DETER 2.0: Understanding Ethereum Mempool DoS Security via Stateful Fuzzing

Yibo Wang, Wanning Ding, Yuzhe Tang, Kai Li, Jiaqi Chen

***Abstract*—This work tackles the systematic understanding and hardening of Ethereum security against asymmetric denial of mempool service (DETER+).**

First, the paper presents an automatic exploit-generation tool-chain that adaptively discovers attack traces to evade protection in real Ethereum clients. It leads to the discovery of seven new attack patterns. Our evaluation shows that while the recent Ethereum clients (incl. Geth $V1.10.13$ and OpenEthereum $V3.3.4$) are patched against the previous known DETER attack, they are found vulnerable to the newly discovered DETER+ attacks with high success rates ($88\%|96\%$) at low costs (as low as zero Ether/Gas). By evaluation on testnets, DETER+ attacks can be propagated to the entire network and deny the global mining service.

Second, the paper presents an online mitigation scheme that prioritizes transaction validity upon admission and detects malicious transactions that form dual attacks. With a prototype implementation, we conduct experiments and show that the mitigation schemes can effectively push the attack cost to be as high as the baseline attack of sending high-price spam transactions.

***Index Terms*—Blockchains, Ethereum, attack discovery, exploit generation, online detection**

1. Introduction

In a permissionless blockchain network supporting open membership, denial of service (DoS) is a prominent threat. In it, an attacker can use the blockchain's limited resources without being held accountable. While blockchain protocols have been designed to defend against certain DoS (e.g., blockchain mining to discourage Sybil nodes [1]–[3] and charging execution fees to dissuade consuming smart contracts [4], [5]), the DoS security of a blockchain system remains a wide open and ongoing problem, thanks to undefined behaviors and the gap between protocol specification and actual implementation.

An understudied blockchain system component is the memory pool or mempool. On a blockchain node, the mempool buffers unconfirmed transactions prior to mining, transaction propagation and other blockchain services. Had a mempool's service in buffering unconfirmed transactions been denied, the damage can be cascaded to these downstream services, even causing global network disruption. For instance, as demonstrated in a recent study [6], denying a top miner's mempool can force the blockchain network to produce empty blocks.

In this work, we formulate the notion of Asymmetrical DeniAl of Mempool Service, coined DETER+, to abstract a class of particularly practical mempool DoS attacks. In DETER+, the attacker sending crafted transactions to a target mempool aims at denying the mempool's service of buffering unconfirmed transactions and victimizing both the downstream components (e.g., miner) and users sending normal transactions. Unlike the high-cost transaction spamming [7], an DETER+ attacker aims at keeping their cost *asymmetrically* low while achieving the above damage goal. That is, the attacker's cost in paying the crafted transactions' fees should be much lower than the fees of the victim normal transactions, that is, the transactions evicted or declined in the mempool under attacks. In the existing literature, the only known attack achieving DETER+ is DETER [6], which is discovered through manual code inspection.

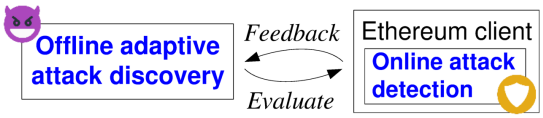


Figure 1: System overview

This work aims to systematically understand and harden Ethereum's mempool security against DETER+ attacks. We approach the goal from two angles: First, given a black-box Ethereum client, we aim to develop offline tools to discover feasible attacks automatically. Second, we aim to design detection modules to harden the target Ethereum client against all discovered attacks. The two angles are complementary, as shown in Figure 1: On the one hand, a comprehensive understanding of feasible attacks through the discovery module is necessary for designing, testing, and evaluating the detection module. On the other hand, an Ethereum client, no matter whether it is hardened with the attack detection, is necessary to provide feedback to the attack discovery module so that its exploit generation can be guided to adaptively evade client-specific protections.

1.1. Systematic Attack Discovery

Automatic exploit generation: We propose an automatic exploit-generation framework for systematical attack discovery that efficiently explores the attack design space. The framework runs in two phases: model checking and concrete exploit generation. Specifically, in the first phase, we abstract complex Ethereum mempool implementations into a simple and checkable model and check the model against as-

sertions induced from DETER+ attacks. The model checking produces feasible patterns to enable DETER+ attacks. This phase is implemented using the explicit-state model-checking tool, TLC [8].

Then, it generates, explores and validates actual attacks against real Ethereum clients by extending, mutating and combining the candidate patterns adaptively. The two phases are complementary as the first phase allows for extensive transaction exploration and the second phase allows for actual exploit validation.

