they do not own it and have no read, write, or execute permissions. However, to `stat()` a file a user needs execute permissions for all `dentry`'s in the file path. Thus, our solution to this flow of information was to place all named pipes in subdirectories that only the moderator and the user who is using the pipe to communicate with the moderator have execute permissions for.

To get students started and give them an example of how to spy on another process, we gave them two examples. The first can only detect keystroke activity through the physical keyboard attached to the computer, it does not work for SSH logins over the network. This attack is described in more detail by Zalewski [27].

```
cat -v /dev/random
```

Because every keystroke on the keyboard adds entropy to the entropy pool, after the entropy pool is drained keystrokes cause blocking reads to the entropy pool to be fulfilled because of the entropy each keystroke adds. The special character device `/dev/random` blocks when the entropy pool is empty, in contrast to `/dev/urandom` which continues to return the results of a cryptographic hash even after the entropy pool runs out of entropy.

To show students the potential for information flow inference in the `/proc` filesystem, we gave them an example that was based off of scheduler information, specifically voluntary context switches. The stack pointer of other processes that was used in Zhang and Wang [28] is no longer visible. However, for a process that is idle except for reading keystrokes, every keystroke is exactly one voluntary context switch. So, it is possible to count the keystrokes of a process (in this example, the process with PID 1234 that could be, *e.g.*, an interactive command `bash` shell) using the following from another `bash` shell:

```
while : ; do grep "nr\_voluntary\_switches" \
/proc/1234/sched; sleep 1; done
```

Every time a key is pressed this variable is incremented. If the process is using a file descriptor in non-canonical mode, then each write to or read from the file descriptor of a line (*e.g.*, through a named pipe) will also appear as a voluntary context switch.

From there, students were encouraged to find their own information flow inference attacks. One example of a more refined variant of the attack on voluntary context switches is to only consider transitions to/from a wait state:

```
while : ; do grep "se\.wait\_count" \
/proc/1234/sched; sleep 1; done
```

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
```

With `root` privileges it is trivial to connect a network socket to a specific process and user via `fuser`, `ss`, or `lsof`. Doing so directly is not permitted in Linux without `root` privileges, but the IP address of other users logged into the system can be obtained via `who --ips`. Also, correlations between network traffic and process activity can help to tie sockets to processes.

It is also possible to get more detailed information about the state of a process, including, *e.g.*, if the file object it is waiting for is a pipe:

