MACTA: A MULTI-AGENT REINFORCEMENT LEARNING APPROACH FOR CACHE TIMING ATTACKS AND DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Security vulnerabilities in computer systems raise serious concerns as computers process an unprecedented amount of private and sensitive data today. Cache-timing attacks (CTA) pose an important practical threat as they can effectively breach many protection mechanisms in today's systems. However, the current detection techniques for cache timing attacks heavily rely on heuristics and expert knowledge, which can lead to brittleness and the inability to adapt to new attacks. To mitigate the CTA threat, we propose using MACTA, a multi-agent reinforcement learning (MARL) approach that leverages population-based training to train both attackers and detectors. Following the best practices, we develop a realistic simulated MARL environment, MA-AUTOCAT, which enables training and evaluation of cache-timing attackers and detectors. Our empirical results suggest that MACTA is an effective solution without any manual input from security experts. MACTA detectors can generalize to a heuristic attack not exposed in training with an 81.2% detection rate and reduce the worst-case attack bandwidth by 15% on average. In the meantime, MACTA attackers are qualitatively more effective than other attacks studied, and the average evasion rate of MACTA attackers from an unseen state-of-the-art detector can reach up to 88%. Furthermore, we found that agents equipped with a Transformer encoder can learn effective policies in situations when agents with multi-layer perceptron encoders do not in this environment, suggesting the potential of Transformer structures in CTA problems.

1 INTRODUCTION

With increasingly sensitive data and tasks, security in modern computer systems is recognized as one of the 14 grand challenges for engineering (National Academy of Engineering, 2007). As a concrete example, cache-timing attacks (CTA) in processor caches have been shown to leak private encryption keys (Yarom & Falkner, 2014; Liu et al., 2015), break existing security isolation (Kocher et al., 2019), cause privilege escalation (Lipp et al., 2018), and break new hardware security features in the latest processors (Ravichandran et al., 2022). In CTA, the attacker is able to gain access to private information (e.g., memory access patterns) from the victim who shares a cache with the attacker. Over decades, the attack and defense policies in CTA have been explored manually by computer architecture experts. To defend against such attacks, statistical analysis and machine learning models with static strategies have been proposed for CTA detection, *e.g.*, CC-Hunter (Chen & Venkataramani, 2014) uses auto-correlation and Cyclone (Harris et al., 2019) uses an SVM classifier. Yet, new CTA attacks are still being reported (Xiong & Szefer, 2020; Briongos et al., 2020; Saileshwar et al., 2021; Guo et al., 2022b;a), showing higher leakage rates or the ability to bypass existing defensive mechanisms.

Computer security can be seen as a competitive game between the attackers and the defenders, and game-theoretic approaches that analyze strategy (policies) for both sides have been proposed (Anwar et al., 2018; Elderman et al., 2017; Eghtesad et al., 2020). These methods highly abstract the attack and defense strategies, usually based on known attacks and defenses, and analyze simplified games in the limited strategy spaces. For example, Anwar et al. (2018) studies CTA-like attack scenarios where the attacker decides when to terminate its attack and the defender decides an abstract security

level. However, real-world CTA has large action and state spaces for different agents, sparse reward, and long game horizons, making the game analysis hard without exploring all possible policies.

In this work, we use multi-agent reinforcement learning (MARL) to jointly explore and optimize complex attack/defense policies in CTA. We take an integrated approach of reinforcement learning and game theory. First, we build a multi-agent gym environment, **MA-AUTOCAT**, that closely models the realistic CTA setting and allows efficient learning for both attackers and defenders. Specifically, we study a detect-and-terminate defense. Second, we propose to use a MARL approach, named **MACTA**, to automatically find both attacker and detector policies through self-play, similar to past successes in games with large state/action spaces (e.g., StarCraft (Vinyals et al., 2019), Go (Silver et al., 2016), and Poker (Brown & Sandholm, 2019)). MACTA adopts Fictitious Play (FP) (Brown, 1951), population-based training in MARL (Vinyals et al., 2019) and Proximal Policy Optimization (PPO) (Schulman et al., 2017) to learn the best response policy to a pool of diverse opponents, to avoid cyclic behaviors of the attacker/defender policies. Finally, MACTA uses a Transformer architecture to parameterize the policy/value function so that an important subset of actions can be picked up quickly during training, yielding fast policy learning.

We performed extensive experiments in a representative setting of cache-timing attack. The experiments show that learned policies trained with MACTA can *generalize* to detectors/attackers that they were not exposed to during the learning phase (henceforth referred to as"unseen detectors/attackers"). The MACTA detector exhibits an 81.2% detection rate on an existing human-designed attack without training on it, and can lower the *worst-case* number of attacks per episode (bandwidth) by 15% on average given a limit on the attacker's learning time. The MACTA attacker can bypass previously unseen detectors, Cyclone and CC-Hunter, with decent success rates, 88% and 52.7%, respectively.

While there has been increasing interest and effort in using machine learning for computer system security recently, our work is the first to introduce the hardware timing attack problem as a promising application of MARL and show that MARL can be effectively applied to detect simulated CTA attacks with strong generalization.

Our main contributions are as follows:

- We contribute a simulated multi-agent environment MA-AUTOCAT that models realistic CTA and allows learning in both cache timing attacks and defenses.
- We train agents using MACTA, and show the resultant detector acquires interesting high-level patterns that can generalize to novel attackers and make the cache less exploitable to high-bandwidth attacks.
- Our study on the neural architecture of learning agents indicates that the CTA is one case where Transformers are significantly better for retrieving state information than multi-layer perceptrons.

2 THE CACHE TIMING ATTACK CHALLENGE

2.1 DOMAIN DESCRIPTION

A cache is a small and fast on-chip memory commonly used in modern processor designs to reduce latency of memory accesses. Accessing memory addresses whose data are available in a cache is fast (called "*cache hit*"), while retrieving data from memory is much slower (called "*cache miss*").

Surprisingly, this timing difference in memory accesses due to caching could leak information across different programs/processes executing with a shared cache, a vulnerability known as *cache timing attacks* (CTA). As shown in Figure 1(a), CTA involves the attacker process and the victim process, both sharing the same cache. An example (Prime+Probe CTA (Liu et al., 2015)) is given in Figure 1(b). The victim's memory access will evict the attacker's cache line from the cache, causing latency changes in the attacker's future memory accesses. Thus, the attacker can infer whether the victim made access to any specific *memory address* by observing its own memory access latency.

2.2 PROBLEM STATEMENT

In this work, our goal is to jointly find attacker and detector policies that can be generalized to unseen opponents. This can discover unknown attacker policies, create a more robust detector, and leads

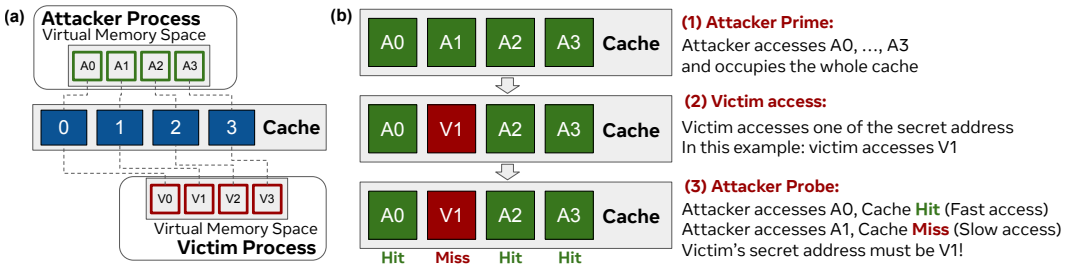

Figure 1: **(a)** Cache timing channel attack is formed when the attacker process and the victim process use the same locations of a shared cache for their memory accesses. **(b)** An example of Prime+Probe CTA in a 4-set direct-mapped cache. The attacker process can infer which memory address the victim process accesses by observing the latency.

to novel insights for future cache design. The problem of joint learning can be formulated as the general-sum Partially Observable Markov Games (POMGs), where the attacker and detector have limited observation and optimize for their own cumulative return. Given the finite set of policies, the resultant attacker is the best response to a mixture of all detector policies explored, and the resultant detector is the best response to a mixture of all attacker policies explored.

