MICE: A Holistic Scorekeeping Mechanism for Cybersecurity Wargames*

Tristan Saldanha, Quinn Vinlove, Jens Mache
Lewis & Clark College, Portland, OR 97219
{tristansaldanha, quinnvinlove, jmache}@lclark.edu

Abstract

Cybersecurity wargames are some of the best tools for teaching security skills to groups of students, but the computational complexity of these games has increased disproportionately with the ability to measure the progress of the game. This paper introduces "Mice", a new way of assessing security skills such as detecting malware, network intrusion, and network defense, which will allow for complex games to be scored and tracked in a way that traditional score keeping can not. Mice are adaptable to any kind of simulation and are easy to use for students and educators, promising more effective learning from a wide range of security exercises.

1 Introduction

Network Wargames are highly complex tools for teaching offensive and defensive cybersecurity. In Wargame-like simulations, security students are placed together on a network and told to attack other hosts or defend from attacks from other players. However, the complexity of these simulations is often paired with a scoring system that can't keep up with the nuances of the game as it evolves. Players can tell when this mismatch occurs, and the result is that players will play the game at the level of the scoreboard, rather than at the higher level of the environment. This can leave overhead in the form of aspects of the virtualized environment that are not utilized by players; these unutilized parts of the environment resulting in the lost benefit to students and wasted effort

(and money) of educators. Limited grading schemes for wargames can defeat the purpose of simulating a real network. simple point markers like flags can be focused in on, allowing students to disregard the real system administration work of seeking out security problems as they appear. Wargames need a better scoring system that allows for a more realistic simulation that doesn't give students a short list of places to look for hackers, but rather encourages them to identify the signs of breach and work backwards to find malware. The system introduced in this paper, called Mice scorekeeping, is a new way of evaluating these kinds of wargames. The Mice solution is adaptable to any imaginable type of wargame; it can be used alongside other scorekeeping systems of an administrator's choosing, and can measure aspects of network control in a way that is fundamentally unique and currently unmeasurable by the flag or uptime based systems in use today. Mice, short for Miner-Imitating Counting Executables, are small programs that will add to a player's score for their time kept running and the amount of computing power they are allocated. A given singular "Mouse" will search for solved hashes, in the style of a Hashcash proof-of-work system [1], similar to how cryptocurrency mining works. Upon finding a solved hash, the Mouse will send proof of this solution to the scoreboard, thus incrementing a player's score proportionally to the approximate computing time needed to solve an average hash.

2 Capture the Flag Competitions and Current Scoring Mechanisms

CTFs, or Capture The Flag competitions, are potentially powerful vehicles for cybersecurity training [2, 4, 6, 3]. CTFs are usually built on task-specific challenges. Example problems in a CTF may be to reverse-engineer a password verification program, or to decrypt a specific conversation encrypted with an unknown algorithm. These simple tasks can be reasonably proven completed with a simple flag or text file at the end of the process- if a user can submit a plaintext string from an encrypted communication, this demonstrates that the student must have decrypted the conversation. Task-specific challenges like this can be easily proven completed because the task itself is straightforward; there are a small number of ways to complete the challenge. As more training scenarios have evolved to become more complex, assessing the state of the game has become more challenging . The CTF style flag-based proof of solution system does not translate well to a large scale live environment like a virtualized network with many hosts.

3 The Evolution of Live Testing Environments and Problem of Assessing Complex Security

Thanks to advancements in cloud computing, entire networks of computers can be virtualized to raise the bar of a capture the flag like exercise [5]. Instead of competition challenges being limited to working with small files like packet captures or individual programs, whole applications, hosts, and group of hosts can be virtualized and accessed over a web interface to engage students in more complex simulations and teach deeper skills. Examples of this new higher capacity simulation include websites like Hackthebox [9], EDURange [11], and OverTheWire [10], which let students test their penetration testing and system administration skills in a way that would be impossible in a file-specific exercise like classic capture the flags. These modern environments are more complex to evaluate than their simpler predecessors. Assessing the security of a host machine or the degree of intrusion and stealth on a network is difficult due to the variety of possible ways to engage with a host. A check that relies on looking for specific malware will miss newly discovered exploits, and measuring system uptime would disregard attackers who have exfiltrated data and locked in their own remote access, to name just a few possibilities. There is simply no one-size-fits-all solution to measuring the overall integrity of a network under attack. A grading system that lists specific criteria for a defending team will always result in a set of priorities being established and, by extension, designate a wide swath of network activity that can be ignored until something goes grossly wrong. This is presenting a misinformed view of security to students, who should instead be taught to seek out intrusion before the signs of it are obvious. Redesigning the scoring system for network wargames will result in better, more valuable exercises for students to learn from. The CCDC Competition [2] is a well known blue teaming exercise. The game is scored using primarily service uptime as the metric of a well-secured network, along with injects and writing incident reports to a lesser extent. Keeping services up and running is an aspect of network security, but this grading scheme identifies a list of services that allow blue teams to focus on a handful of tasks, while leaving the scoring system blind to whether an attacker has gained access to the network or not. A common scenario that occurs in these kinds of games is that the attacking team will take some time to establish their persistence and elevate privileges in host machines, before attacking scored objectives. This phase of the attack, when nothing has visibly gone wrong to the untrained sysadmin's eye, is being ignored by modern score systems, which is a disservice to both teams playing the game. Mice can fill this gap. Mice are less ambiguous in terms of detection - simple process monitoring commands will show the majority of mice running on a system. This limits the amount of stealth that the red team can use,

