COMP 7800/7806 Spring 2021 Final Report (revised)

Competitive Coevolution for Continuous Authentication

Alaeddin Almubayed
Auburn University
aas0072@auburn.edu

Dennis Brown
Auburn University
dgb0028@auburn.edu

Dongji Feng
Auburn University
dzf0023@auburn.edu

Tripp Isbell
Auburn University
cai0004@auburn.edu

***Abstract*—Continuous authentication (CAuth) based on behavior has been explored as a promising solution to reduce the risk associated with session hijacking and credentials theft. Yet, such an authentication mechanism is susceptible to attacks. One potential solution is to discover behavioral patterns of attackers and defenders interacting in a CAuth simulated environment. Those interactions can be viewed as strategies generated by AI-driven attackers and defenders. Those strategies intend to inform hardening efforts that result in new defensive mechanisms that significantly increase the cost to attack CAuth systems. We present a novel approach that uses a competitive co-evolutionary algorithm to evolve genetic programs that express those behaviors as executable structures. We model CAuth as a game of attackers versus defenders in a dynamic and evolutionary arms race. This game should serve as a platform to simultaneously evaluate the effectiveness of attacks against CAuth systems alongside their resistance to such attacks. We introduce the Basic Architecture for Genetic Programming and Competitive Coevolution (BAGPACC) to implement this environment and run several experiments across relevant scenarios. Experimental results show that BAGPACC is sensitive to game parameters and evolves attacker and defender behaviors as decision tree-like genetic programs. As implemented, the results do not yield sophisticated behaviors that would translate into real-world strategies of attackers and defenders; the cause of this behavior and suggestions of future work to improve the situation are presented.**

***Index Terms*—continuous authentication, competitive co-evolution, machine learning, security.**

I. Introduction

Continuous Authentication (CAuth) is an extension of the user authentication process that operates transparently to the user throughout their entire session. It has the potential to significantly improve both security and usability, and can be used as a second line of defense against attackers who've managed to bypass initial authentication. Typically, CAuth is accomplished by observing some available metrics of current user behavior and determining whether they match those of the user's typical behavior. Because this situation becomes a bit of a complex classification problem, CAuth has seen extensive use of machine learning and AI over the past few decades.

As AI techniques are increasingly utilized in CAuth systems, so too will they be used by adversaries to thwart those systems. We model a game between attacker and defender agents in a simulated network that captures the dynamic behavior of such a CAuth arms race. Our attacker agent will attempt to masquerade as a legitimate user to carry out malicious activity while evading a CAuth system controlled by the defender agent.

We designed and built a competitive coevolutionary framework to coevolve agent controllers, the Basic Architecture for Genetic Programming and Competitive Coevolution (BAGPACC), leveraging the expressive richness of Genetic Programming (GP) to evolve the behaviors as executable structures that control the agents. These structures capture the logic used in engagements between attacking and defending agents. A successful implementation of this approach should reveal novel and effective courses of action for both attackers and defenders of simulated CAuth schemes, which will in turn inform improvements to real-world CAuth systems.

Our work concentrates on modeling CAuth as a game of attackers versus defenders in a dynamic and evolutionary arms race. This game should serve as a platform to simultaneously evaluate the effectiveness of attacks against CAuth systems alongside their resistance to such attacks. We use GP to express attacker and defender behaviors as executable structures that capture the logic of those agents as they assess the game state and decide what action to take. Those actions could be viewed as strategies generated by AI-driven attackers and defenders. The generated strategies are intended to inform hardening efforts that result in new defensive mechanisms that significantly increase the cost to attack CAuth systems. Those efforts should result in more robust AI-based CAuth systems that optimize security, convenience, and usability.

The rest of this paper is organized as follows: Section II overviews background on CAuth and adversarial attacks. Next, Section III discuss the related work. Section IV describes the methodology and implementation. Then, Section V outlines the experimental setup, followed by presentation of results in Section VI. Section VII discusses those results and we conclude with future work directions.

II. Background

Authentication is accomplished by verifying the user identity once via either static credentials or biometric means such as fingerprint or iris. CAuth was devised as means to improve authentication in response to the rising number of session hijacking and credentials theft. Along with it come promises to significantly improve the security and usability of authentication systems. CAuth can be based on physical biometrics or behavioral biometrics. Behavioral-based CAuth typically relies on Artificial Intelligence; this makes it susceptible to AI-based adversarial attacks. To help uncover patterns of the CAuth adversarial attacks, Saritaş et al. [16] tried to

model the interaction between an attacker and an operator using CAuth as a stochastic game. In the paper, the attacker observes and learns the behavioral patterns of an authorized user whom it aims at impersonating in CAuth system, whereas the operator designs the security measures to detect suspicious behavior and to prevent unauthorized access while minimizing the monitoring expenses. This work does not, however, reveal programmatic strategies of attackers and defenders.

Our goal is to discover novel and effective courses of action for both attackers and defenders of simulated behavioral-based CAuth systems. Competitive coevolution with GP can be employed to search the space of attacker and defender behaviors and identify programmatic strategies of those agents. GP is particular suitable to search the behavior space due to its rich expressiveness. The use of competitive coevolution with GP has been successfully established in academic research including applications to security problems, as described in Section III. However, to the best of our knowledge, our work is the first to use GP to model attackers and defenders as populations in a competitive co-evolutionary environment for CAuth.

III. Related work

In this section, we briefly review previous studies in CAuth research and adversarial work in CAuth, then review coevolution and GP related to security problems. Finally, we will discuss game theoretic applications in security.

Continuous Authentication: First, we look at prior work relating to CAuth. Over the past several years, CAuth has received increased attention both from industry and academia. Shen et al. [18] reported a novel approach by using a pattern-growth-based mining method to extract frequent mouse behavior segments to better account for mouse behavior variation. This work, along with further research [5], has improved the reliability of using mouse dynamics for the CAuth task. Although different strategies are used to implement the CAuth, using behavioral biometrics is still very common. Deutschmann et al. [8] focused on keystroke dynamic, mouse movements, application usage and system footprint to make a authentication system test, found that keystroke dynamics was the most appropriate for CAuth behaviormetric scenario. Now, with the application of machine learning technology in security, researchers are trying to implement adversarial machine learning to find vulnerabilities in CAuth. Kufel et al. in their thesis [12] carried out adversarial attacks against behavioral-based CAuth with the use of generative models. Tan et al. [22] built a set of attacks that are applications of several generative approaches to construct adversarial mouse trajectories that bypass authentication models.

Coevolutionary Algorithms and Genetic Programming in Security: Next, we look at prior work that implements coevolutionary algorithms and GP in the security domain. Competitive coevolutionary methods have seen a surge in popularity and applicability in the security domain due to their apt modeling of common attacker-defender scenarios. A classic example of competitive coevolutionary algorithms' usage is their application to the Iterated Prisoner's Dilemma [1], [2], which resulted in breakthrough discoveries in how cooperation evolves among populations. Hingston et al. [11] presented a coevolutionary algorithm designed to be used as a computational tool to assist in red teaming studies. Cardona et al. [6] used competitive coevolutionary algorithms to optimize Pac-Man and ghost controllers for the game Ms. Pac-Man. In the realm of network security, Rush et al. [15] proposed the coevolutionary agent-based network defense lightweight event system (CANDLES), a simulation-level framework that sits in the middle between abstract game-theory models [16] and more realistic emulation environments [17]. Bader et al. wrote about using genetic algorithms in a network intrusion detection system [21]. O'Reilly et al. presented an explanatory framework called RIVALS that uses GP and competitive coevolution for several security problems [13]. Schoonover et al. developed a network emulation framework designed to support rapid, parallel experimentation with the automated design of software agents in mind [17]. Furthermore, Harris et al. demonstrated how that emulation framework [17] can be used as a setting to evolve network scanning behavior [10].

What's in a game?: Game theory is frequently used to model cyber attack-defense scenarios. Of recent note, Bao et al. structured a game in which players discover zero-day vulnerabilities and either sit on them for exploitation or disclose them for patching [3]. Their work demonstrates how clever assumptions and game modeling can yield real-world insights. For example, they presented an improved strategy for Shellphish [19] from DARPA's Cyber Grand Challenge [7], which their model claimed was too eager to attack upon discovering new vulnerabilities [3].

Base Game by Saritaş et al.: Saritaş et al. provided a formulation of the CAuth problem where a defender attempts to distinguish rogue attacker behavior from benign user behavior [16]. They presented a dynamic (i.e., sequential) two-player game involving an attacker and defender. The defender controls a CAuth system with an enrolled user, which monitors the user's behavior and classifies it as legitimate or illegitimate. The environment (representing something like an enterprise network) is in one of three states at any given time-step:

1) *unblocking*, representing the normal state of the network;
2) *blocking*, in which the network is temporarily shut down due to a false positive — the CAuth system has observed behavior from the user that falls outside of a threshold that it considers "normal;" and
3) *attacker detected*, in which the attacker has tripped an IDS and lost access to the network, ending the game.