Partially Observable Markov Games (POMGs) Formally, the n-player episodic POMGs can be described using a tuple $\{\mathcal{I}, \mathcal{H}, \mathcal{S}, \mathcal{P}, \{\mathcal{A}\}_{i=1}^n, \{\mathcal{O}\}_{i=1}^n, \{\mathcal{R}\}_{i=1}^n, \gamma\}$, where $\mathcal{I}$ is the finite set of players, $\mathcal{H}$ is the episode length, $\mathcal{S}$ is the true state space, $\mathcal{P}$ is the state transition probability. $\mathcal{A}_i$ is the state space of player i, and the joint action space of all agents is $\{\mathcal{A}\}_{i=1}^n = \mathcal{A}_1 \times \mathcal{A}_2 ... \times A_n$. Similarly, $\mathcal{O}_i$ is the observation space of player i, and $\mathcal{R}_i$ is the reward function for the player i. Lastly, $\gamma \in [0, 1]$ is a reward discount factor. In POMGs, the agents can only have access to their own observation and actions, and its goal is to maximize the cumulative episodic reward for itself given the opponents' policies, $\mathcal{J}^i(\pi^i, \pi^{-i}) = \mathbb{E}\big[\sum_t^T \gamma^t r_t^i | s_0, a_t^i \sim \pi^i(s_t), a_t^{-i} \sim \pi^{-i}(s_t)\big]$. A Nash Equilibrium (NE) is an efficient solution concept to the POMGs. Formally, the NE is defined as a point where for any player's policy π^i, we have $\mathcal{J}^i(\pi_*^i, \pi_*^{-i}) \geq \mathcal{J}^i(\pi^i, \pi_*^{-i}), \forall i \in \mathcal{N}$. Namely, given all other agents' equilibrium policies π_*^{-i}, there is no motivation for agent i to unilaterally deviate from its current policy π_*^i to achieve higher returns.

3 MA-AUTOCAT

To study the learning dynamics of the attackers and the detectors in CTA, we develop MA-AUTOCAT, a gym (Brockman et al., 2016) environment that models realistic multi-agent interactions on CTA. We build the environment based on a cache simulator, as prior works on CTA detection schemes also use micro-architecture simulators (Harris et al., 2019; Mirbagher-Ajorpaz et al., 2020). Figure 2 demonstrates the environment components and game mechanism.

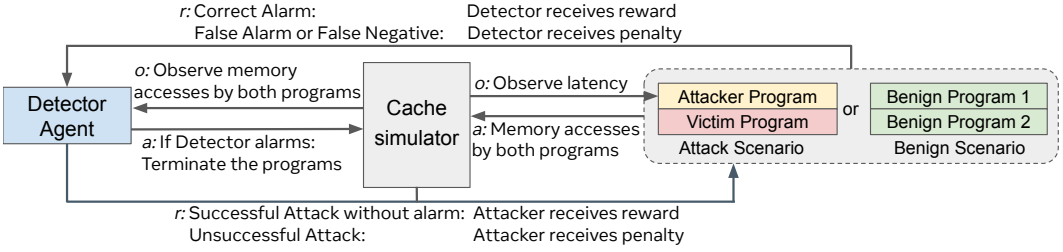

Figure 2: We propose MA-AUTOCAT, a multi-agent environment to jointly explore and optimize the policies of the attacker and the defender processes in CTA. In this environment, multiple agents can play different roles and learn from each other. The end goal is to learn policies that can generalize to deal with previously unseen opponents (e.g., those designed by human heuristics).

In MA-AUTOCAT, each agent plays a different role, and each role has a specific goal (i.e., reward), a different level of privileged accessibility (i.e., observation) to the information of the environment, and a different way to take actions (i.e., action space), listed as below:

Benign Program (B) accesses memory in a regular way, implemented by replaying an offline log of memory access pattern of regular programs (e.g., standard benchmark suite such as SPEC (Bucek et al., 2018)). It has no observation and no policy needs to be learned.

Victim (V) accesses memory with addresses that depend on the secret. Studies have shown that such secret-dependent memory accesses are common in real-world applications (e.g., HTTP parser), libraries (e.g., OpenSSL), and Linux kernel (Johannesmeyer et al., 2022; Qi et al., 2021; Oleksenko et al., 2020). In CTA, a victim's secrets usually contain multiple bits, and attackers target one bit at a time; after guessing one bit of a secret, the attacker moves to the next bit. To model this, in our environment, the secret is reset after the attacker's attempt to guess the secret and the victim accesses an address depending on the secret when triggered.

Attacker (A) aims to obtain the secret memory address of the victim process, by checking the patterns of latency of memory access. An attacker may learn a policy to *pick* which memory address to visit, and observes the binary latency signal (slow/fast). The attacker can also *trigger* the victim process to execute, and regain control after its execution, and *guess* the secret address of the victim if it is confident to do so. Importantly, the attacker can only see the latency of its own accesses.

Detector (D) aims to raise the alarm as soon as possible when an attacker is present while avoiding a false alarm for benign programs. As a system process, we assume the detector can observe memory accesses of all running processes in the environment. The detector will *terminate* the episode if an alarm is raised.

See Appendix A.2 for detailed specifications of the observation, action, and rewards.

In each episode, we may pick multiple agents of different roles to be in the environment and let them interact. In this work, we mainly test the following two possible scenarios:

- Attack Scenario (**DAV**). The environment contains a detector, an attacker and a victim. The attacker aims to obtain the secret address of the victim and the detector aims to identify the situation and terminate processes as soon as possible.
- Benign Scenario (**DBB**). The environment contains a detector and two benign programs with no malicious intent. In this case, the detector should not cause any false alarms.

We leave more complicated settings, such as scenarios with both victims and benign programs (e.g., DAVB), as future work.

4 METHOD

The CTA we consider is a POMG with three fundamental characteristics: **1) Partial Observability**. In CTA, the attacker knows which program to attack but can only see the attacker's own actions and latencies, while the detector does not know if there is any attacker nor which program the attacker is targeting. **2) Sparse-Reward Markov Game**. The CTA game can have a long episode length, and agents have to come up with a good action sequence before receiving the reward. Especially the attacker must learn both low-level *skills* to perform attacks and high-level *strategies* to avoid defenders. **3) Environment Randomness**. Such randomness comes from randomized victim secret addresses and the random trajectory sampling of benign programs. We propose our method based on the three crucial features.

4.1 OUR APPROACH: MACTA

In this paper, we introduce our approach, MACTA (Figure 3 Right), as an initial solution to the CTA challenge using MARL. MACTA adopts Transformers (Vaswani et al., 2017) as the neural encoder of policy nets, Proximal Policy Optimization (PPO) (Schulman et al., 2017) as the reinforcement learning oracle, and Fictitious Play (FP) (Brown, 1951) as the meta game-theoretic tool.

To deal with the history-dependent partial observation and sparse reward, both the attacker and the detector are equipped with policy nets with transformer encoders. The transformer encoder is mainly composed of scaled dot-product attention and multi-head self-attentions. It can effectively integrate information from long time horizons and large-scale data while not suffering from vanishing or exploding gradients in recurrent neural networks (RNNs) (Parisotto et al., 2020).