and gives blue teams more to look for. Speeding up the silent intrusion and elevation of privilege phase of the red team's gameplay improves the experience for both teams.

4 The MICE System

We present a new way of evaluating network security during live testing environments, using what we call Mice (henceforth referred to as a single Mouse program or several Mice programs, for readability). Students will be given small python programs that will add points to their score while running, called Mice. These Mice can be run on any machine on the given exercise network to add points to a player or team's score. The Mouse program is undergoing continuous active development [7], to keep the software as simple as possible while including all necessary functionality to be used effectively. The two variables that matter to a Mouse include the Mouse's ownership and it's speed. Each Mouse has an "owner", which is the player or team who receives the points that the Mouse generates while running. A player would want as many mice as possible to be running under their own ownership. The "Speed" of a Mouse is the rate at which it gains points. The more system resources a Mouse is given, the faster it will gain points. This incentivises players to be mindful of how hard they push their Mice; an unrestrained Mouse may utilize too many system resources and cause a user's machine to slow down or even crash. A hostile Mouse placed on an opponent's machine would have the same effect, which could result in detection of foreign Mice. This adds a level of complexity to a player's Mouse strategy; a quiet, undemanding Mouse may avoid detection and garner points slowly, while a Mouse taking lots of system resources to gain points fast may lead to its detection and removal.

5 Blockchain-inspired Mouse Design for Realism in Threat Modeling

While the Mouse scoreboard system itself does not maintain a complete blockchain, much of its design is owed to the proof-of-work mining process that makes Blockchains secure, or, colloquially, "mining". Mice measure their speed and contribution to player's score by computing hashes, such that the main objective is to keep as many of your Mice running as possible with as many system resources obtainable so as to mine the most hashes. The "points" that mice generate are measured in hashes. While a mouse is running, the work it does is search for a proof-of-work "solved" hash- that is, a hash of a random string that ends in an arbitrary number of zeroes. In practice, these solved states are called "solves" or "solved hashes". Since each character in

a SHA256 hash has a 1/16 chance of being any character, the probability of generating a hash ending in n many zeros is $(1/16)^n$. For a difficulty 5 hash, the MICE default, this is approximately one per million hashes, or one per megahash. When any mouse finds one of these solved hashes, it will send this solution to the scoreboard, and that mouse's team will be awarded one megahash of points. The blockchain style of the Mouse program has several benefits:

5.1 Realistic Threat Landscape

With the rising value of cryptocurrencies in the last few years, "Cryptojacking" is a very real threat to modern networks [8]. So-called Cryptojacking attacks involve attackers placing mining malware on remote servers to steal computing power for financial gain. Identifying and removing this kind of cryptocurrency mining malware is a real task that modern system administrators have to deal with. Building a network defense simulation around finding and destroying unwanted Mice on a host is a relevant primer for the current threat landscape experienced by network security professionals.

5.2 More Ways to Play

Using more complex programs as objects of the game, as opposed to finding text files on one another's machines, allows students to play their games in more nuanced ways. For example, rather than simply deleting a hostile Mouse found on one's machine, why not change its owner and make it gain points for your own team? Or better yet, modify a well performing player's Mice to belong to you, and let other players infect one another with your mice to build your own personal botnet. Using these programs in a more complex game allows players to utilize elements of reverse engineering, and grapple with the overlap between offensive and defensive security skills.

5.3 Simple Code

A priority during Mouse development is to keep the code simple enough that users can feasibly make modifications to the code during an ongoing exercise. The Mouse program is written in Python, which can be edited without needing to be recompiled, and is a high enough level language that modifying simple variables and methods can be done without necessitating an unreasonably high level of programming expertise. At the time of writing, the open-source Mouse program is receiving regular updates and improvements on its GitHub page, linked below.