A user periodically generates traffic to interact with a resource, with successful interaction generating reward for the defender. The attacker starts off with no information on the user's behavior and has a low chance of success in impersonating the user. The attacker can listen to the user's traffic, carrying some risk of being detected but increasing their chance of successful impersonation. *Waiting*, *attacking* (impersonating), and *listening* are the three available actions for the attacker.

This design is obviously a simplification of a realistic CAuth system, but that allows the authors to derive an optimal strategy for the attacker: to listen until a critical threshold is reached and then attack, and furthermore, to identify that optimal attack threshold for a given set of game parameters through value iteration. We believe, to the best of our knowledge, this is the first work to implement GP to model attackers and defenders as populations in a competitive co-evolutionary environment for CAuth task.

IV. Methodology and Implementation

A. Game Details

We model our scenario as a two-player, sequential game between attacker and defender agents. The game plays out in rounds and is in either a `BLOCKED` or `UNBLOCKED` state during gameplay. A *round* of the game consists of a turn for each agent if the game state is unblocked, or only the attacker agent if the game is blocked. During the attacker's turn, they choose to attack, listen, or wait. The defender may choose to block or remain unblocked. Additionally, a round involves two environmental elements, a user and an Intrusion Detection System (IDS), that aren't controlled by the attacker or defender.

The attacker's active actions, both listening and attacking, make some amount of "noise" within the system. This noise has a chance of being detected by the IDS, which serves as our game-ending condition. Listening is detected with probability δ_l, and attacking is detected with probability δ_a (typically, $\delta_a > \delta_l$). Upon detection of either action, the game state transitions to `ATTACKER DETECTED` and promptly ends.

During each round, the user randomly generates traffic according to a Poisson process[1] with mean λ_u. If the user generates any traffic at all for a particular round ($T > 0$ for some $T \sim \text{Pois}(\lambda_u)$), a separate random variable enters the picture.[2] *Behavior* generated by the user appears to the CAuth system as a normal random variable β with mean μ and variance σ^2.

The attacker has three actions available: wait, listen, and attack. *Wait* is a filler action intended to be used when the game is in a blocked state. *Listen* is used to observe the user in the hopes of increasing the attacker's ability to mimic the user in further rounds. The total amount of traffic observed by the attacker is ω, and is the sum of traffic generated by the user during the rounds in which the attacker is listening. When the attacker is confident in their abilities, they can choose to attack. An attack, in our game, is just the generation of behavior in an attempt to mimic the user. The attack is considered successful if the defender is tricked into letting it through (i.e., not blocking).

[1]Note that this makes the particularly simplifying assumption that the user's traffic is independent of previously generated traffic.

[2]For clarity on the distinction, user *traffic* is what's observed by the attacker and what rewards the defender. User *behavior* is what's scored by the CAuth system from which the defender decides whether to block.

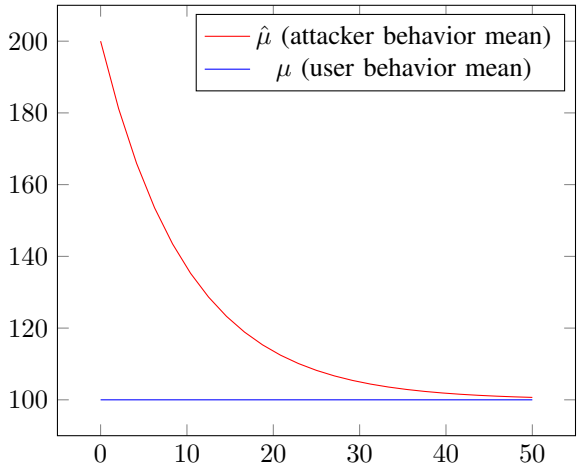

Fig. 1. The mean of the attacker's behavior as a function of the amount of traffic observed, compared with the mean of the user's behavior. For our default setup, we let $\mu = 100$ and $\gamma = 0.1$ as plotted.

More specifically, the attacker generates behavior with mean $\hat{\mu}$ and variance $\hat{\sigma^2}$, such that

$$\hat{\mu} = \left(1 + e^{-\gamma\omega}\right)\mu$$
$$\hat{\sigma^2} = \left(1 + e^{-\gamma\omega}\right)\sigma^2$$

where the factor that the attacker is "off" exponentially decays[3] as ω increases, scaled by a learning rate γ.

The defender reacts to the attacker or user-generated behavior and decides whether or not to block the system. Thus, the defender's actions are block or unblock, where the latter action just continues normal game operation into the next round. If the defender decides to block, the game state transitions to `BLOCKED` for the next round.

When the game state is `BLOCKED`, the user is prevented from generating any traffic with the system. The attacker may choose to attack or listen, but attacking is automatically unsuccessful and listening has no chance of observing user traffic where there isn't any. Therefore, the optimal choice in this state is to wait. The defender plays no role in unblocking the game and remains inactive while the game is blocked. For each round the game state is blocked, the game has probability q of automatically transitioning to unblocked for the next round.[4]

B. Steps of the Game

We now proceed to describe an unblocked round of the game in order.[5]

1) The attacker — aware of the current game state (`UNBLOCKED`), how much traffic they've observed, and

[3]herein lies another simplifying assumption that the attacker's behavior adjusts to match the user's so cleanly (Figure 1) as they observe user traffic

[4]Note that blocking is an equal hindrance on both the user and attacker; another aspect where the game falls short. In a realistic scenario, the time frames in which the user and attacker are attempting to interact with the system are unlikely to entirely overlap.

[5]The order of most of these steps is not dependent on all of the previous and could be switched around arbitrarily.

how much time (how many rounds) has elapsed — chooses their move.
2) The user, independent of the attacker (and all previous rounds), generates a random amount of traffic $T_t \in \{0, 1, 2, \ldots\}$. If the attacker is listening, this traffic is observed and contributes to ω.
3) If the attacker attacks, the attack generates behavior $\hat{\beta} \sim \mathcal{N}(\hat{\mu}, \hat{\sigma^2})$.
4) If the attacker does not attack but the user generates some traffic ($T_t > 0$), the user generates behavior $\beta \sim \mathcal{N}(\mu, \sigma^2)$.
5) The defender, in response to the behavior generated, decides to block or remain unblocked for the next round.
6) If the attacker chose to listen or attack, that action is detected by the IDS with its corresponding probability. If detected, the game transitions to `ATTACKER DETECTED` and we may as well forget the other events of this round (i.e. goto end).
7) If the attacker chose to attack and the defender does not choose to block, the attack is counted as successful.
8) If the defender decides to block, the game transitions to the `BLOCKED` state described in Section IV-A. Otherwise, the game continues unblocked into the next round.

Step 7 Bug: We note that our implementation of step 7 was fundamentally broken during our original pre-print of this paper. We discuss this further in VII-A, but in short, attacks that triggered a blocked state were erroneously counted as successful and added to the attackers reward. This was essentially game-breaking, and encouraged attackers that only attacked to take advantage of this bug.

C. Fitness Functions

At the end of each game, the participating attacker and defender are assigned a fitness value. The attacker's fitness is defined as

$$\sum_t r_t \rho^t$$

where r_t is an indicator variable (i.e., $r_t \in \{0, 1\}$) indicating a successful attack in round t, and ρ is a discount factor with $0 < \rho \leq 1$ which diminishes the reward from successful attacks the later they occur in the game.

The defender's fitness is defined as

$$\texttt{user\_bonus} \times \left(\sum_t T_t \right) - \texttt{attacker\_penalty} \times \left(\sum_t r_t \right)$$

where user bonus and attacker penalty are parameters used to balance the defender between favoring user interaction or attacker prevention. As the user bonus is increased, the defender's fitness should increasingly take into account the total amount of traffic generated by the user over a game, thus favoring defenders who err on the side of usability by reducing their false positive rate. As the attacker penalty increases, the defender's fitness becomes more weighted down by the total amount of successful attacks.