New attacks: Our automatic exploit generation framework can rediscover the two known attacks (i.e., DETER-X and DETER-Z [6]), explore a previously unexplored attack design space, and discover new attack patterns. Specifically, the existing DETER attack works by having an attacker node send invalid transactions and evict normal transactions in a target mempool; we denote this attack pattern by "evict by sending straight invalid transactions." Our framework discovers several new attack primitives beyond the known "evict-by-invalid" pattern: 1) Instead of evicting the normal transactions existing in a mempool, DETER+ can be achieved by "locking" a mempool against subsequent normal transactions by occupying the mempool when it is empty (e.g., the vacant mempool slots created by mining). 2) Instead of sending straight invalid transactions, DETER+ can be achieved by sending stealthier transactions that appear valid at admission and turned into invalid ones at the time of mining. We call this primitive by valid-turned-invalid transactions. 3) DETER+ can also be achieved by avoiding invalid transactions at all and only by sending chargeable but low-cost valid transactions. We call the primitive by low-cost valid transactions. Note that low-cost valid transactions differ from the existing spamming techniques that rely on sending high-cost transactions [7].

We discovered seven new attack patterns by systematically exploring the new attack design space, as shown in Figure 2. Notably, some of the discovered patterns can form dual attacks in the sense that the defense to mitigate one attack can be exploited to successfully mount another attack. Specifically, evicting a mempool using valid-turned-invalid transactions (i.e., upright triangle in Figure 2) can be misused to lock the mempool using low-cost valid transactions (i.e., rectangle in Figure 2).

Attack evaluation: We evaluate automatically generated DETER+ attacks on the two dominant Ethereum clients, that is, Geth and OpenEthereum of different versions. In each experiment, we obtain the DETER+ attacks adaptively generated against a given Ethereum client, and drive the attacks against a local node running the client with mining turned on and receiving normal transactions. Our experiment shows that while the latest versions of both clients are patched to defend against DETER attacks [6], they are vulnerable under the new DETER+ attacks. Specifically, on the latest OpenEthereum/Geth client, while DETER attacks incur a low success rate ($29.5\%/12.5\%$), the new DETER+ attacks can achieve a much higher success rate of $96\%/87.9 - -90.9\%$ and lower costs of $21,000/21,000$ Gas per block. Notably, our tool can generate multi-pattern

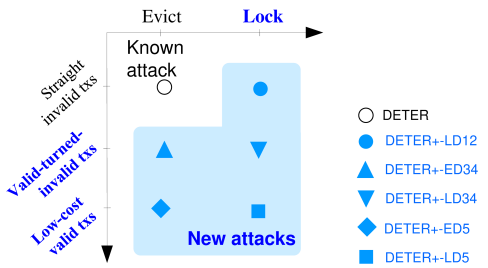


Figure 2: Attacks found by automatic exploit generation: It can rediscover the known DETER attack (in the hollow circle) and discover new DETER+ attacks (in the four dark blue circles). Our work explores previously unexplored attack design space (in light blue shades) by following new attack patterns such as "mempool locking" and "valid-turned-invalid transactions" (in blue text).

DETER+ attacks in an adaptive way to evade the defense adopted in Geth. By sending attacks to the Rinkeby testnet at a constrained load, we show that DETER+ attacks can be propagated from the attacker node to the entire network. We also demonstrate DETER+'s effectiveness in emptying generated blocks at a low cost on the Rinkeby testnet.

We have disclosed the found bugs to the Ethereum developer community through bounty programs, and our bug reports have been confirmed.

1.2. Online Attack Mitigation

Method: Ideally, one may want to mitigate DETER+ attacks precisely (completely and soundly) to achieve both the security against DETER+ and the utility in preserving miner revenue. Moreover, even for complete (and unsound) attack mitigation, the naive design that tracks the formation of all possible DETER+ attacks, while being a generic solution, can be too expensive to be executed online.

We design a secure and lightweight mitigation scheme in two iterations: First, we start from a simple and domain-specific mitigation scheme F1 that enforces transaction priority at the mempool admission time. F1 prioritizes the admission of valid transactions over invalid ones and is designed to mitigate the DETER+ variants exploiting invalid transactions (i.e., circles and triangles in Figure 2). Then, we propose lightweight attack detection called F2 to run on top of F1. F2 hardens the mempool against DETER+ variants that exploit low-cost valid transactions (i.e., the rectangle and diamond in Figure 2). Specifically, F2 monitors the transaction trace, matches it to different attack patterns, estimates the success rate and cost of candidate patterns, and guide the mempool admission decision-making to avoid patterns of positive damage. Thanks to the duality among discovered attack patterns, the mempool may be trapped in a state with no possibility of preventing all DETER+ attacks. In this situation, we make our best efforts and suggest the admission decision with minimal damage.