5.4 Compatibility with Other Scorekeeping Mechanisms

The Mouse system is not perfect for some teaching objectives. The way that Mice go about incrementing points for a user is highly dependent on the hardware of the host they run on. In reality, the true value of an arbitrary machine may not directly correlate with its hash-solving potential. This is why the Mice system is made able to run alongside other, more traditional flag or service-uptime based systems. Nothing about the Mouse system limits educators from layering Mice on top of other systems that may better reflect the specific goals of a lesson- for a course in Windows Administration, a service-uptime scorekeeping style may be more effective than in a general pentesting simulation, while a Mouse system running on the side can also be in place to subtract student's points based on discovered remote code execution vulnerabilities. The ultimate purpose of the Mice is to aid educators in creating better simulations; to that end, flexibility and adaptability have been the main priorities through the development and design of the system.

6 Different Uses of Mice Scorekeeping

The rules for how to use the Mice can change to best suit the kind of war game being played. Mice are lightweight, simple, and their usage can be adapted based on the nature of the simulation they are applied to. Here, we will outline some ways that the Mice scoring system can be put to use in many different types of network security simulations.

6.1 Player versus Environment Simulation

For a player vs. environment simulation, like a firewall exercise, players may have Mice that they have to keep running against an incoming, possibly automated attack. In this case, a user would have only their own Mouse or Mice under their own ownership, and would have to keep their Mouse safe and running while defending from an incoming intruder looking to disable their Mice. This type of game may teach skills on the more defensive side of security, like obfustication of critical processes and host hardening. A player would also need to be able to detect intrusion as fast as possible, since any time with their mouse not running would cost them points.

6.2 Blue Team Versus Red Team Simulation

In a blue team / red team simulation, a blue team tries to keep the red team's score as low as possible while the red team tries to run their Mice on blue team machines. There may be only a single team of Red Mice, or for a more

complex game, a Blue and Red team of Mice. The blue team may be responsible for finding and stopping as many Mice as they can while having no Mice of their own to run, while the red team can only run Mice on the blue team's network. Or, if the simulation is desired to be more complex, there may exist both blue and red mice on the network, while Mice are only allowed to run on blue team hosts, thus making the players of both teams search for enemy mice on the limited number of hosts and disable them. This teaches the red team network intrusion and, more uniquely, stealth tactics, as the red team needs their Mice to go undetected for as long as they are able. This game would also be providing a valuable exercise for the blue team to practice identifying the signs of malware running on their workstations, with time being a factor in their response abilities.

6.3 Player versus Player Simulation

In a player vs. player scenario, players may be allowed to sabotage the other player's Mice and attempt to install their own Mice on other machines. This kind of simulation allows for the greatest range of creativity for students. This exercise teaches a huge range of skills, including workstation hardening, penetration testing, intrusion detection, reverse engineering, and more. It teaches the balance of offensive and defensive tactics in a way that is utterly unique to other forms of scoring, and allows a live network security simulation unparalleled by artificial tools and limited evaluation metrics.

6.4 More Possible Use Cases and Abstractions

More ways to use the Mouse system are discovered as the idea is shared with more educators with different experiences and expertise. Below are some ideas of ways to put the Mice system to work in a variety of more abstract types of simulations.

6.4.1 Firewalls

Imagine a networking game being played on a network with many hosts. Each host could have a mouse server running on it, but each behind different firewalls configured in different ways; an exercise could be made of "unlocking" these firewalls so that these scoreboard could be reached by Mice, with points representing uptime that the scoreboard was reachable.

6.4.2 Network Mapping / PING Sweeping

Another possible exercise could place scoreboards on many different ports of a given machine, encouraging students to learn to use tools like Nmap to discover

hidden scoreboards or mice. The score at the end of the game could in some way reflect the amount of mice that were linked to the right scoreboards; perhaps some mice would be best fit to certain boards, being locked to broadcast on a specific port within the python script. The possible implementations to the Mice system are truly limitless. The simple code makes the programs adaptable, and the uses of the hash-based scorekeeping are the perfect way of measuring time, persistence, and compute resources managed by a single player.

7 Classroom Tests

To simulate what a real world applications might look like, a group of research assistants played a sample game with the Mice programs. We played a free-for-all style of game; each of us had our own client machines on the same subnet. We played with 4 players, each of us given a machine on the same subnet. A scoreboard was set up on a fifth machine, and we agreed to leave it alone from tampering.

7.1 Software Sabotage

7.1.1 Hardcoded Variables

The Mouse program sets its team ownership through a command line argument. If this argument is altered, the mouse runs for a different team. One player modified an opponents mouse program to immediately discard this argument variable, and hardcoded their own team name inside the program.