D. Competitive Coevolution

The attacker and defender that play the game are formed as genetic programs that are evolved through a process of competitive coevolution. This process is employed to search a space of candidate solutions (attacker and defender programs). To cast this problem as a competitive coevolution problem, we conceptualize two populations: attackers and defenders. Both populations are initialized with genetic material unique to each individual. Next, the evolutionary loop begins. Each member of each population is evaluated in the engagement (game) environment against a member of the opposing population in order to receive a fitness value, where the actions taken by each member are decided by their genetic program. Members of each population are selected for reproduction based on their fitness values, and the next generation of each population is created through genetic operations within that population. The genetic material passed from generation to generation contains elements of a program. The loop repeats with this new generation and each successive generation until a stopping condition is met. At this point, the best individuals of each population are presented for consideration as sufficient solutions to the problem.

This process is implemented in the Basic Architecture for Genetic Programming and Competitive Coevolution (BAGPACC). The major components of the architecture are shown in Figure 2.

- The *Competitive Coevolution with GP Strategy* module contains methods to perform the evolutionary functions of initialization, selection, mutation, and recombination within each of two *Populations* — collections of, and statistics about, individuals. The strategy executes the evolutionary loop and is highly configurable allowing selection from multiple alternatives for the evolutionary functions and the characteristics of the populations.
- The individuals contain *Controllers* that enumerate the supported actions of members of the Attacker and Defender populations and determine an individual's action during a turn of the game. The action an individual takes is determined by executing the genetic program of that individual. The programs use sensors that measure aspects of the game (separate sets for attackers and defenders) as variables in the program, and executing the program yields the next action to be taken in a round of the game. These programs are the genetic material of the members of the populations, and they evolve within this evolutionary framework.
- The *Game State* executes the CAuth game. Section IV-A goes into the specific implementation of the game, while this paragraph addresses how the game fits into the evolutionary framework. The game is set up to engage an individual attacker versus an individual defender at a time, and the game is sliced into rounds, where the attacker and defender each get one turn in a round. For its turn, the attacker will generate one of several actions such as *attack*, *wait*, and *listen* by executing its genetic

program, and likewise for the defender generating actions such as *block* or *unblock*. The game is stochastic and reinitialized every time it runs, ensuring that attackers and defenders don't evolve toward a singular scenario. The game ends when the attacker is detected, a convergence state is reached, or a configurable time limit has elapsed.

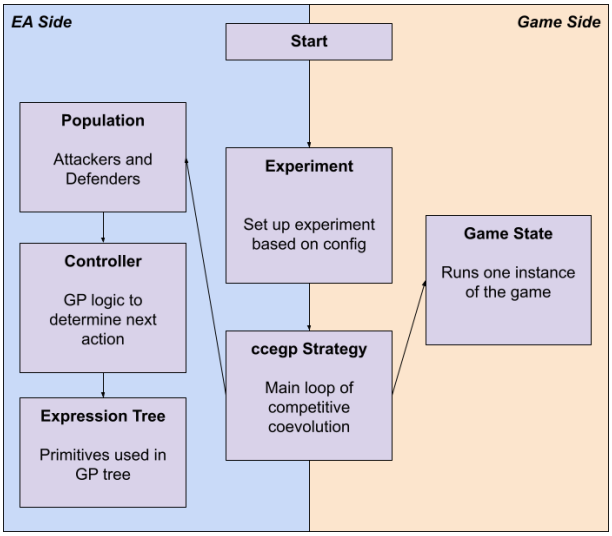

Fig. 2. Basic Architecture for Genetic Programming and Competitive Coevolution (BAGPACC)

E. Genetic Programming

The genetic material of each attacker and defender individual is a genetic program that drives the individual's behavior within the game. The genetic program is a binary decision tree. At each interior node of the tree, a condition that may involve sensor data from the state determines which branch is taken. Terminal/leaf nodes represent the action the individual will take in the game. This representation was chosen for its ease of human readability and that it naturally supports an arbitrary number of leaf-node actions. The grammar for the attacker is shown in Figure 3; the grammar for the defender is shown in Figure 4; and a sample attacker program is shown in Figure 5.

⟨*node*⟩ ::= ⟨*node*⟩ ↙ ⟨*bool*⟩ ↘ ⟨*node*⟩ | ⟨*action*⟩
⟨*bool*⟩ ::= '`if `'⟨*expr*⟩
⟨*expr*⟩ ::= ⟨*state*⟩ | ⟨*value*⟩ ⟨*compare*⟩ ⟨*value*⟩
⟨*state*⟩ ::= '`B`' | '`not B`'
⟨*compare*⟩ ::= '`>`' | '`<`'
⟨*value*⟩ ::= '`T`' | '`AO`' | '`AR`' | constant
⟨*action*⟩ ::= '`attack`' | '`listen`' | '`wait`'

Fig. 3. Grammar for Attacker programs. Sensor data: B = blocked?; T = time step; AO = amount of attacker observation; AR = amount of attacker reward

⟨*node*⟩ ::= ⟨*node*⟩ ↙ ⟨*bool*⟩ ↘ ⟨*node*⟩ | ⟨*action*⟩
⟨*bool*⟩ ::= '`if `'⟨*expr*⟩
⟨*expr*⟩ ::= ⟨*state*⟩ | ⟨*value*⟩ ⟨*compare*⟩ ⟨*value*⟩
⟨*state*⟩ ::= '`BM`' | '`not BM`'
⟨*compare*⟩ ::= '`>`' | '`<`'
⟨*value*⟩ ::= '`T`' | '`BH`' | constant
⟨*action*⟩ ::= '`block`' | '`unblock`'

Fig. 4. Grammar for Defender programs. Sensor data: BH = last observed behavior value from user or attacker; BM = user or attacker generated behavior; T = time step

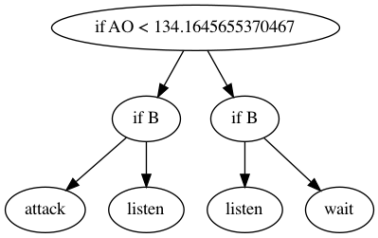

Fig. 5. Sample Attacker program generated from Scenario F. Depending on a threshold at 135 traffic observed by the attacker ("AO"), then branch to the terminal actions depending on the blocked/unblocked state of the game. This is something like the strategy we're looking for, but the threshold for attacking should be significantly lower. Also for this particular scenario, the detection risk from listening was low enough that listening during the blocked game in the left subtree is only marginally worse than waiting.

V. Experimental Setup

A. Experimental Scenarios

Because we don't have our system in a state where it's generating precise enough strategies for a fine-grained, nuanced analysis of parameters over a spectrum, we instead chose different configurations adjusted in coarse-grained sets of parameters organized into scenarios — based on scenarios that could happen in the real world — that were designed to evolve significantly different attacker and defender strategies.

Scenario A – Extremely Active User:

- The user generates a lot of traffic each turn: $\lambda_u = 3$
- The user generates traffic with high variance: $\sigma_u = 10$
- The system is frequently blocked, so it becomes unblocked relatively quickly: $q = 0.85$

How would we expect these changes to affect strategies? The defender should relax the detection cutoff (false positive rate). The user generates more traffic, so more traffic is missed during system blockages. The user's traffic varies wildly, which makes it more likely to trigger the cutoff. These make blocking unfavorable for the defender.

In response, the attacker should become more aggressive. Since the user generates traffic with high variance, they are easier to mimic. Essentially as the defender relaxes, the attacker should attack earlier (with fewer total listening observations).

Scenario B – Patient Attacker:

- The attacker has a long time horizon: $\rho = 0.995$
- The attacker makes little noise when listening: $\delta_l = 0.001$

These changes should promote a patient attacker that listens over a long period of time before attacking. Since the attacker has little IDS detection risk from listening, they should abuse that and gather a lot of user observation before attacking.

Scenario C – Skillful Attacker:

- The attacker learns a lot about the user per observation: $\gamma = 0.5$
- The attacker makes little IDS noise: $\delta_l = 0.01$ and $\delta_a = 0.05$

These parameters should promote both an attacker that listens early on, and an attacker that becomes aggressive early on. In contrast to scenario A, they should also encourage a stricter defender since the attacker will quickly become indistinguishable from the user.

Scenario D – Risk-averse Defender:

- The defender loses a lot of points if the attacker succeeds: `attacker_penalty` $= 1.2$
- The system stays blocked for a while if the defender suspects something: $q = 0.4$

These parameters should cause the defender to favor blocking. However, if the defender accidentally blocks the user, the system shuts down for a longer period of time, which adds counteractive pressure to avoid blocking. The attacker should also favor listening more, since attacking is likely to bring it down for a longer period of time.

Scenario E – Risk-taking Defender:

- The user must not be interrupted at all costs: `user_bonus` $= 0.15$

This parameter should balance the defender's reward so that it cares a lot more about successful user interaction rather than data breaches. We'd expect the same type of defender to come from this as scenario A, and, as a byproduct, a more aggressive attacker.