```
ps -eo user,pid,args,wchan | grep pipe
```

This ability to get detailed information about process state created a very powerful information flow inference attack for the townspeople to exploit in the first version of our server. Specifically, the werewolves were forced into a very conspicuous interaction with the server that took them through process state transitions that the townspeople did not need to go through. For example, the townspeople do not transition out of a `pipe_wait` state when the server makes an announcement such as "Werewolves, cast your votes..." However, at that point in each round the werewolves are forced to be in a `pipe_wait` state and transition out of it. Part-way through the semester we fixed the server by synchronizing the townspeoples' and werewolves' transitions so that the server always printed a message to all players, *e.g.*, "Werewolves, cast your votes..." to the werewolves and "The werewolves are now casting their votes..." to the townspeople.

In terms of strategies for the werewolves to hide their activities, at least one werewolf must make at least one write to their named pipe during the night if they are to kill any townsperson. This introduces many interesting strategies for how the werewolves vote and kill, but another interesting strategy is to hide which process they are using to interact with the game server, *e.g.*, by continually forking child processes that inherit the file descriptors and state of their parent and then the parent exits, so that the PID keeps changing.

We expect the game to continually evolve over time both in terms of general strategies and in terms of specific information flow inference attacks. As long as the probability of any given townsperson detecting any given werewolf in a round is neither too close to 0% nor to 100%, we hope that the game will remain interesting and students will continue to increase the sophistication of their attacks and defenses. To maintain this, however, we anticipate also needing to adapt the game server over time.

5.2 Advanced attacks and strategies we did not anticipate

One important lesson we learned is that if the game server reveals the identities of the werewolves in a way that the werewolves have no way of ameliorating, then the game quickly devolves to the point where it is impossible for the werewolves to win. Our server had a bug where it forked separate `cat` binaries as children to write into named pipes of the werewolves. Since command line arguments are publicly visible and the named pipe was specific to one group, the townspeople could easily find the identities of the werewolves by monitoring the child processes the server forked, no matter what the werewolves did. By the end of the semester, at least two groups had found this and were easily identifying the werewolves in the first round of every game. We plan to fix this bug in our server before releasing the server and its source code.

One method that appeared to be effective for the werewolves to hide their identity was for one werewolf to vote right at the end of a voting round and the others to remain silent. In other words, if the werewolves were given 60 seconds to vote and there were three werewolves, two would remain silent and one would count down 60 seconds and try to register their vote right before the server transitioned to the witch phase. This strategy was effective in one game where one of two groups who could identify the werewolves from server actions was a werewolf and the other was killed in the first night by the werewolves. Another method that one group attempted was to make many system calls all the time, whether or not they were a werewolf. This resulted in this group being accused of being a werewolf and subsequently being hanged every game.

We also found it interesting that several groups also tried social engineering or more conventional attacks on the game and on other users. Our instructions had stated that only denial-of-service attacks were prohibited, all other types of attacks were allowed. In fact, students were told that for this lab, "If you're not cheating, you're not trying." One group obtained the `/etc/shadow` file of the server containing the password hashes of the moderator and all other groups, and made an unsuccessful attempt to find weak passwords with a dictionary attack[1]. Another group discovered a vulnerability in our game server where they could change their identity to any other user during the chat phase, then made it appear as though a werewolf were confessing to being a werewolf. At least two of the thirteen groups attempted to exploit a local privilege escalation vulnerability in Ubuntu 10.04 that we had already patched.

6 Lessons learned and discussion

We remained in constant contact with the students throughout the semester to understand the early development of the game. We also read writeups that the students submitted at the end of the semester detailing their techniques and strategies for the game and the attacks they developed. The lessons we learned include:

1. **It is necessary for the server to be designed carefully so that the werewolves have at least a fighting chance.** After playing several games, we learned that the design of our server made it too easy for the townspeople to identify the werewolves. This was explained in Section 5.1. Basically, twice we were made aware by students of bugs in the game server where it wrote to the werewolves' pipes in a way that created a blatant leak of information that the werewolves had no control over. We anticipate that the game server will require careful design and maybe some updates to encourage the evolution of the game over time.

2. **Students exceeded our expectations in terms of the sophistication of their attacks.** Students discovered the aforementioned information flow vulnerability in the server, and developed many other attacks that exceeded our expectations. For example, a vulnerability in our server was discovered by students that enabled them to post messages appearing to be from other users and employ social engineering attacks (this was described in Section 5.2).

3. **All students were challenged, but still able to contribute.** Because of the continuous evolution of the game and the open-ended nature of possible techniques and strategies, students with strong backgrounds in UNIX system hacking were challenged and students who were learning UNIX system hacking were still able to contribute. In other words, the nature of the game put our students on relatively equal footing and challenged them to learn.

4. **Students' enthusiasm for the game led to many interesting discussions about information flow and game theory.** We found that our students were extremely enthusiastic about the game, and were so excited to talk about different inference channel attack techniques and strategies that they shared detailed information even with students from other groups, despite the ostensibly competitive nature of the game. After games, students would often come from the scattered locations in which they had met with their groups during the game and congregate

outside our offices for discussion. This level of enthusiasm is contrary to our experience from past semesters of teaching information flow in a lecture-style format.

5. **Students needed to apply techniques to deal with the noise in the system.** Soon after understanding the game and our game implementation, students learned to use context switches as a way of detecting werewolves. However, any efforts to count the context switches of other processes in turn generated a lot of context switches. Students had to rethink their techniques and strategies, and in particular they needed to think more deeply about how to enumerate possible covert inference channels as exhaustively as possible to overcome this signal-to-noise-ratio issue.

6. **Operating-system-level covert inference channels are numerous and serious enough that lower-level inference channels are not necessary.** Covert inference channels that are at a higher layer of abstraction, such as the operating system (see examples in Section 5.1), are much easier to understand and to exploit than low-level attacks such as the timing of cache misses (*e.g.*, attacks such as Percival's [22] or Bernstein's [3]). When we first designed the game, we were not sure if the operating system would provide enough information flow leaks to allow students to be successful with simple attacks implemented as shell commands or in scripting languages. We had been anticipating that we would need to teach some more advanced hardware-level attacks that were very delicate and difficult to implement. We even initially used a resource-constrained server to make the implementation of hardware-level attacks more viable. After we realized that many very simple attacks were possible, even from just shell commands, we moved the server to more high-performance hardware.