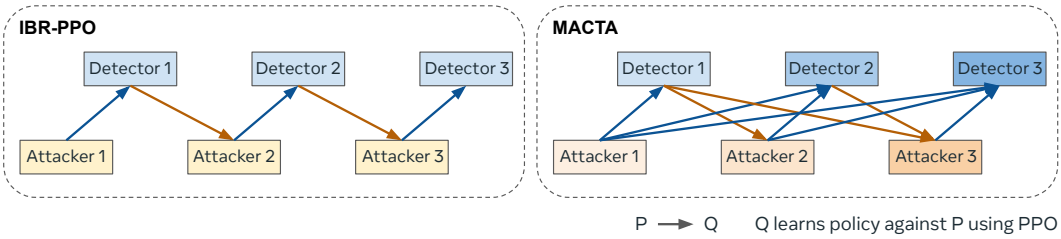

Figure 3: Method. Iterative Best Response PPO (IBR-PPO) learns the best response to the previous opponent only, while MACTA learns the best response to a uniform mixture of all historical opponents.

The attacker and the detector optimize their policies by the PPO algorithm to effectively learn a policy in the Markov game. Although independent reinforcement learning, where all agents are updating their policies simultaneously, is notoriously known for the instability issue in training Tan (1993), if we only train one agent at a time and keep others stationary, then other agents can be taken as a part of the environment, and PPO can effectively optimize the policy for higher cumulative rewards. Iterated Best Response PPO (IBR-PPO) (Figure 3 Left) is the most naive way of implementing the above idea. It alternates the training of the attacker and detector so that they learn the best policy against the most recent opponent. However, it may fall into the cyclic policy learning issue and never converge to any Nash Equilibrium (Roughgarden, 2010).

As a widely accepted method in MARL, creating a diverse pool of opponents and learning the best response to a mixture of them can alleviate the cyclic issue and help with generalization. Similar to fictitious play in the game theory, we create a pool for each agent and add their historical policies to the pool. Concretely, for each iteration τ, we denote the set of policies explored until τ of agent i by our method as Π_τ^i, the opponents' joint policy set as Π_τ^{-i}. Then we learn the best response (BR), $\pi_*^i(\mathbb{U}(\Pi_\tau^{-i}))$, to the uniform mixture of opponents' policy pool using a *best response oracle* (e.g. PPO), and add the best response to the policy pool. Mathematically, for each iteration

$$\forall i : \Pi_{\tau+1}^i \leftarrow \Pi_\tau^i \cup \{\pi_*^i(\mathbb{U}(\Pi_\tau^{-i}))\}$$

where $-i$ represents all players except for player i, and $\mathbb{U}$ is the uniform distribution. There are more advanced meta game frameworks like Double Oracle (McMahan et al., 2003) and Policy Space Response Oracle (Lanctot et al., 2017), which measures the meta game payoff matrix among different explored policies and solves the matrix for the best opponent mixture. In our case, since the environment contains some randomness, it is inefficient to estimate precisely the payoff matrix. We will leave the more advanced game framework as future work.

The above components constitute our approach (Algorithm 1). MACTA alternates the training of attacker and detector every E epochs and adds one *deterministic* policy checkpoint of the learning agent to the agent's policy pool every N epochs. During one agent's training, the agent faces a *uniform* mixture of all opponents' past deterministic policy checkpoints. Note that we create such a mixture by uniformly sampling policies from the opponent's policy pool at every action step.

4.2 Implementation Details

Specifically, we start with empty policy pools for both agents, first train the attacker for 50 epochs (each epoch contains 3000 training steps) to gain the basic skills of obtaining information from the victim program, and add one policy to the attacker's policy pool every 10 epochs. Then we stop the attacker's training and switch to train the detector against the pool of the first 5 attacker policies for 50 epochs. Similarly, the detector will have 5 policies by the end of this training iteration (50 epochs). The above process is repeated until the target training iterations (2000 epochs). We adopt an Actor-Critic implementation of PPO for both the attacker and the detector, and both policy net and value net are 1-layer 8-head Transformer encoders with different output heads. We leverage the RLMeta (Yang et al., 2022) learning framework for the PPO implementation, which is an asynchronous version of PPO with sampling and learning in parallel, and construct our multi-agent learning framework on top of it. Refer to Appendix A.6 for a more detailed description of training and environment hyper-parameters.

5 EXPERIMENTS

5.1 EVALUATION SETUP AND METRICS

To evaluate the proposed MARL method, we study a few attackers and detector baselines. For the attackers, we consider a textbook attack Prime+Probe (Algorithm 2), the RL-based attacker (AutoCAT) (Luo et al., 2022), and PPO with Iterative Best Response Oracle (IBR-PPO Attacker). For the detectors, we include CC-Hunter (Chen & Venkataramani, 2014) (Appendix A.7), Cyclone (Harris et al., 2019) (Appendix A.7), and PPO with Iterative Best Response Oracle (IBR-PPO Detector).

The episodic cumulative reward is one way to measure the performance of the attacker and the detector, but it is less intuitive. In this work, we employ more intuitive metrics including ***Attack Correct Rate***, ***Attacks per Episode***, ***Detection Rate***, ***Episode Length***, and ***False Alarm Rate***. Details are listed in Table 1.

Table 1: Evaluation metrics.

Metrics	Object	Description
Attack Correct Rate	Attacker	Measures the ability of an attacker to infer a secret correctly (attack successfully). It is the percentage of correct guess among the all guesses aggregated over episodes.
Attacks per Episode	Attacker, Detector	Measure the speed of an attacker or the attacker's ability to bypass detection or the detector's ability to prevent attacks. It is the average number of correct guesses per episode.
Detection Rate	Attacker, Detector	Given the attacker and detector, detection rate is the percentage of attacks flagged by the detector within the time limit in the evaluate episodes.
Episode Length	Attacker, Detector	Given attacker and detector, it measures how fast the detector can find out the existence of the attacker.
False Alarm Rate	Detector	Measures the false positive (terminate episode before time limit) rate of a detector given all benign agents.

5.2 RESULTS

All the experiment results below are reported on an 8set-1way L1 cache. The attacker's memory address range is 8-15 and the victim's secret address is randomly chosen between 0-7. The episode length is 64 steps. To evaluate different methods, we report the statistics based on three independent training instances for each learning-based method and control the final policies from different instances of a method undergoing the same amount of optimization steps.

5.2.1 ATTACKER PERFORMANCE

Table 2: Attacker performance. Evaluation of the attacker's correct rate and number of attacks in 64-step episodes without detectors.

Metrics \ Attackers	Prime+Probe	AutoCAT	IBR-PPO Attacker	MACTA Attacker
Attack Correct Rate ↑	100.0 ± 0.0	100.0 ± 0.0	99.5 ± 0.3	99.8 ± 0.3
Attacks per Episode ↑	3.0 ± 0	5.2 ± 0.1	4.9 ± 0.1	4.4 ± 0.3

We first evaluate the attacker agent's performance in terms of attack correct rate and number of attacks in an episode, to validate that the attacker agent is conducting effective attacks. We compare the PPO attacker agents (IBR-PPO and MACTA) trained by us with existing attacker strategies (textbook Prime+Probe and AutoCAT). We evaluate the attackers without detectors for 10,000 episodes and the statistics are shown in Table 2. Every attacker evaluated can achieve a decent attack correct rate, showing the agent acquires effective attack policies. In addition, MACTA has the smallest number of attacks per episode among the learning-based methods, indicating the preventative strategies for detection. Example attack sequences demonstrating the strategic attack behaviors can be found in Section A.3.

5.2.2 HEAD-TO-HEAD EVALUATIONS

In this head-to-head evaluation, we have an attacker play against a detector from *different* training instances for 10,000 episodes and report the mean detection rate and the mean episode length for all attacker and detector pairs. The head-to-head evaluation results can be found in Table 3 and Table 4. We also report the mean false alarm rate and mean episode length of the detectors on the *unseen* Benign agents in the last column of the table.