Scenario F – "Threshold" Defender Mode:

When we were originally building the game, we implemented a static defender that blocks any behavior that surpasses a predefined threshold. We used this defender to test our attacker's evolution before moving on to co-evolving both agents. Using this, we mistook our attacker's basic "attack" strategies and occasional variation as a sign that our implementation was at least working. Recently, we discovered a flaw in our implementation that severely impacted this mode, and further investigation led us to realize our assumption of a working game as a whole was flawed. For further discussion on this, see Section VII-A.

After fixing the game, we ran several experiments (both with coevolution and the threshold defender) to refresh our results. In the coevolved mode, the poor defender optimization still produces unimportant results that show no improvement over those in our original pre-print. However, we do note actual strategies being learned by the attacker in the threshold defender mode (where the only agent being evolved is the attacker).

We've configured the game parameters to heavily bias a certain strategy that we're looking for.[6] Namely, the attacker starts out with no chance of successfully impersonating the user without listening to some amount of traffic. This means strategies that don't involve some amount of "patience" will effectively generate no fitness. Our specific noteworthy parameters in this scenario are:

- The attacker has very little risk from listening: $\delta_l = 0.01$
- The attacker has high risk from attacking: $\delta_a = 0.3$
- The attacker's learning curve is set to the default settings and looks like Figure 1. In other words, the attacker should listen a significant amount before attacking.

From this we expect strategies that look somewhat like Figure 5, which is an example of one of the strategies generated by this mode.

B. Evolutionary Algorithm configuration

We ran one experiment for each of the six scenarios A through F listed in section V-A. Through small-scale testing, we found a configuration for the evolution that gives the best results. This configuration is used for all experiments; only the game parameters described in section V-A vary between the experiments. The parameters of the configuration are as follows:

- 30 runs of 10000 evaluations each (termination at 10000 evaluations = 67 generations)
- Timer-initialized random seed
- Maximum GP tree depth $D_{max} = 7$ for initialization
- Maximum GP tree depth $D_{max} = 9$ post-initialization
- Parent selection method: Overselection of top 32%
- Individual probability of mutation: 0.05
- Survival strategy: $\mu + \lambda$ with $\mu = 100$, $\lambda = 50$. μ is the size of the population that survives to the next generation; λ is the number of offspring generated in a generation; and "+" indicates both current-generation individuals and offspring are eligible for survival selection into the next generation.
- Survival selection method: Truncation
- Parsimony: Based on tree size (rather than depth) with a parsimony pressure penalty coefficient of 0.001
- Evaluation method: Every member of each population plays against at least one randomly-sampled member of the opposing population ensuring that each member of

[6]This might seem bad, particularly with the alarming phrasing, but there is a rationale for doing so. After fixing the game, we discovered that our default parameters were tuned so that an attacker under the optimal strategy had very little chance of every carrying out a successful attack, making our fitness function which relied on that very unreliable. Our evolutionary search could land on somewhat optimal strategies but they likely wouldn't generate any fitness under our evaluations.

each population plays at least one game. If a member plays multiple times, fitnesses are averaged. This method is incurs an elevated risk of random drift while providing fast enough execution times to make the run time reasonable.

VI. Results

First we look at the characteristics of the evolution of the attackers and defenders. We'll start by looking at the average fitness of the attacker and defender populations. Figure 6 shows the average fitnesses of the attacker and defender populations for the six experimental configurations, averaged over the 30 runs of each experiment. The attackers in scenarios A through E evolved against the defenders for approximately 1000 evaluations (7 generations) until falling sharply. The attackers in scenario F performed very poorly, hitting a small early peak before slowly falling back down. The defenders in all scenarios evolved against the attackers for approximately 2000 evaluations (13 generations) until leveling off. Figures 7 and 8 show, for the attackers and defenders, the averages and bests averaged over the 30 runs, along with error bars. The variance for the attackers is high and increasing in many cases.

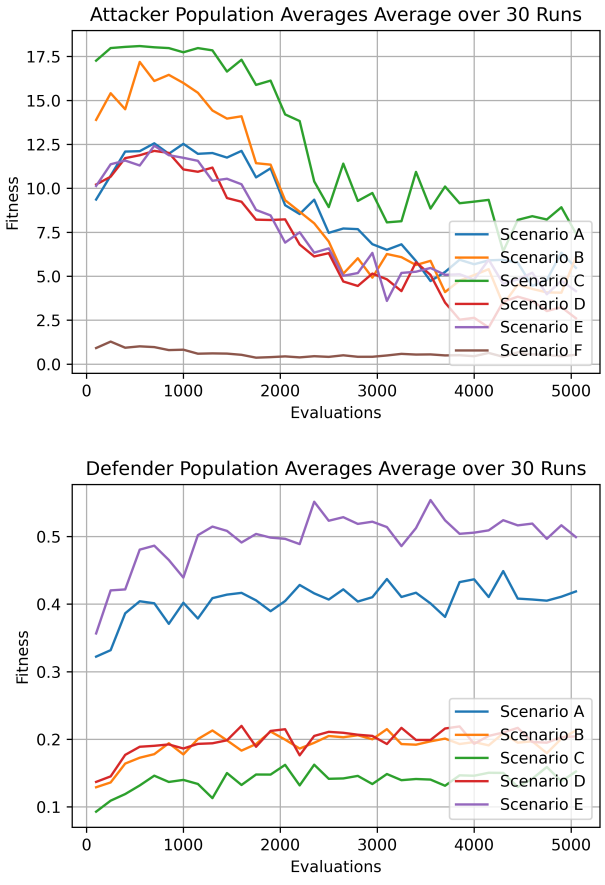

Fig. 6. Attacker and Defender population average fitness for all scenarios

We also consider the best attackers and defenders of each generation. Figure 9 shows representative Current Individual against Ancestral Opponents (CIAO) plots for all six scenarios. A CIAO plot shows the results of playing the best attacker of each generation against the best defender of all previous generations. The pixel color represents the outcome of each such match-up, where a white pixel means the attacker had the highest advantage over the defender and a black pixel means the attacker had the lowest advantage on a normalized scale. Understand that the colors are relative to that scenario only — colors are not comparable between plots. The lighter bars in Scenario C (skillful attacker) are brighter than the other scenarios, indicating a stronger performance of the best attackers against their defenders in Scenario C compared to the attackers and defenders in the other scenarios. However, the strongest conclusion to draw from these plots is their indication of cycling between the populations, where the genetic knowledge to succeed at the game seems to be lost and regained over time.

We looked at whether these attacker and defender populations are different, with statistical significance, between pairs of scenarios. Since the performance of one population in a scenario depends on the other, comparing across scenarios may be invalid; therefore we proceed with caution. Reviewing Figure 10, the attacker performance is largely indistinct between scenarios A, B, C, and E. That group of scenarios shows minor improvements over Scenario D (risk-averse defender) and huge improvements over Scenario F (threshold defender). For the defenders, the performance in Scenario A (extremely active user) stands out, as well as Scenario E (risk-taking defender). Figure 15 in the Appendix displays the results of the F- and t-tests for each pair of scenarios represented in Figure 10. Again, however, remember that the performance of one population is truly relative to the performance of the opposing population, so we consider that next for each scenario.

Next, we'll look at each scenario individually. Each subsection lists the overall "best" (highest-scoring) Attacker and Defender programs of each experiment. In almost all scenarios, the very simple one-line program of `ATTACK` provided the best results for the attacker, and in all scenarios where the defender evolved, the defender's best results came from another very simple one-line program, `BLOCK`. We suspect this phenomena is a result of how the decision tree form of genetic program (as we've constructed it) is too coarse a structure to thoroughly exploit the search space of behaviors.

Scenario A: Extremely Active User

- Best Attacker program of the experiment: `ATTACK`
- Best Defender program of the experiment: `BLOCK`

In this scenario, the attackers had relatively low performance in relation to other scenarios compared to the defenders' performance in relation to other scenarios. We suspect the aggressiveness of the attacker revealed enough behavior for the defender to detect it. The best programs are one-liners so they don't offer much insight into the behavior of attacker and defender that could be applied in a real-world situation.