7.1.2 Slowed Mining

Other software changes were made to other teams to damage their mice. One was slowed down by adding some extra work to the hash searching function. In this case, the variable "fish" was created, assigned the value "glub glub", and then discarded every time a hash was checked. This didn't affect the program in any other way than to roughly double the hash searching loop, slowing down opponent's point scoring.

7.1.3 Tampered Debug Outputs

Another change removed the debug outputs from one mouse program, requiring the student to spend time downloading a fresh copy of the Mouse program to understand what was going wrong with it as it was failing to run. One student went a step further by locking the debug outputs to show that a mouse was

running correctly, while all it was actually doing was printing the expected outputs and submitting no points.

7.2 Propogating Faulty Mice

This is the behavior that was most anticipated in the early phases of designing this mouse system. All four students attempted in some way to run their mice using the hosts of other machines. On the lab machines this test run was carried out on, all players knew all other user's credentials (username and password student), and both SSH and FTP were enabled on all machines. Some students copied their mice to other machines, while others just ran copies of the other player's mice under their own name.

7.3 Player Secured Mice

Given the above examples of players tampering with each other's mice, one player bypassed this risk by modifying his mouse and compiling it to a .pyc file before using it. The player hardcoded his own team name into that mouse and made sure that it was working properly, then played the game with his compiled mouse rather than the stock python file. Using the compiled version hardened his mice from attack, and he was able to use them without having to check if they had been modified while he wasn't watching.

8 Codebase

The code for the Mice system is being developed and is openly available in this github repository: https://github.com/kh3dron/mice-scoreboard. The code is still undergoing development at the time of this writing, but mainly feature updates; the current version of the system is ready to be put to use. Two Python scripts are involved in the Mice: the Mouse client, called mouse.py, and the scoreboard, called server.py. The client is a short program, at the time of writing only about 60 lines long. The program will search for a proof of work hash solution until one is found, at which point it will send the solve to the server, wait for positive confirmation of reception, then go back to searching for a new solve. The simplicity of the Mouse client introduces some fairly large security holes, which either team can exploit to their own advantage, or possibly repair to make the Mouse run more securely. While the Mouse development is ongoing, the goal is not to make the program immune from abuse. The simple and imperfect nature of the Mice introduces yet another nuance to their usage, encouraging a baseline of scripting knowledge as a way to give players an extra edge on one another during an exercise.

9 Upcoming Features

The Mouse and Scoreboard programs are both complete enough to be used as they are, but more updates are coming primarily to improve the data from the server. Some planned updates include: server-side difficulty requirement broadcasting to manage traffic and server logs/a proper administrator dashboard. A more detailed account of upcoming software additions can be found at the GitHub repository for the project, as linked above.

10 Acknowledgements

This material is based upon work supported by the National Science Foundation under grant numbers 1723705, 1723714, 1516100 and 1516730. We would like to thank Lewis & Clark students Alex Lotero and Kris Gado, as well as professor Richard Weiss from the Evergreen State College.

References

[1] Adam Back. Hashcash - a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`.

[2] CCDC. CCDC competition rules and guidelines, 2019. `https://www.nationalccdc.org/index.php/competition/competitors/rules`.

[3] Andy Davis, Tim Leek, Michael Zhivich, Kyle Gwinnup, and William Leonard. The fun and future of CTF. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.

[4] Patrick Hulin, Andy Davis, Rahul Sridhar, Andrew Fasano, Cody Gallagher, Aaron Sedlacek, Tim Leek, and Brendan Dolan-Gavitt. AutoCTF: Creating diverse pwnables via automated bug injection, 2017.

[5] George Louthan, Warren Roberts, Matthew Butler, and John Hale. The Blunderdome: An offensive exercise for building network, systems, and web security awareness, 2010.

[6] Cheung R. S., Cohen J. P., Lo H. Z., Elia F., and Carrillo-Marquez V. Effectiveness of cybersecurity competitions, 2012.

[7] Tristan Saldanha. MICE scoreboard codebase. `https://github.com/kh3dron/mice-scoreboard`.

[8] Symantec. What is Cryptojacking? how it works and how to help prevent it. `https://us.norton.com/internetsecurity-malware-what-is-cryptojacking.html`.

[9] Hack the Box. Hack the box, 2019. `https://www.hackthebox.eu/`.

[10] Over the Wire. Wargames, 2019. `https://overthewire.org/wargames`.

[11] Richard S. Weiss, Stefan Boesen, James F. Sullivan, Michael E. Locasto, Jens Mache, and Erik Nilsen. Teaching cybersecurity analysis skills in the cloud. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE 15*, 2015.