Table 3: Mean detection rate. Head-to-head evaluations with unseen opponents from different training instances. The higher the better for detectors when the opponent is an attacker, the lower the better when the opponents are benign programs. '-' as Cyclone is trained on Prime+Probe.

Detectors \ Opponents	Prime+Probe ↑	AutoCAT ↑	IBR-PPO Attacker ↑	MACTA Attacker ↑	Benign ↓
CC-Hunter	67.6 ± 1.3	50.1 ± 2.0	45.0 ± 1.6	**43.3** ± 5.4	45.2 ± 4.6
Cyclone	-	0.0 ± 0.0	6.6 ± 3.1	12.0 ± 4.1	3.5 ± 0.6
IBR-PPO Detector	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	**0.8** ± 0.6
MACTA Detector	**81.2** ± 18.6	**77.6** ± 20.4	**71.7** ± 16.1	13.5 ± 14.7	2.1 ± 1.2

Table 4: Mean episode length. Head-to-head evaluations with unseen opponents from different training instances. The lower the better for detectors when the opponent is an attacker, the higher the better when the opponents are benign programs. Cyclone and CC-Hunter both require a fixed episode length of 64 steps.

Detectors \ Opponents	Prime+Probe ↓	AutoCAT ↓	IBR-PPO Attacker ↓	MACTA Attacker ↓	Benign ↑
IBR-PPO Detector	64.0 ± 0.0	64.0 ± 0.0	64.0 ± 0.0	64.0 ± 0.0	**63.6** ± 0.3
MACTA Detector	**33.4** ± 18.4	**33.7** ± 12.6	**33.5** ± 11.8	**58.4** ± 7.5	62.8 ± 0.7

We find that the heuristic detector CC-Hunter can not effectively discriminate the RL attackers from benign agents. Tuning the auto-correlation threshold only returns either a high false alarm rate or a low detection rate. The SVM-based detector, Cyclone, is able to perform well (99.3% detection rate) on the heuristic attack (Prime+Probe) that it is trained on, but has low detection rate on RL attackers. Another drawback of these methods is that they require fixed-length observation that is longer than the steps needed for RL to complete attacks (usually 12 steps in this cache configuration). IBR-PPO obviously falls into the cyclic policy learning issue, as the detector is able to react well (98.3% detection rate) to the attacker that it trained against but fails to respond to any other kind of attackers.

MACTA, however, is able to generalize to unseen attacks such as Prime+Probe and the IBR-PPO attacker. At the same time, MACTA also has a low false positive rate and fast detection speed. We hypothesize that MACTA can abstract the general pattern of the attackers from interacting with diverse attacker strategies during training.

On the other hand, since the detector is trained to block all the previous attack policies, the attacker had to explore new policy space to escape from the detection. The MACTA attackers are able to escape from a variety of unseen detectors. All of the above findings highlight the benefits of using MARL solution concepts in learning the detectors.

5.2.3 EXPLOITABILITY EVALUATIONS

We measure how a detector can be exploited by attackers within one fictitious play iteration (50 training epochs). We fix a detector strategy and train an RL exploiter (i.e., an attacker) against the detector from scratch. The training curve of the exploiters of MACTA detectors can be found in Figure 4. As the training time of the MACTA detectors increases, it becomes more difficult for an RL attacker to bypass the detectors. Specifically, it will take the RL exploiter attacker longer to find a meaningful attack strategy. And even though the RL exploiter attacker can learn to attack eventually, the number of total attacks decreases when the detector sees more attackers. The decrease in the number of attacks can come from the slower attack speed (counter to possible detection) or faster detection speed so fewer attacks can be performed.

5.3 NEURAL ARCHITECTURE STUDY

The neural architecture design for the policy networks has been relatively under-explored in the RL community. Our CTA task is an example where neural architecture plays a critical role in learning

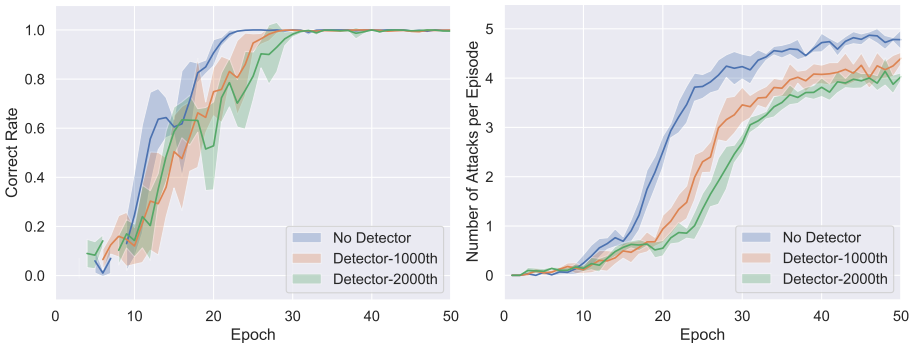

Figure 4: Exploitability evaluation. We fix the detector policies and train an RL attacker against the detectors from scratch. **Left:** Attacker Correct Rate. **Right:** Attacker's Number of Attacks per episode.

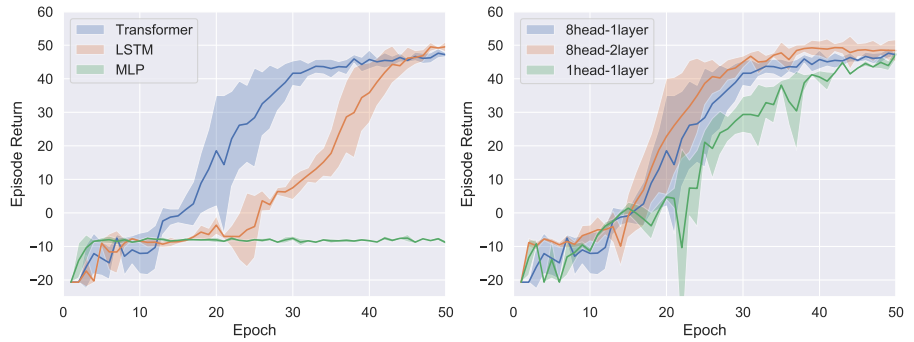

Figure 5: A study on different neural architectures. We use a Transformer with 8-head attention and one Transformer encoder layer in MACTA experiments. **Left:** Train attacker-only task using different neural architectures. **Right:** Train attackers with different Transformer configurations.

a meaningful policy. We train attacker-only tasks using different network architectures (details in Appendix A.5), and the comparisons can be found in Figure 5. For RL attackers, MLP with residual connection (He et al., 2016) fails to achieve a high episode return, while the Transformer and LSTM (Hochreiter & Schmidhuber, 1997) entries succeed. For Transformers, our study shows that increasing the number of encoder layers in the Transformer can speed up learning while reducing the number of heads has counter effects. The above evidence shows that the sequence modeling structure is critical for CTA attack policy learning and Transformer is more efficient than LSTM in terms of training steps. Our hypothesis is that a successful attack is composed of a series of events, which may contain history-dependent relations among events, and Transformer can effectively model such relations.

6 Related Work

Detectors for Cache Timing Attacks Since the discovery of side-channel attacks, research efforts have also been made to build a detector against attacks. CC-Hunter (Chen & Venkataramani, 2014) captures recurrent patterns generated during cache contention while adversaries exploit the victim process. More specifically, it used autocorrelation to detect periodic interleaving between the two event trains (Luo et al., 2022). ReplayConfusion (Yan et al., 2016) records and deterministically replays a program's memory traces, changing the mapping of cache addresses but retaining the cadences. Executing the traces in different memory addresses can expose abnormal access patterns observed between an attacker and a victim, which do not exist in benign traces. Cyclone (Harris et al., 2019) uses cyclic interference identified in cache contention during an attack. This detector assigns domain tags between the two processes, then uses performance counters to enumerate abnormal cache contention behaviors triggered by each domain tag. PerSpectron (Mirbagher-Ajorpaz et al., 2020) trains a neural network classifier using the memory and latency event logs generated from attack examples. Existing detectors based on known attacks cannot deal with the evolving attacker issue. Our study shows that the RL attacker can learn novel strategies to bypass the existing static detectors. Our method solves this problem by enabling the auto-discovery of attacker policies.