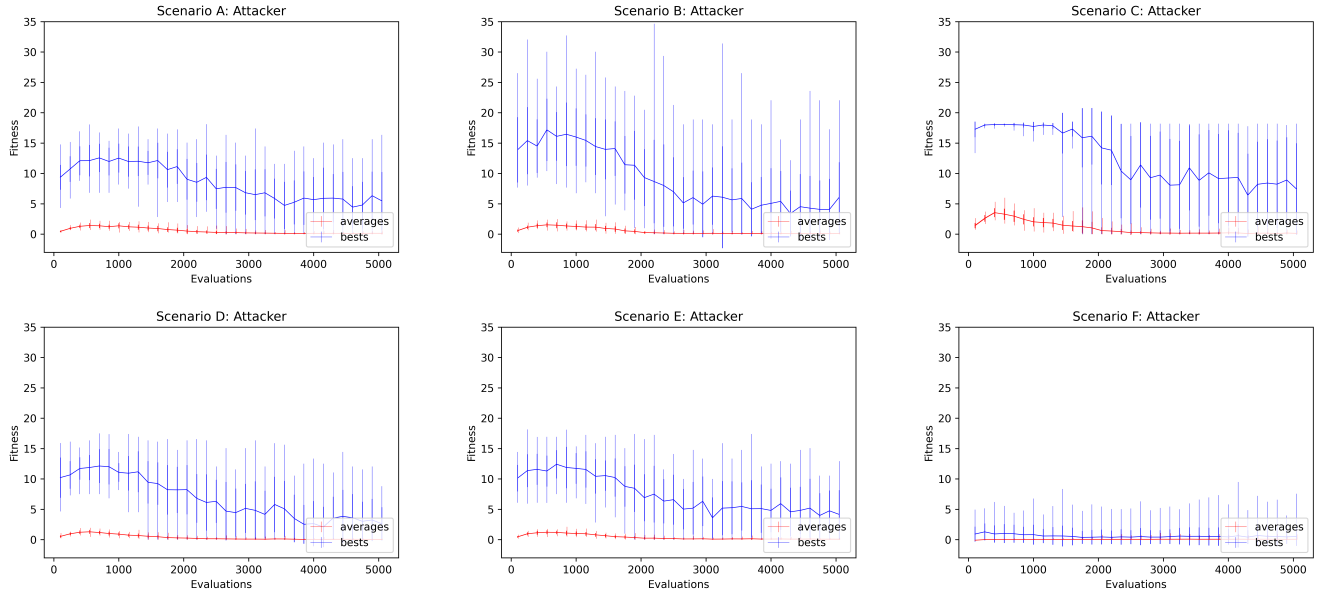

Fig. 7. Attacker averages and bests, averaged for 30 runs, for each scenario A through F

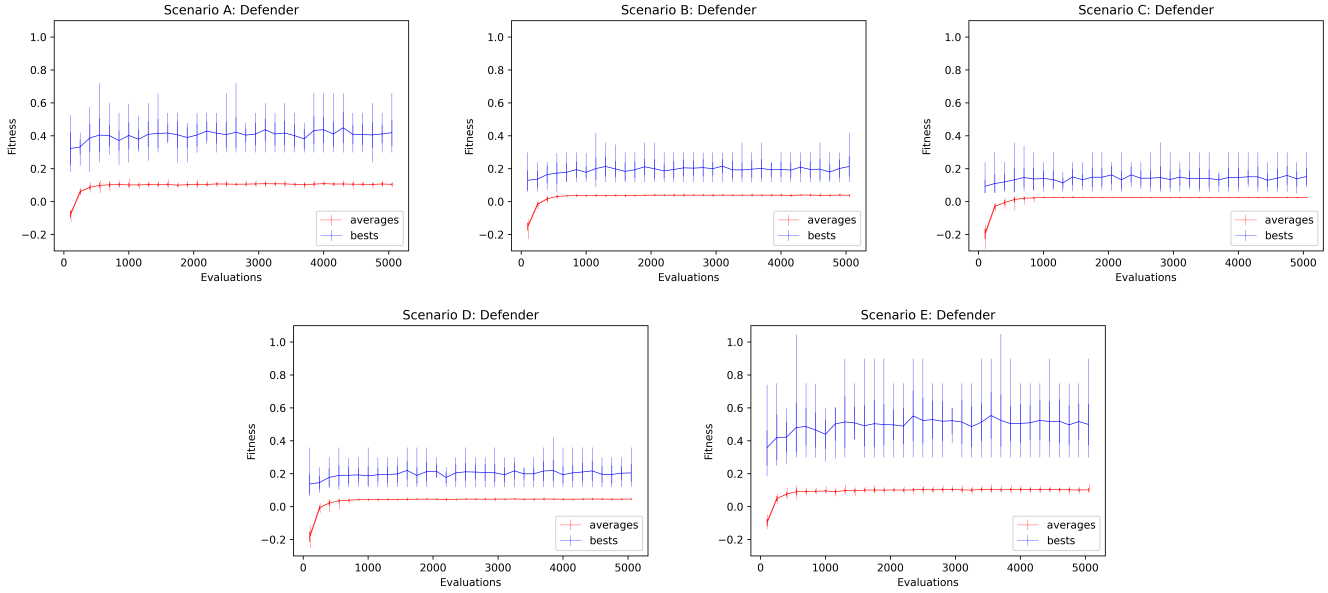

Fig. 8. Defender averages and bests, averaged for 30 runs, for each scenario A through E (F not included since defender was not evolved)

Scenario B: Patient Attacker

- Best Attacker program of the experiment: `ATTACK`
- Best Defender program of the experiment: `BLOCK`

Scenario B set conditions for a less aggressive / more patient attacker. The results show the second-best performance of the attackers and low performance of the defenders, indicating this strategy (patience, listening) is effective, at least in the simulation environment.

Scenario C: Skillful Attacker

- Best Attacker program of the experiment: `ATTACK`
- Best Defender program of the experiment: `BLOCK`

Scenario C aimed to model a skillful attacker, and the results show this attacker is more effective against its defender than the attackers and defenders in all other scenarios. Evidence is in the fitness averages in Figure 6, where Scenario C shows a high attacker fitness and low defender fitness, and in the CIAO plots in Figure 9 where scenario C is lighter (attacker has the highest advantage) than the other scenarios. However, the best programs are still one-liners so they don't offer much insight into the behavior of attacker and defender that could be applied in a real-world situation.

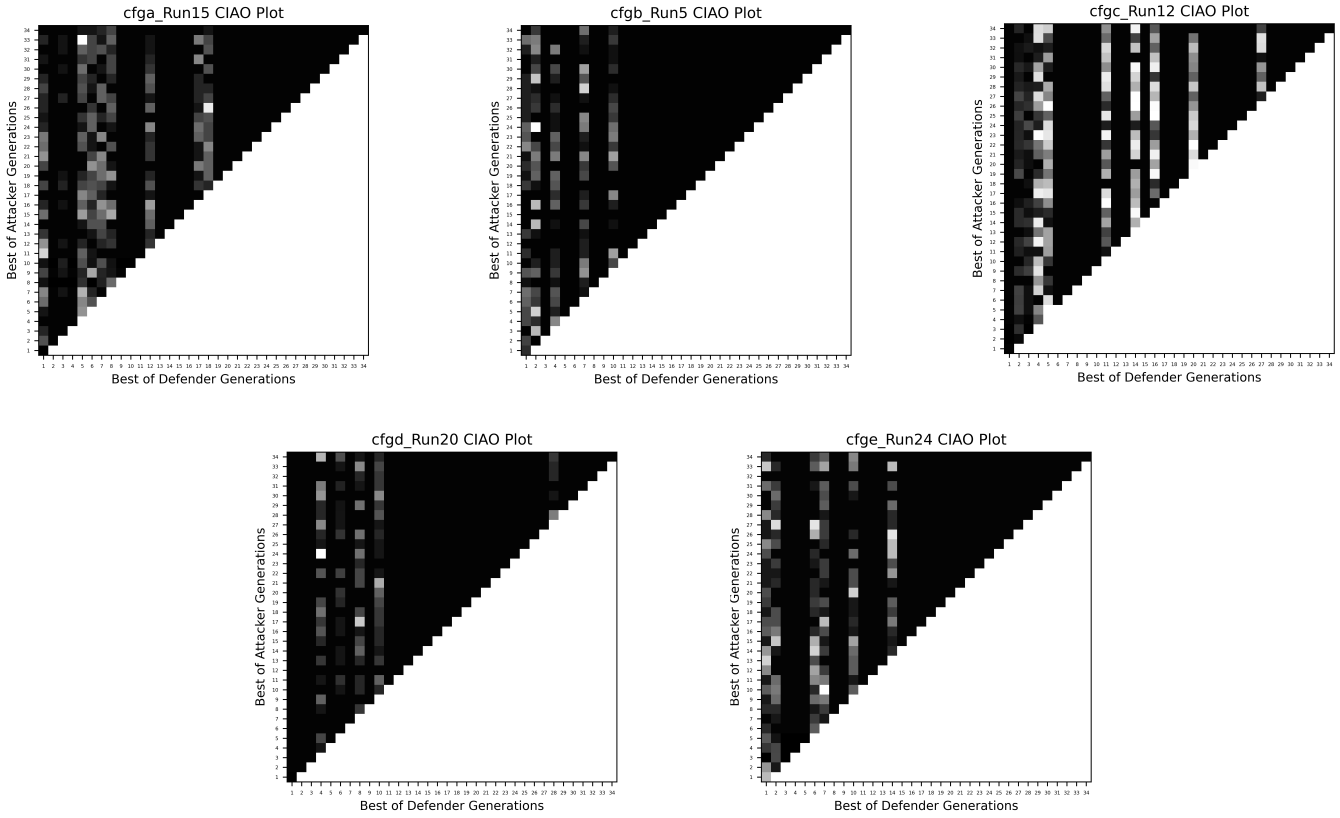

Fig. 9. Representative CIAO Plots for each scenario A through E (F not included since defender was not evolved). Lighter color = better performance of the attacker.