7. **Continuously engaging the students between games turned out to be more valuable than we had imagined.** Because we had individual and whole-class discussions during the week between the games on Mondays, we were able to create a tight feedback loop where the students understanding and sophistication of attacks and our understanding of how the game was evolving were tightly coupled. We learned a great amount from the students and they learned a great amount from each other.

6.1 Evolution of the game over time

We expect the game to continually evolve over time both in terms of general strategies and in terms of specific information flow inference attacks. As long as the probability of any given townsperson detecting any given werewolf in a round is neither too close to 0% nor to 100%, we hope that the game will remain interesting and students will continue to increase the sophistication of their attacks and defenses. To maintain this, however, we anticipate also needing to continually adapt the game server.

Our strategy for this anticipated evolution of the game is to have "knobs to turn" that change the dynamics of the game in favor of either the werewolves or the townspeople. However, we first need to do an inference channel analysis of our server to make sure that it is not possible to infer the identities of the werewolves directly from the server in a way that the werewolves cannot ameliorate.

A very powerful knob that can shift the game toward either direction is to vary the number of werewolves in the game. The werewolves always have perfect information about who is a werewolf and who is a townsperson. At one extreme, if the townspeople have zero information about who the werewolves are then they can only hang players at random and the optimal number of werewolves to make the game fair is $\sqrt{R}$ where R is the total number of players. At the other extreme, where the townspeople have full information about who the werewolves are, the game devolves into a deterministic sequence of killings, but as we approach this extreme the optimal number of werewolves is approximately $\frac{R}{2}$. In between these extremes, with partial information, the optimal number of werewolves is $\Omega(R)$, *i.e.*, it grows linearly with respect to the number of players [6].

Another possible knob, that can shift the game in favor of the townspeople, is to enforce rules on the werewolves' voting. For example, a rule that stated that the werewolves could only kill someone if all living werewolves voted unanimously would force all werewolves to vote and would encourage discussion amongst the werewolves.

The progress of the game could become an issue, particularly if the werewolves decided that total silence was their best chance of survival. Progress could be enforced by limiting the number of rounds the werewolves have in order to win, for example. Denial-of-service was prohibited by us this semester, but could create an interesting aspect of the game that might be a good learning aid for availability issues and robustness techniques. In fact, many aspects of information security could be explored within the context of the online Werewolves game we have developed that pervade the McCumber INFOSEC Model [19, 17].

7 Concluding remarks and future work

Mafia, the game that Werewolves is a variant of, was invented in Russia over 25 years ago and has inspired a vibrant community and a large amount of research and learning in everything from psychology to game theory. Our hope is that our online variant of Werewolves will inspire a similar community within the academic computer security community. We envision that the game, and the community's understanding of information flow and covert inference channels, can continue to develop indefinitely. This paper has described our experiences in pursuit of this goal.

Acknowledgments

We would like to thank Nick Aase, the CSET anonymous reviewers, and our shepherd, Sean Peisert, for valuable feedback. We would also like to thank the students in our Spring 2012 CS 444/544 "Introduction to Cybersecurity" class for their input and patience. Finally, we would like to thank the UNM Department of Computer Science for supporting Mike Jacobi with a teaching assistantship. This material is based upon work supported by the National Science Foundation under Grant Nos. #0844880, #0905177, and #1017602.

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] BATCHELLER, A. L., HILLIGOSS, B., NAM, K., RADER, E., REY-BABARRO, M., AND ZHOU, X. Testing the technology: playing games with video conferencing. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2007), CHI '07, ACM, pp. 849–852.

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

[4] BISHOP, M. *Computer Security: Art and Science*. Addison-Wesley, 2003.

[5] BRATUS, S. What hackers learn that the rest of us don't: Notes on hacker curriculum. *IEEE Security and Privacy 5* (2007), 72–75.

[6] BRAVERMAN, M., ETESAMI, O., AND MOSSEL, E. Mafia: A theoretical study of players and coalitions in a partial information environment.

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

[8] DU, W., TENG, Z., AND WANG, R. SEED: a suite of instructional laboratories for computer SEcurity EDucation. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education* (New York, NY, USA, 2007), SIGCSE '07, ACM, pp. 486–490.

[9] 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.

[10] FANELLI, R. L., AND O, CONNOR, T. J. Experiences with practice-focused undergraduate security education. In *Proceedings of the 3rd international conference on Cyber security experimentation and test* (Berkeley, CA, USA, 2010), CSET'10, USENIX Association, pp. 1–8.

[11] FENTON, J. S. Memoryless subsystems. *The Computer Journal 17*, 2 (1974), 143–147.

[12] GOGUEN, J. A., AND MESEGUER, J. Security policies and security models. In *IEEE Symposium on Security and Privacy* (1982), pp. 11–20.

[13] 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).

[14] KEMMERER, R. A. Shared resource matrix methodology: an approach to identifying storage and timing channels. *ACM Trans. Comput. Syst. 1*, 3 (1983), 256–277.

[15] LAMPSON, B. W. A note on the confinement problem. *Communications of the ACM 16*, 10 (1973), 613–615.

[16] LIPNER, S. B. A comment on the confinement problem. In *SOSP '75: Proceedings of the fifth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1975), ACM Press, pp. 192–196.

[17] MACONACHY, W. V., SCHOU, C. D., RAGSDALE, D., AND WELCH, D. A model for information assurance: An integrated approach. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security* (2001).

[18] MATETI, P. A laboratory-based course on internet security. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (New York, NY, USA, 2003), SIGCSE '03, ACM, pp. 252–256.

[19] MCCUMBER, J. R. Information systems security: A comprehehnsive model. In *Proceedings of the 14th National Computer Security Conference* (1991), National Institutde of Standards and Technology.

[20] MIGDA, P. A mathematical model of the Mafia game. *Current* (2010), 12.

[21] OBLINGER, D. The next generation of educational engagement. *Journal of Interactive Media in Education 2004*, 1 (2004).

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

[23] 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).

[24] VAUGHN, R. B., AND III, J. E. B. Integration of computer security into the software engineering and computer science programs. *Journal of Systems and Software 49*, 2-3 (1999), 149–153.

[25] WIKIPEDIA. Covert channels — Wikipedia, the free encyclopedia. [Online; accessed 22-April-2012].

[26] WRAY, J. C. An analysis of covert timing channels. In *IEEE Symposium on Security and Privacy* (1991), pp. 2–7.

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

[28] 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] We do not know how they obtained the `/etc/shadow` file, which is usually only visible to the `root` user.