Game Theory in Security Games Game theory provides a framework for decision-making and strategy, modeling how selfish agents interact and affect system outcomes. In Stackelberg games, a defender must first commit limited resources to protect disparate locations and an attacker that subsequently targets locations, potentially having seen the configuration of defenses (e.g., (Bier et al., 2007)). Such games have masked systems from probes (Schlenker et al., 2018), defended systems against varied attack types (Thakoor et al., 2020), and assigned human analysts to automated system alerts (Schlenker et al., 2017). Whereas Stackelberg requires the defender to move first, we consider how the defender's policy should respond to the attacker's evolving policy. Game theory inspires generative adversarial networks (GAN) for security (Zolbayar et al., 2021; Baimukan & Zhu, 2021). Unlike prior works that explore adversarial samples in the neighborhood of a heuristic attack policy, our RL approach explores a broader, unknown space of attack policies with a well-defined objective. RL is an instance of stochastic games, often modeled by a Markov Decision Process. Representative studies of such games for distributed systems include threat detection and resource allocation (Krishnamurthy et al., 2007; Fan et al., 2019). We are the first to formulate a stochastic game for realistic, practical hardware timing attacks.

Population-based Multi-agent Reinforcement Learning Independent Reinforcement Learning in multi-agent environments suffers from the non-stationary opponent issue (Tan, 1993). While Iterative Best Response methods alleviate the above problem by learning from stationary opponents, they tend to over-fit other players' policies and cause cycles in policy learning (Vinyals et al., 2019). Interacting with diverse opponent policies or heterogeneous agents is one effective way to avoid such cycles. Population-based MARL is thus proposed to solve large-scale extensive form games by creating a diverse pool of agents. Related work includes population-based reinforcement learning (Parker-Holder et al., 2020), Neural Fictitious Self-Play (Heinrich & Silver, 2016), Fictitious Co-Play (Strouse et al., 2021), prioritized self-play (Vinyals et al., 2019), Double Oracle (DO) (McMahan et al., 2003) and its generalization Policy Space Response Oracle (PSRO) (Lanctot et al., 2017) and so on. The most related application of population-based MARL to the security games, such as Eghtesad et al. (2020) and Wang et al. (2019), use variants of Double Oracle, but they deal with different and less stochastic domains than ours.

7 Conclusions and Future Work

In conclusion, our work explores the application of multi-agent reinforcement learning in the cache timing attack and detection domain. We first introduce the environment MA-AutoCAT that allows learning for both attackers and detectors, and their complex interactions with caches. Then we propose to combine the game-theoretic concept of Fictitious Play and Proximal Policy Optimization to train both agents (MACTA). Empirically, we found that the detector generated by MACTA can capture the general pattern of attacks and generalize to novel attacks. The exploitability study of the detector also indicates the detectors can impede the learning process of new attackers and slow down the attacks. On the other hand, the MACTA attacker is able to explore new policy space and mimic the benign agents to bypass the detectors. Finally, the neural architecture study demonstrates the strong representation-ability of Transformers.

Deployment in Real Systems Like prior works (Luo et al., 2022; Harris et al., 2019; Mirbagher-Ajorpaz et al., 2020), we use a cache simulator to study CTA. We believe the trained attacker and detector can be applied to real hardware with engineering efforts. For attackers, a recent study (Luo et al., 2022) demonstrated that an attack pattern learned by the RL agent using a cache simulator can be applied to multiple Intel processors. Similarly, we also show that the attack sequences from a MACTA attack work on commercial processors in Appendix A.4. For the detectors, with hardware changes, the neural network model can be deployed inside a processor with reasonable area and power overhead, as demonstrated by Mirbagher-Ajorpaz et al. (2020).

Convergence of the Policies Three main concerns for reinforcement learning algorithms are deep learning, bootstrapping, and off-policy. In MACTA, we adopt the Transformer based PPO algorithm as the policy learning oracle, so there is no guarantee that the algorithm will return the best response to the opponents in limited optimization steps in such a complex and stochastic environment. Meanwhile, although Fictitious Play can generate a NE policy in two-player zero-sum games, it has been proven to result in cyclic policies in a specific case (Shapley, 1964). Little previous work discusses its convergence when used as a game-theoretic meta solver in the general-sum MARL setting. In this work, we only establish empirical findings but leave the theoretical analysis for future work.

8 OPTIONAL STATEMENTS

Dear reviewers, please note that this section is optional and does not count into the 9-page limit.

Ethical Statement Our work builds an environment enabling automated exploration of both attack and defense policies of CTA. The attacker agent may be used for developing new attacks maliciously. However, as shown in the vast amount of attack papers in computer security venues, exploring and understanding security attacks is a necessary and important step in developing future secure systems. Compared with prior works on security attacks on real systems Liu et al. (2015); Oren et al. (2015); Kocher et al. (2019); Lipp et al. (2018); Ravichandran et al. (2022), the MACTA agent only explores the attacks in a cache without considering other system-level activities. In addition, our framework results in a stronger detector, which can help design more secure systems.

Reproducibility Statement Our experiments are reproducible in a reasonable range near the mean performance across multiple training instances reported. It is unavoidable that training with different random seeds will result in different results, given the variance of explorations in reinforcement learning. However, our result is not a selection among multiple random seeds biased towards our benefits. We repeated the training for another three instances, giving us a similar result. We will provide model checkpoints and publish the code upon the publication of this paper.

REFERENCES

Ahmed H Anwar, George Atia, and Mina Guirguis. Toward a protected cloud against side channel attacks: A game-theoretic framework. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 78–83. IEEE, 2018.

Nurpeiis Baimukan and Quanyan Zhu. Concealment charm (concealgan): Automatic generation of steganographic text using generative models to bypass censorship. *Game Theory and Machine Learning for Cyber Security*, pp. 357–365, 2021.

Vicki Bier, Santiago Oliveros, and Larry Samuelson. Choosing what to protect: Strategic defensive allocation against an unknown attacker. *Journal of Public Economic Theory*, 9(4):563–587, 2007.

Samira Briongos, Pedro Malagón, José M Moya, and Thomas Eisenbarth. RELOAD+ REFRESH: Abusing cache replacement policies to perform stealthy cache attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1967–1984, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1): 374, 1951.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 41–42, 2018.

Jie Chen and Guru Venkataramani. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 216–228. IEEE, 2014.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Taha Eghtesad, Yevgeniy Vorobeychik, and Aron Laszka. Adversarial deep reinforcement learning based adaptive moving target defense. In *International Conference on Decision and Game Theory for Security*, pp. 58–79. Springer, 2020.

Richard Elderman, Leon JJ Pater, Albert S Thie, Madalina M Drugan, and Marco A Wiering. Adversarial reinforcement learning in a cyber security simulation. In *ICAART (2)*, pp. 559–566, 2017.

Songchun Fan, Seyed Majid Zahedi, and Benjamin C Lee. Distributed strategies for computational sprints. *Communications of the ACM*, 62(2):98–106, 2019.

Yanan Guo, Xin Xin, Youtao Zhang, and Jun Yang. Leaky way: a conflict-based cache covert channel bypassing set associativity. In *2022 IEEE International Symposium on Microarchitecture (MICRO)*, pp. 1458–1473. IEEE, 2022a.