Attacker	B	C	D	E	F
A			110%		934%
B			133%		1047%
C			185%	79%	1304%
D					392%
E					684%

Defender	B	C	D	E	F
A	97%	175%	105%	-16%	51%
B		40%		-57%	-23%
C			-25%	-69%	-45%
D				-59%	-26%
E					79%

Fig. 10. Statistical differences between attackers across scenarios and defenders across scenarios. A number in a cell represents the percent improvement of the scenario in the row over the scenario in the column. No number in a white cell indicates no statistical difference per F- and t-tests. Black cell = N/A.

Scenario D: Risk-averse Defender

- Best Attacker program of the experiment: `ATTACK`
- Best Defender program of the experiment: `UNBLOCK`

In this risk-averse scenario, looking at the fitness averages in 6, the defender did keep the attacker's fitness among the lowest of the scenarios in which we evolved a defender. Again, the best programs are one-liners.

Scenario E: Risk-taking Defender

- Best Attacker program of the experiment: `ATTACK`
- Best Defender program of the experiment: `BLOCK`

The risk-taking defender had the highest performance of all scenarios and kept attacker performance among the lowest of all evolved scenarios. This result is also counter-intuitive: the risk-taker keeps the network open at all costs. We are not sure why this defender would be the most successful. Since the best evolved programs are again one-liners, they do not provide insight into this observed behavior.

Scenario F: "Threshold" Defender mode

An actual attacker generated by this method is shown in Figure 11. The process that generates decision trees is only very basically optimized and still produces redundant trees by including non-value-added conditional nodes. Figure 12 zooms in on a section of the attacker tree. One will notice the tree asking `if T > AO` three times in a row, even after it has been determined to be false. The left-hand side of this attacker program essentially reduces to "if blocked then wait."

For the most part, our search is very unreliable at the moment (Figure 13), as most attackers don't generate any fitness under the newly fixed game (maxing out at the single node trees with the least amount of parsimony pressure). We sporadically find attackers that generate fitness, which all take the form of "listening" for some period of time before attacking, but we don't successfully utilize those findings to improve our search (for reasons we suspect are partly due to issues discussed in Section VII-B).

VII. Discussion

Our results are not great. Both before and after the fix, we were unable to devise scenarios that yielded steadily increasing fitness as they went on. This is especially apparent in the scenarios involving coevolution (A-E), as the defender frequently converged to only blocking which halted the evolution on both sides. For scenario F (i.e., non-coevolution easy mode), we are able to at least see the kind of attacker strategies we expect by adjusting some parameters. However, these certainly aren't the kind of interesting strategies we were going for.

A. Uncovered Ground [7]

Upon review of our implementation since initial pre-print, we came across a major issue with the "threshold" defender mode of our game. The threshold at which the defender blocks unusual behavior was derived from derived from the standard normal distribution, not the one defined by user behavior. As evident by figure 14, this completely broke that mode of the game as the defender effectively reacted to anything with a block. The attacker strategies we thought we were evolving were us seeing patterns in the stars; they were effectively meaningless as it didn't matter what action the attacker took.

This illuminated a deeper issue, however. Why was the attacker generating any reward *at all?* If the game is eternally blocked, we should see that reflected in the attacker's scores. Examining the code more closely, we realized an incorrect implementation of step 7 in Section IV-B: there was nothing stopping the attacker from collecting reward upon attacks that were subsequently blocked. In the most simplistic of strategies, the attacker could exploit this bug by all-out attacks, which would collect reward regardless of consequential but temporary blocks.

That's not to say that our approach was working all along. Such an exploit would be found by monkeys on a typewriter. As shown by our revised results (Section VI), our work is far from working. In the static, threshold defender mode, our findings of plausible attacker strategies are sporadic and inconsistent across experiments. The main indication we have that things are at least partially working now is that the typical strategies evolved by the attacker are at least adaptive to the extreme parameter tuning changes we provide. This puts us at effectively a new starting point from which to work on our optimization. We discuss some ideas on that in Section VII-B, but they're small things unlikely to lead to big improvement.

[7] Also known as the "this didn't quite fit in our methodology section in short time but we didn't know where else to put it" section, or "no research paper would ever include this tidbit so there's no precedent, but we thought it important for our situation."

B. Where could we improve?

Our results, and our thoughts on how we could improve them over the past few months have produced many ideas for improvement, a few of which we note here.

Our decision-tree approach, while interesting, is not optimized nor particularly appropriate in both the attacker and defender's situations. While we were originally drawn to the idea because of the simplicity of our actions, the features that the attacker and defender operate with aren't very conducive to the approach. Decision trees would be more appropriate were we dealing with categorical, discretized variables.

Additionally, not all decision trees that we're evolving are meaningfully different from one another. There are the obvious redundancies, such as subtrees that only evaluate to one action at all of their leaves. We can prune such trees easily, but there are further redundancies due to "always true/false" conditions arising from the game state. Detecting these kinds of redundancies becomes significantly complex. We could use heuristics based on the frequencies the branch is taken, but that would remove branches that might *eventually* be taken, such as time-based conditions.

To improve our decision tree searching, we also need better constant generation and mutation. A lot of the strategies we're trying to evolve rely on sensitive thresholds (the attacker should begin attacking after observing X user traffic). However, since we pick constants from a large uniform interval, we have a low chance of generating particularly useful constants. It might have made sense to pick constants from a range dependent on certain game parameters. For example, as we increase the difficulty for the attacker to mimic the user, perhaps the constants we generate should be on average higher to compensate for the attacker's need to gather more observations. In other words, if we know certain changes increase the threshold we're searching for, we should adjust our search space accordingly.

Also, we don't currently mutate constant values. Once they're in the tree, they're final. We think we could improve our searching by using an additional mechanism to nudge constants somehow during mutation while maintaining the structure of the tree. If we find good constants that are close to what we want, we'd do better to adjust from there rather than trying to generate better constants randomly.

We don't take into account the probability of generating different decision trees. The frequencies of our search space are not equal. We generate trees using two different approaches of equal probability: growing them randomly from the root, or randomly filling in full binary trees. So, we're less likely to land on mid-size trees than we are single-node trees. As a result, the simple one-node '`attack`' trees might be beating out better strategies just due to the frequency that they appear.

Of course, these are all nuanced issues and addressing them

Fig. 11. Shape of the Best Attacker from Scenario F (not intended to be fully legible)

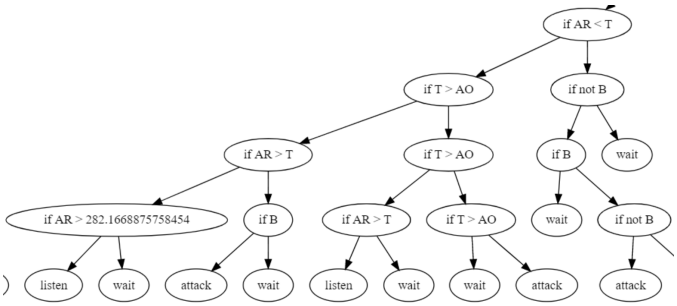

Fig. 12. Excerpt of Scenario F's best attacker

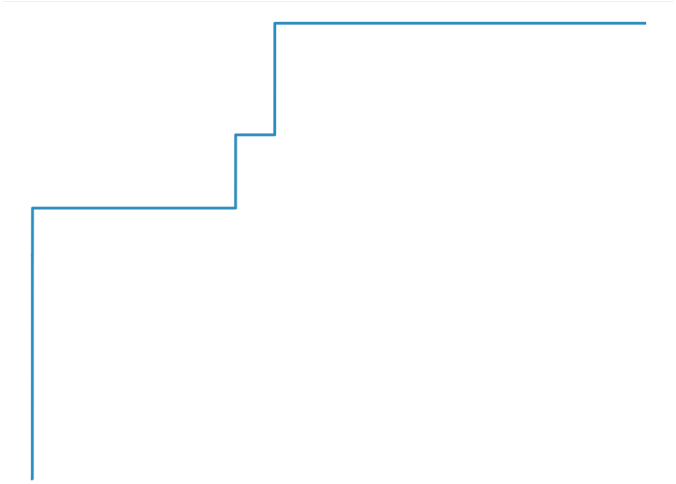

Fig. 13. The "learning curve" of our evolved attackers, approximated by the best fitness evaluation seen so far over the course of about 1.5 million games. This shows the sporadic attacker optimization we're getting under scenario F. In other words, we don't take a newly found strategy and make incremental increases based on that strategy. Or, in other words, the chance of us finding the optimal strategy after 5000 games is not much better than the chance of finding it after 1 game.