Yanan Guo, Andrew Zigerelli, Youtao Zhang, and Jun Yang. Adversarial prefetch: New cross-core cache side channel attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1458–1473. IEEE, 2022b.

Austin Harris, Shijia Wei, Prateek Sahu, Pranav Kumar, Todd Austin, and Mohit Tiwari. Cyclone: Detecting contention-based cache information leaks through cyclic interference. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 57–72, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Brian Johannesmeyer, Jakob Koschel, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Kasper: Scanning for generalized transient execution gadgets in the linux kernel. In *NDSS Symposium 2022*, 2022.

Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19. IEEE, 2019.

Vikram Krishnamurthy, Michael Maskery, and Minh Hanh Ngo. Game theoretic activation and transmission scheduling in unattended ground sensor networks: A correlated equilibrium approach. *Wireless Sensor Networks: Signal Processing and Communications Perspectives*, pp. 349–388, 2007.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 973–990, 2018.

Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pp. 605–622. IEEE, 2015.

Mulong Luo, Wenjie Xiong, Geunbae Lee, Yueying Li, Xiaomeng Yang, Amy Zhang, Yuandong Tian, Hsien Hsin S Lee, and G Edward Suh. Autocat: Reinforcement learning for automated exploration of cache timing-channel attacks. *arXiv preprint arXiv:2208.08025*, 2022.

H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 536–543, 2003.

Samira Mirbagher-Ajorpaz, Gilles Pokam, Esmaeil Mohammadian-Koruyeh, Elba Garza, Nael Abu-Ghazaleh, and Daniel A Jiménez. Perspectron: Detecting invariant footprints of microarchitectural attacks with perceptron. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1124–1137. IEEE, 2020.

National Academy of Engineering. 14 grand challenges for engineering of the 21st century. 2007. URL `http://www.engineeringchallenges.org/challenges/cyberspace.aspx`.

Oleksii Oleksenko, Bohdan Trach, Mark Silberstein, and Christof Fetzer. SpecFuzz: Bringing spectre-type vulnerabilities to the surface. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1481–1498, 2020.

Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1406–1418, 2015.

Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *Cryptographers' track at the RSA conference*, pp. 1–20. Springer, 2006.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Jack Parker-Holder, Aldo Pacchiano, Krzysztof M Choromanski, and Stephen J Roberts. Effective diversity in population based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18050–18062, 2020.

Zhenxiao Qi, Qian Feng, Yueqiang Cheng, Mengjia Yan, Peng Li, Heng Yin, and Tao Wei. Spectaint: Speculative taint analysis for discovering spectre gadgets. In *NDSS*, 2021.

Joseph Ravichandran, Weon Taek Na, Jay Lang, and Mengjia Yan. Pacman: attacking arm pointer authentication with speculative execution. In *ISCA*, pp. 685–698, 2022.

Tim Roughgarden. Algorithmic game theory. *Communications of the ACM*, 53(7):78–86, 2010.

Gururaj Saileshwar, Christopher W Fletcher, and Moinuddin Qureshi. Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1077–1090, 2021.

Aaron Schlenker, Haifeng Xu, Mina Guirguis, Christopher Kiekintveld, Arunesh Sinha, Milind Tambe, Solomon Sonya, Darryl Balderas, and Noah Dunstatter. Don't bury your head in warnings: A game-theoretic approach for intelligent allocation of cyber-security alerts. 2017.

Aaron Schlenker et al. Deceiving cyber adversaries: A game theoretic approach. In *International Conference on Autonomous Agents and Multi Agent Systems*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Lloyd Shapley. Some topics in two-person games. *Advances in game theory*, 52:1–29, 1964.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, January 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34: 14502–14515, 2021.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Omkar Thakoor, Phebe Vayanos, Milind Tambe, and Minlan Yu. Game theory for strategic ddos mitigation. 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Pepe Vila, Pierre Ganty, Marco Guarnieri, and Boris Köpf. Cachequery: Learning replacement policies from hardware caches. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 519–532, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1401–1408, 2019.

Wenjie Xiong and Jakub Szefer. Leaking information through cache LRU states. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 139–152. IEEE, 2020.

Mengjia Yan, Yasser Shalabi, and Josep Torrellas. ReplayConfusion: detecting cache-based covert channel attacks using record and replay. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–14. IEEE, 2016.

Xiaomeng Yang, Brandon Cui, Teng Li, and Yuandong Tian. RLMeta: A Flexible Framework for Distributed Reinforcement Learning, 1 2022. URL `https://github.com/facebookresearch/rlmeta`.

Yuval Yarom and Katrina Falkner. FLUSH+ RELOAD: A high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX security symposium (USENIX security 14)*, pp. 719–732, 2014.

Bolor-Erdene Zolbayar, Ryan Sheatsley, Patrick McDaniel, and Mike Weisman. Evading machine learning based network intrusion detection systems with gans. *Game Theory and Machine Learning for Cyber Security*, pp. 335–356, 2021.

A APPENDIX

A.1 WHY STUDY CACHE TIMING ATTACKS

CTA are stealthy but powerful. They do not violate any access control policies enforced by the operating system and low-level hardware and they are shown to pose serious security concerns in practice. For example, some implementations of security critical software such as encryption algorithms have a secret dependent access pattern, and an attacker can use CTA to obtain secret keys (Osvik et al., 2006; Liu et al., 2015). CTA also enables covert communication channels between two domains and breaks the existing security isolation mechanism, e.g., sandbox in javascript (Oren et al., 2015), isolation between processes (Kocher et al., 2019), and the system privilege levels (Lipp et al., 2018). CTA can also facilitate brute forcing hash values stealthily without triggering exceptions, which is shown to help break the ARM pointer protection mechanisms (Ravichandran et al., 2022). One of the important defensive strategies is to detect unique characteristics of memory access patterns of attacker programs that are different from usual benign ones, as leveraged by the state-of-the-art cache-timing channel detectors (Yan et al., 2016; Chen & Venkataramani, 2014; Harris et al., 2019; Mirbagher-Ajorpaz et al., 2020). However, many new attacks (Briongos et al., 2020; Luo et al., 2022) avoid the characteristics that the detector uses and it is hard to adapt existing detectors to previously unseen attacks or access patterns.

A.2 ENVIRONMENT CONFIGURATIONS

Table 5: Environment hyper-parameters.

Parameter Group	Parameter Name	Parameter Value
MA-AUTOCAT	Max Episode Length	64 steps
MA-AUTOCAT	Observation Window Size for the attacker and the detector	64 steps
MA-AUTOCAT	Probability between Attack Scenario and Benign Scenario during Training	[50%, 50%]
MA-AUTOCAT	Benign Program Logs (Train)	1 Million Steps
MA-AUTOCAT	Benign Program Logs (Validation)	1 Million Steps
MA-AUTOCAT	Attacker Memory Address Range	8-15
MA-AUTOCAT	Victim Memory Address Range	0-7
Cache Simulator	Cache Configuration	L1 Cache, 8 set 1 way
Cache Simulator	Replacement Policy	Least Recently Used (LRU)

Game Mechanism. In the MA-AUTOCAT, within a fixed-length episode, the attacker agent can guess the secret address of the victim as many times as possible and get a reward for every correct guess (*successful attack*). In the meantime, the detector agent can monitor the cache access history and interactions of two programs and decide whether to raise a flag/alarm to terminate the episode to prevent further information leakage.

Attacker's Reward Function. The attacker is punished by 0.01 for every time step, +10 if guess the victim's secret successfully, -10 if incorrectly. It will get a one-time punishment of 20 if it reaches a timeout without any attack, a one-time punishment of -10 if identified by detector. The episode length to collect reward is affected by the detector.