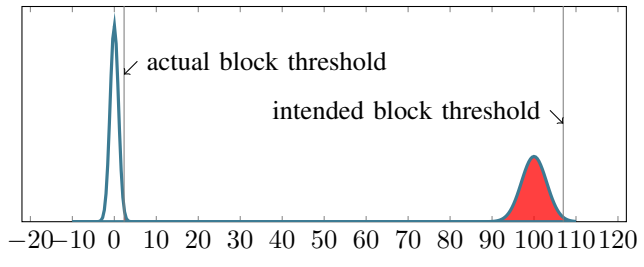

Fig. 14. The block threshold of the static "threshold" defender mode was erroneously tuned to the standard normal distribution.

might be to minimal effect.[8] We aren't, after all, entirely sure that the game state is correct, especially considering our discovery of the game breaking bug. Our biggest issues ~~might be~~ were hiding in plain sight while we searched for minimal optimizations. We could fix parts of the game, further adjust it to fit our approach, or maybe make it a little more realistic, but that all leads to an important question.

C. Is this worth it?

There have been many attempts to replace passwords in the name of usability and security [4]. Behavior-based CAuth is a class of such attempts with a significant amount of research (Section III). However, in the typical desktop scenario we've envisioned for our experiments, we think the only biometric indicators that might be useful to infer unique user signatures are human interactions with the keyboard and mouse.[9] A full survey of the state of the art is needed to make a proper argument, but we (the authors) don't see behavior-based CAuth replacing passwords in such desktop environments any time soon.

Thus, we ask the reader to take our claim for granted that CAuth systems will only supplement other initial-access authentication mechanisms for the foreseeable future. If that is the case, how much value is added by studying attacks on these systems? It seems reasonable for system designers to exclude particular things like replay attacks and adversarial AI from their threat model. In other words, CAuth might be best purposed at catching low-hanging fruit in a defense in depth approach.

Regardless, until CAuth systems enjoy widespread use in practice, [10] there is a lack of motivation for research in attacks on such systems. We know of a few examples [14], [20], [22], but the overall state of the research appears to be in its infancy.

Does it make sense, then, to model the dynamics of CAuth attacks when we don't know what these attacks may eventually look like? For example, we assume attacks carried out by an adversary capable of monitoring a specific user and using that capability to mimic that user. What if the best attacks are white box attacks in which the user isn't important? Or, what if the adversary is capable of piggybacking off of an active user's session without inducing any sort of behavior detectable by the CAuth system?

[8] Especially considering most of these ideas address decision trees, while the first idea indicates we're looking to move away from them entirely

[9] Behavioral biometrics might be more promising in devices like phones and wearables with a larger range of sensors.

[10] We don't actually know how common CAuth systems are in the wild, so take this claim with a grain of salt.

Is research better served to focus on improving initial access authentication (i.e., passwords, hardware keys, MFA, etc.)? If that were a solved problem, our work here (or work following down this path) would be made obsolete, as we assume an adversary capable of bypassing initial authentication.

As we think more about the kind of attacks we're modeling here, they become increasingly restrictive to extremely specific scenarios. The adversary needs access to the user's credentials and needs to be able to monitor the user. The setting needs to be some kind of enterprise network with a CAuth system in place that is monitoring user behavior. The adversary likely needs to login and carry out the attack through the user's computer, and further, actually interact with whatever behavioral authentication mechanism is implemented (e.g., the keyboard or mouse). And, if anything suspicious occurred, the entire attack could be thwarted by something like a login challenge (a defense which, we note, appears both more effective and more practical in general scenarios than behavior-based CAuth [9]).

VIII. Conclusion

We have discussed the application of competitive co-evolutionary algorithms to study CAuth scenarios. based on our analysis, we argue that existing co-evolutionary algorithms can be helpful in analyzing CAuth scenarios, in particular, the options available for attackers and defenders in a dynamic and evolutionary arms race. We discussed what choices that are likely to make a strategy strong and what weaknesses need to be kept in mind. Our results show that in almost all scenarios, the best attackers evolved to simply *ATTACK* and the best defenders evolved to simply *BLOCK* or *UNBLOCK*. We intended to evolve rich and expressive genetic programs and ended up with one-liners. As implemented, the results do not yield sophisticated behaviors that would translate into real-world strategies of attackers and defenders. We suspect that by expressing the genetic program as a limited form of decision tree, as we implemented it in this work, it is too coarse a structure to thoroughly exploit the search space of attacker and defender behaviors. We presented potential remedies to this problem that could be explored in further revisions of this work.

IX. Future Work

One interesting future direction will be to utilize the ML classification function on defender side and involve the intelligent insight into our gameplay. If so, we need to use a comprehensive dataset which our classification model can use to learn the feature of different users, for instance, a keystroke dynamic or mouse movement dataset that can help to verify the identify of users. Another direction is to implement the generative adversarial network as the baseline environment to involve the attacker and defender.

Acknowledgment

We would like to express our gratitude to Daniel Tauritz and Drew Springall for their enormous encouragement and motivation throughout the course. Without their guidance and assistance, this research would not have come to fruition. We are also very thankful to COMP 7800/7806 class for the insightful and valuable discussions on both security and AI, as well as the constructive feedback and suggestions throughout the research.

References

[1] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, NY.

[2] R. Axelrod and W. D. Hamilton. The Evolution of Cooperation. *science*, 211(4489):1390–1396, 1981.

[3] T. Bao, Y. Shoshitaishvili, R. Wang, C. Kruegel, G. Vigna, and D. Brumley. How shall we play a game?: A game-theoretical model for cyber-warfare games. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 7–21, 2017.

[4] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Security & Privacy (Oakland 2012)*, pages 553–567, 2012.

[5] Z. Cai, C. Shen, and X. Guan. Mitigating Behavioral Variability for Mouse Dynamics: A Dimensionality-Reduction-Based Approach. *IEEE Transactions on Human-Machine Systems*, 44(2):244–255, 2014.

[6] A. B. Cardona, J. Togelius, and M. J. Nelson. Competitive Coevolution in Ms. Pac-Man. In *2013 IEEE Congress on Evolutionary Computation*, pages 1403–1410. IEEE, 2013.

[7] Defence Advanced Research Projects Agency (DARPA). Cyber Grand Challenge (CGC) (Archived), Aug. 2016. https://www.darpa.mil/program/cyber-grand-challenge.

[8] I. Deutschmann, P. Nordström, and L. Nilsson. Continuous Authentication Using Behavioral Biometrics. *IT Professional*, 15(4):12–15, 2013.

[9] P. Doerfler, K. Thomas, M. Marincenko, J. Ranieri, Y. Jiang, A. Moscicki, and D. McCoy. Evaluating Login Challenges as A Defense Against Account Takeover. In *The World Wide Web Conference*, WWW '19, page 372–382, New York, NY, USA, 2019. Association for Computing Machinery.

[10] S. Harris, E. Michalak, K. Schoonover, A. Gausmann, H. Reinbolt, J. Herman, D. Tauritz, C. Rawlings, and A. S. Pope. Evolution of Network Enumeration Strategies in Emulated Computer Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1640–1647. Association for Computing Machinery (ACM), 2018.

[11] P. Hingston and M. Preuss. Red Teaming with Coevolution. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1155–1163. IEEE, 2011.

[12] M. Kufel. Adversarial Attacks against Behavioral-based Continuous Authentication, 2020. https://web.archive.org/web/20210225185644/https://kth.diva-portal.org/smash/get/diva2:1498889/FULLTEXT01.pdf.

[13] U. O'Reilly, J. Toutouh, M. A. Pertierra, D. P. Sanchez, D. Garcia, A. E. Lugo, J. Kelly, and E. Hemberg. Adversarial Genetic Programming for Cyber Security: A Rising Application Domain Where GP Matters. *Genetic Programming and Evolvable Machines*, 21(1-2):219–250, 2020.

[14] K. A. Rahman, K. S. Balagani, and V. V. Phoha. Snoop-Forge-Replay Attacks on Continuous Verification With Keystrokes. *IEEE Transactions on Information Forensics and Security*, 8(3):528–541, 2013.

[15] G. Rush, D. R. Tauritz, and A. D. Kent. Coevolutionary Agent-Based Network Defense Lightweight Event System (CANDLES). In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, page 859–866, New York, NY, USA, 2015. Association for Computing Machinery (ACM).

[16] S. Sarıtaş, E. Shereen, H. Sandberg, and G. Dan. Adversarial Attacks on Continuous Authentication Security: A Dynamic Game Approach. In *Proceedings of the 10th International Conference on Decesion and Game Theory for Security (GameSec 2019)*, volume 11836 of *Lecture Notes in Computer Science*, pages 439–458, Stockholm, Sweden, 2019.

[17] K. Schoonover, E. Michalak, S. Harris, A. Gausmann, H. Reinbolt, D. R. Tauritz, C. Rawlings, and A. S. Pope. Galaxy: A Network Emulation Framework for Cybersecurity. In *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*, Baltimore, MD, Aug. 2018. USENIX Association.

[18] C. Shen, Z. Cai, and X. Guan. Continuous Authentication for Mouse Dynamics: A Pattern-growth Approach. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12. IEEE, 2012.

[19] Y. Shoshitaishvili, A. Bianchi, K. Borgolte, A. Cama, J. Corbetta, F. Disperati, A. Dutcher, J. Grosen, P. Grosen, A. Machiry, C. Salls, N. Stephens, R. Wang, and G. Vigna. Mechanical Phish: Resilient Autonomous Hacking. *IEEE Security & Privacy*, 16(2):12–22, 2018.

[20] J. Solano, C. Lopez, E. Rivera, A. Castelblanco, L. Tengana, and M. Ochoa. SCRAP: Synthetically Composed Replay Attacks vs. Adversarial Machine Learning Attacks against Mouse-Based Biometric Authentication. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, AISec'20, page 37–47, New York, NY, USA, 2020. Association for Computing Machinery.

[21] E. Talal Abdel-Ra'of Bader and H. Nasereddin. Using Genetic Algorithm in Network Security. *International Journal of Research & Reviews in Applied Sciences*, 5(2):148–154, 2010.

[22] Y. X. M. Tan, A. Iacovazzi, I. Homoliak, Y. Elovici, and A. Binder. Adversarial Attacks on Remote User Authentication Using Behavioural Mouse Dynamics. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2019.

APPENDIX

Actor	Scen 1	Scen 2	Mean 1	Mean 2	Var 1	Var 2	Obs	df	F	P one-tail	F crit	alpha	Equal vars?	t Stat	P two-tail	t crit	Sig diff?	Diff
Attacker	A	B	5.47	6.07	23.52	34.63	30.00	29.00	0.68	0.15	0.54	0.05	FALSE	-0.43	0.67	2.00	FALSE	-0.59
Attacker	A	C	5.47	7.43	23.52	58.67	30.00	29.00	0.40	0.01	0.54	0.05	FALSE	-1.18	0.24	2.00	FALSE	-1.96
Attacker	A	D	5.47	2.61	23.52	7.77	30.00	29.00	3.03	1.00	0.54	0.05	FALSE	2.81	0.01	2.00	TRUE	2.87
Attacker	A	E	5.47	4.15	23.52	16.69	30.00	29.00	1.41	0.82	0.54	0.05	FALSE	1.14	0.26	2.00	FALSE	1.32
Attacker	A	F	5.47	0.53	23.52	2.49	30.00	29.00	9.43	1.00	0.54	0.05	FALSE	5.31	0.00	2.00	TRUE	4.94
Attacker	B	C	6.07	7.43	34.63	58.67	30.00	29.00	0.59	0.08	0.54	0.05	FALSE	-0.77	0.44	2.00	FALSE	-1.36
Attacker	B	D	6.07	2.61	34.63	7.77	30.00	29.00	4.45	1.00	0.54	0.05	FALSE	2.91	0.01	2.00	TRUE	3.46
Attacker	B	E	6.07	4.15	34.63	16.69	30.00	29.00	2.07	0.97	0.54	0.05	FALSE	1.47	0.15	2.00	FALSE	1.92
Attacker	B	F	6.07	0.53	34.63	2.49	30.00	29.00	13.89	1.00	0.54	0.05	FALSE	4.98	0.00	2.00	TRUE	5.54
Attacker	C	D	7.43	2.61	58.67	7.77	30.00	29.00	7.55	1.00	0.54	0.05	FALSE	3.24	0.00	2.00	TRUE	4.83
Attacker	C	E	7.43	4.15	58.67	16.69	30.00	29.00	3.52	1.00	0.54	0.05	FALSE	2.07	0.04	2.00	TRUE	3.28
Attacker	C	F	7.43	0.53	58.67	2.49	30.00	29.00	23.53	1.00	0.54	0.05	FALSE	4.83	0.00	2.00	TRUE	6.90
Attacker	D	E	2.61	4.15	7.77	16.69	30.00	29.00	0.47	0.02	0.54	0.05	FALSE	-1.71	0.09	2.00	FALSE	-1.54
Attacker	D	F	2.61	0.53	7.77	2.49	30.00	29.00	3.12	1.00	0.54	0.05	FALSE	3.55	0.00	2.00	TRUE	2.08
Attacker	E	F	4.15	0.53	16.69	2.49	30.00	29.00	6.69	1.00	0.54	0.05	FALSE	4.53	0.00	2.00	TRUE	3.62
Defender	A	B	0.42	0.21	0.01	0.00	30.00	29.00	1.66	0.91	0.54	0.05	FALSE	11.07	0.00	2.00	TRUE	0.21
Defender	A	C	0.42	0.15	0.01	0.00	30.00	29.00	2.44	0.99	0.54	0.05	FALSE	15.29	0.00	2.00	TRUE	0.27
Defender	A	D	0.42	0.20	0.01	0.00	30.00	29.00	2.19	0.98	0.54	0.05	FALSE	12.10	0.00	2.00	TRUE	0.21
Defender	A	E	0.42	0.50	0.01	0.02	30.00	29.00	0.40	0.01	0.54	0.05	FALSE	-2.93	0.01	2.00	TRUE	-0.08
Defender	A	F	0.42	0.28	0.01	0.00	30.00	29.00	1.58	0.89	0.54	0.05	FALSE	7.50	0.00	2.00	TRUE	0.14
Defender	B	C	0.21	0.15	0.00	0.00	30.00	29.00	1.47	0.85	0.54	0.05	FALSE	4.11	0.00	2.00	TRUE	0.06
Defender	B	D	0.21	0.20	0.00	0.00	30.00	29.00	1.32	0.77	0.54	0.05	FALSE	0.57	0.57	2.00	FALSE	0.01
Defender	B	E	0.21	0.50	0.00	0.02	30.00	29.00	0.24	0.00	0.54	0.05	FALSE	-11.10	0.00	2.00	TRUE	-0.29
Defender	B	F	0.21	0.28	0.00	0.00	30.00	29.00	0.95	0.45	0.54	0.05	FALSE	-3.98	0.00	2.00	TRUE	-0.06
Defender	C	D	0.15	0.20	0.00	0.00	30.00	29.00	0.90	0.39	0.54	0.05	FALSE	-3.81	0.00	2.00	TRUE	-0.05
Defender	C	E	0.15	0.50	0.00	0.02	30.00	29.00	0.17	0.00	0.54	0.05	FALSE	-13.89	0.00	2.00	TRUE	-0.35
Defender	C	F	0.15	0.28	0.00	0.00	30.00	29.00	0.65	0.12	0.54	0.05	FALSE	-8.38	0.00	2.00	TRUE	-0.13
Defender	D	E	0.20	0.50	0.00	0.02	30.00	29.00	0.18	0.00	0.54	0.05	FALSE	-11.71	0.00	2.00	TRUE	-0.29
Defender	D	F	0.20	0.28	0.00	0.00	30.00	29.00	0.72	0.19	0.54	0.05	FALSE	-4.81	0.00	2.00	TRUE	-0.07
Defender	E	F	0.50	0.28	0.02	0.00	30.00	29.00	3.92	1.00	0.54	0.05	FALSE	8.53	0.00	2.00	TRUE	0.22

Fig. 15. F- and t-test values for pair-wise comparisons of best fitnesses, averaged over 30 runs, in the six scenarios, for attacker then for defender