Detector's Reward Function. The detector can raise a flag to terminate the episode. If the detector raises a flag in an attack scenario, then the detector receives a reward for remaining steps [max step - current step]; if the detector raises a flag in a benign scenario, then it receives a large penalty [5 $\times$ max step]. If the detector lets the episode going, for benign scenario there is no punishment; while the detector gets -10 every time the attacker attacks successfully.

Attacker's Action. For each time step, the attacker can choose an action $a^a \in \{a^a_X, a^a_v, a^a_{vr}, a^a_{gY}\}$, where a^a_X represents access address X, a^a_v represents letting the victim access a secret-related address, a^a_{vr} represents letting the victim access a random address and a^a_{gY} represents guessing the secret address to be Y.

Detector's Action. For each time step, the defender can choose $a^d \in \{a^d_{term}, a^d_{cont}\}$ where a_{term} means terminate the episode and a^d_{cont} means let the episode keep going.

Attacker's Observation. The attacker's observation space includes a history of attacker actions and memory access latency it receives from the cache simulator. For each time step, a new step observation $(s^a_{lat}, s^a_{vt}, s^a_{act}, s^a_{step})$ is appended to the observation window, where $s^a_{lat} \in \{s_{hit}, s_{miss}, s_{N.A.}\}$ represents the access latency, $s^a_{vt} \in \{s_t, s_{nt}\}$ represents whether to wait for the victim's action, s_{act} records the attacker's current action and s^a_{step} is the current time step.

Detector's Observation. The detector can observe a history window of the memory access actions of both programs. For each time step, the new observation is composed of $(s^d_{lat}, s^d_{program}, s^d_{addr}, s^d_{step})$, where $s^d_{lat} \in \{s_{hit}, s_{miss}\}$ represents the access latency (access latency of all programs are visible to the detector), $s^d_{program} \in \{s_a, s_b\}$ indicates the identity of the program, s^d_{addr} represents the address being accessed at the current step, and s^d_{step} represents the current defender time step.

Benign Programs. The actions of benign programs are sampled from the Standard Performance Evaluation Corporation (SPEC) 2017 benchmark Bucek et al. (2018), a collection of memory access records of non-malicious programs. Since the data is collected under different cache configurations, we project the actions onto the valid action space given the current cache configuration.

A.3 Trajectory Analysis

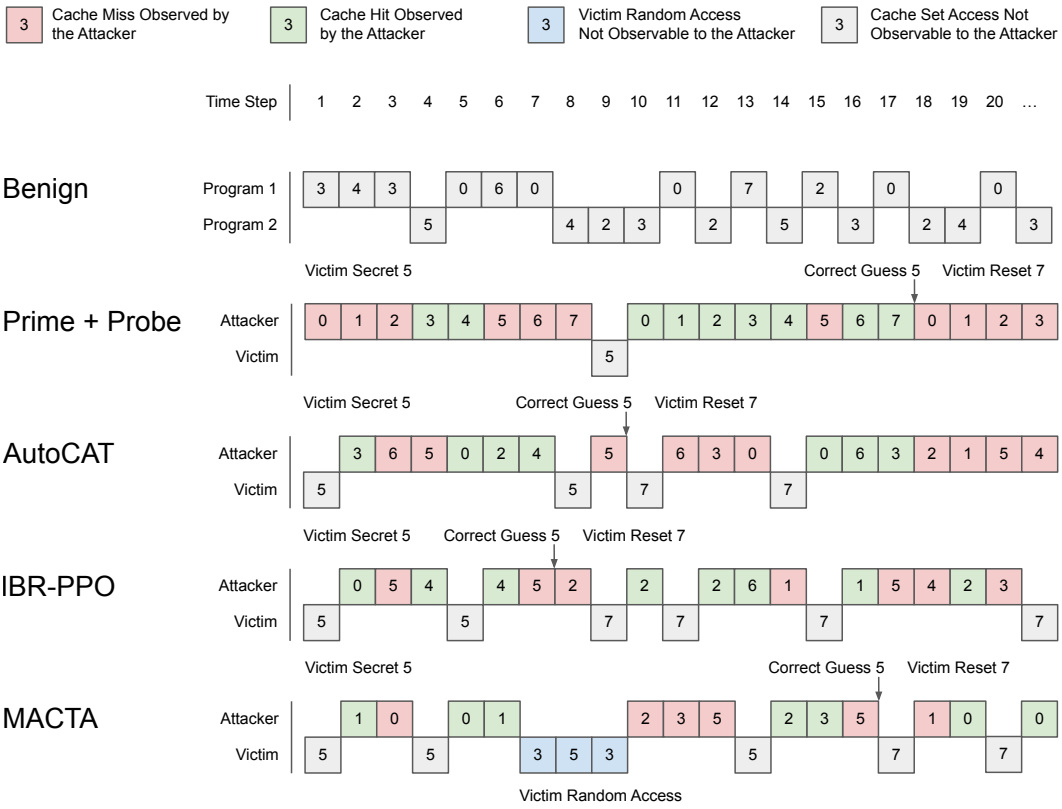

Figure 6: Example trajectories of different attackers and benign agents in a 8-set 1-way L1 cache. The number indicates the cache set being accessed. Red and green boxes show the observation by the attacker. The latency of other programs (i.e., victim or benign) cannot be observed by the attacker, but they can be observed by the detectors. The program IDs are randomized during training, and the attacker can be any of the two programs in the system. The cache is initialized with random states.

Figure 6 illustrates different attackers' attack sequence given a fixed secret bit. Here, we use victim secret=5 as an example. The top row shows a sampled pattern of benign programs. In that case, the two programs act independently and alter the access to the cache frequently. The Prime+Probe attacker cause cache contention by accessing cache frequently, and only invoke the victim to access

the cache when needed. The AutoCAT attacker is more efficient in retrieving victim's information then Prime+Probe. Once contention with victim in one cache set is observed (i.e., a cache miss after the same address being accessed by the attacker), the attacker will make a guess without more memory accesses. The IBR-PPO attacker takes a similar strategy as AutoCAT's, but it learns to insert some extra victim invocation steps to confuse the detector. The issue with extra invocation strategy is that the victim only accesses its secret bit, which can be easily captured by a detector. The MACTA attacker however, learns more advanced strategies. It learns to invoke random victim accesses to alter its behavior to be more similar to benign programs. Note that invoking random accesses from a victim can cause noise in the attacker's latency observation and make the steps needed for a success attack longer. This means that to evade the MACTA detector, just inserting some extra victim invocation steps is far from enough. The attacker has to take a risk to invoke random accesses from a victim instead because the "easiest" policy space has been exhausted.

A.4 Real Hardware Analysis

Table 6: Attack evaluation on commercial processors. We report the attack correct rates of MACTA attack sequences on three commercial Intel processors for 10,000 episodes. MACTA attackers achieve a 99.9% correct rate in the simulator, and still $> 99\%$ on real hardware.

CPU	Cache Level	#Ways	Attack Correct Rate ↑
i7-6700 (SkyLake)	L1	8	99.73%
i7-7700k (KabyLake)	L3	8 (partitioned from 16 way)	99.32%
i7-9700 (CoffeeLake)	L1	8	99.98%

We evaluated the effectiveness of the attack sequences produced by MACTA attackers on real hardware by running them on three commercial processors through CacheQuery (Vila et al., 2020). The attack sequences are generated from a MACTA attacker that is trained using a simulated environment for a 1-set 8-way cache configuration, which match the number of ways in real hardware. In this setting, the victim's secret is either 0 or E (Empty) with equal probability, and the attacker can choose to invoke victim to access the secret address or a random address. Then, the attack sequences were performed on real hardware to obtain the attack correct rate. Table 6 demonstrates the attacker policy from simulator can be transferred to real hardware with negligible discrepancy. Even though the caches in commercial processors have a large number of sets, the attacks from the MACTA attacker transfers well because each set operates independently.

A.5 Model Hyper-parameters

In section 5.3, we compared three different neural architectures: Transformers, LSTM and MLP. All of the methods are controlled at a same scale of optimizable parameters, and trained with the same algorithms and computing resources.

Transformer. In our experiments, all the policies uses a 8-head 1-encoder-layer transformer with hidden dimension of 128. For model architecture study, we study the changes in the number of heads in the multi-head attention mechanism, and number of transformer layers.

LSTM. We employed a 1-layer LSTM with hidden dimension 256. The input to the LSTM is a pre-padding history of the observation and we concatenate the last layer hidden and cell before connecting with a MLP.

MLP. The MLP model we used directly maps the one-hot encoded input to 4 residual blocks with hidden dimension 128. Each residual block is composed of 2 ReLU-Linear layers with a residual connection.

A.6 Algorithm and Training Hyper-parameters

The MACTA algorithm is explained in Algorithm 1, and detailed training hyperparameters can be found in Table 7. In MACTA, the policy pool is created by sampling a new model to do batch actions per step. It is an infrastructure implementation to produce faster sampling speed, so it is not strictly a per-step sampling, but it is per-step sampling considering a large amount of data. Additionally, we are exploring the per-trajectory policy sampling because it has nice theoretical

properties. Nevertheless, it needs further infrastructure support, and we are exploring that direction.

Algorithm 1 MACTA

Initialize Number of Fictitious Play Iterations I, Attacker Policy Pool $\mathcal{P}_A$, Detector Policy Pool $\mathcal{P}_D$, Number of Epoch per Fictitious Play Iteration E, Add a policy to Pool per N epochs. `PPO` the Proximal Policy Optimization. $\mathbb{U}$ the uniform random sampling of policies per step.
$i \leftarrow 0, j \leftarrow 0, k \leftarrow 0$
while $i < I$ **do**
  $j \leftarrow 0$
  while $j < E$ **do**
    π_{Aj}=PPO($\mathbb{U}(\mathcal{P}_D)$) ▷ Train attacker policy against the pool of the detectors
    if j mod $N - 1$==0 **then**
      $\mathcal{P}_A \leftarrow \mathcal{P}_A \cup \pi_{Aj}$
    end if
    $j \leftarrow j + 1$
  end while
  $j \leftarrow 0$
  while $j < E$ **do**
    π_{Dj}=PPO($\mathbb{U}(\mathcal{P}_A)$) ▷ Train detector policy against the pool of the attackers
    if j mod $N - 1$==0 **then**
      $\mathcal{P}_D \leftarrow \mathcal{P}_D \cup \pi_{Aj}$
    end if
    $j \leftarrow j + 1$
  end while
end while
return π_A, π_D ▷ return the last policy of attacker and detector

Table 7: Training hyper-parameters.

Parameter Group	**Parameter Name**	**Parameter Value**
Fictitious Play	Fictitious Iterations	20 iterations
Fictitious Play	Epochs per Iteration per Agent	50 epochs
Fictitious Play	Training Steps per Epoch	3000 steps
Fictitious Play	Frequency of Adding one Policy to Pool	10 epochs
Computing Resource	Number of Sampling Actors	48 Actors
Computing Resource	Sampling Instance per Worker	2 Actors / Worker
Computing Resource	Remote Model Push Frequency	10 steps
Computing Resource	GPU Information	2 Nvidia Tesla V100 16G / 32G
Computing Resource	CPU Information	80 Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
Proximal Policy Optimization	Replay Buffer Size	262144
Proximal Policy Optimization	Training Batch Size	512
Proximal Policy Optimization	Learning Rate	1e-4
Proximal Policy Optimization	Entropy Coefficient	0.03
Proximal Policy Optimization	Discount Factor γ	0.99
Proximal Policy Optimization	Max Gradient Norm	1.0
Proximal Policy Optimization	GAE λ	0.95
Proximal Policy Optimization	Policy Ratio Clipping ϵ	0.2
Proximal Policy Optimization	Value Clipping ϵ	0.2
Proximal Policy Optimization	Value Loss Coefficient	0.5
Model Architecture	Number of Transformer Heads	8
Model Architecture	Number of Transformer Encoder Layers	1
Model Architecture	Transformer Hidden Dimension	128

A.7 HEURISTIC CACHE TIMING ATTACKS AND DETECTORS

- Heuristic Attacker Algorithms
  - Prime+Probe (Algorithm 2) Osvik et al. (2006). First, in the prime phase, the attacker fills the cache set with its address value (lines 3), then waits for the victim to access the cache set. Next, the victim accesses one of the cache sets, then replaces the loaded address value with its address (lines 5). Lastly, in the probe phase, the attacker accesses the cache sets again, then measures the cache latency to load each set of the primed address value (lines 7 to 8). In a cache set accessed by the victim, the attacker observes increased latency (cache miss).
- Detector Algorithms
  - CC-Hunter Chen & Venkataramani (2014). Cache timing channels rely on the latency of events to perform timing modulation. To send information, two processes (*i.e.*, the trojan and the spy) generate a sufficient number of alternating conflict events (cache misses) to allow the adversary to decode the transmitted bit based on the average memory access times (hit/miss). Those behaviors show periodic, oscillating patterns of conflicts between the two processes. Therefore, autocorrelation is used to identify those patterns. Autocorrelation is the correlation coefficient of the signal with a time-lagged version of itself, along with measuring the event train X, as a variable at a time instance of t. Two cases of conflict miss, *i.e.*, either the victim eviting the attacker's cache line or the attacker evicting the victim's cache line, are considered for the event trains Luo et al. (2022). For example, we can check the autocorrelation C_p at a time lag p, which is expressed as:

$$C_p = \frac{\sum_{i=0}^{n-p} \left[\left(X_i - \bar{X} \right) \left(X_{i+p} - \bar{X} \right) \right]}{\sum_{i=0}^{n} \left(X_i - \bar{X} \right)^2}$$

    If there exists a time lag p which $1 \leq p \leq P$, where P is a predefined parameter such that makes C_p larger than a threshold value, then it is assumed as an attackLuo et al. (2022).
  - Cyclone Harris et al. (2019). The concept of *cyclic* interference is commonly found in all known cache contention side-channel attacks Harris et al. (2019) and has been used for detecting those attack patterns. Interference occurs from the attacker to the victim process or vice versa, considered directional, and affects the behavior of microarchitecture in a disruptive manner. The cyclic interference can be noted as (a $\rightsquigarrow$ b $\rightsquigarrow$ a), where interference (a $\rightsquigarrow$ b) is followed by (b $\rightsquigarrow$ a). However, interference including a third party between attacker and victim, like (a $\rightsquigarrow$ b $\rightsquigarrow$ c), is not considered cyclic interference. To classify the anomaly and benign patterns, Cyclone adapted one-class support vector machine (SVM) and a scalable end-to-end tree boosting system (XGBoost Chen & Guestrin (2016)) models.

Algorithm 2 Prime+Probe Attack

1: $step \leftarrow step + 1$
2: **if** $step < len(attacker\_address\_range)$ **then**
3: action = $prime\_address(step, cache\_size)$ ▷ attacker fills cache by attacker's address
4: **else if** $step = len(attacker\_address\_range)$ **then**
5: action = $trigger\_victim(step)$ ▷ victim accesses a cache and fills its own address
6: **else**
7: action = $probe\_address(step, cache\_size)$ ▷ attacker access caches again
8: measure $latency(action)$
9: **end if**
10: **if** $latency = 1$ **then** ▷ attacker observes for any cache miss
11: action = $guess(action, cache\_size)$ ▷ attacker makes a guess on a victim's secret address
12: **end if**
13: **Return** action