We implement the attack detection/mitigation in a retrofittable subsystem over Geth with little instrumentation.

The attack detection scheme is a pre-check on Geth's admission decision on any incoming transaction: If the decision is rejected, the scheme further recommends executing a safer admission decision.

Defense evaluation: Through analysis and experiment, we evaluate the security of proposed online mitigation schemes. The result shows that a mempool hardened by F1 and F2 can mitigate the DETER+ patterns generated from our exploit-generation framework, pushing each potential attack's success rate to be 0% or its attack cost to be as high as the baseline transaction spamming [7].

Our evaluation also verifies the duality among attacks by showing that partial mitigation (e.g., running F1 only) does not secure the mempool against all DETER+ attacks. For instance, on a Geth hardened with F1, the mempool locking attack using the low-cost valid transactions (i.e., rectangle in Figure 2) can succeed with a 100% rate at a cost lower than the baseline spamming by order of magnitude.

Contributions: The paper makes contributions as follows:

• *New attacks*: Discovered new attack patterns in the previously unexplored design space to deny Ethereum's mempool service at a symmetric low cost. Built automatic exploit generation tools to adaptive generate and synthesize transaction traces against real-world Ethereum clients. Evaluated the effectiveness and low cost of the new attacks on the latest Ethereum clients patched secure against the known attacks. Also evaluated the effectiveness of the new attacks in denying mining services and propagating malicious transactions in the testnet.

• *Non-trivial defense*: Designed a suite of online migration schemes. Identified the duality among attack patterns and proposed a lightweight online detection scheme. Implemented the mitigation schemes on top of the Geth client. Evaluated the security by showing it can mitigate all discovered attacks.

2. Threat Model

In the threat model, an DETER+ attacker controls a node and joins an Ethereum network. The attacker node is connected to several normal nodes from which the attacker selects her victim node and aims to disable the mempool service there. In practice, the attacker can choose victim nodes serving critical functions in the network to amplify the damage; for instance, as shown in the previous work [6], it is practical for an attacker node to be connected to and select nodes serving RPC services or mining pools. To mount an DETER+ attack, the attacker sends crafted transactions to the victim node and disables its mempool in two ways: 1) evicting the existing transactions in the target mempool, and 2) occupying the mempool with the crafted transactions, so to decline subsequent normal transactions and prevent them from entering the mempool. In other words, an DETER+ attacker sends crafted transactions and victimizes two types of normal transactions, the ones preexisting in the mempool and the ones arriving at the node subsequently.

Besides, an DETER+ attacker aims at lowering her attack's cost asymmetrically, that is, the cost of crafted transactions in a successful DETER+ attack must be lower than that of victim transactions.

3. Exploit Generation Framework

3.1. Design Rationale and Overview

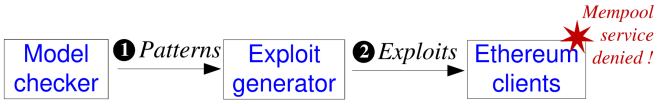


Figure 3: Workflow overview of attack discovery

We formulate the research of exploit generation: Given an Ethereum client, we aim at generating as many successful DETER+ payloads as possible, that is, generating the crafted transaction sequences that can cause successful DETER+ attacks on the client's mempool implementation.

Since the malicious transaction sequence should be at least the same length with the target mempool (e.g., 5120 transactions in Geth's mempool), the input space to search for exploits (e.g., at least 2^{5120}) would be too large to be exhausted. What compounds the problem is that real Ethereum clients admit transactions packed in the unit of "messages", which further bloat the input space (e.g., from a transaction set to a powerset of the transaction set).

We aim at efficiently exploring the large search space without missing valid exploits. We formulate exploit search at two levels: We first build a simple but generic mempool model and extensively search exploits at this abstract level (e.g., by model checking). We then use the patterns as the "seed" exploits to constrain the search space against actual mempool implementation. The idea is that the capability to disable the simple mempool is a precondition for an actual exploit: An actual exploit must be able to deny the simple mempool model and should additionally evade/exploit implementation-specific features in the actual mempool.

The two-level exploit-generation framework is depicted in Figure 3. 1) We first extract generic mempool designs from different Ethereum clients, notably Ethereum transaction validity and price-based transaction admission. We model the DETER+ threats in this simplistic mempool and run model checking to discover the exploit patterns. 2) We then use the patterns to guide the exploit generation against actual mempool implementations in target Ethereum clients. Specifically, the patterns define a much smaller input space than the original unconstrained space (aforementioned). In this smaller space, the proposed search algorithm validates each trial trace iteratively against actual mempool implementation so that the generated exploit can evade various implementation-specific countermeasures adopted.

Next, we first describe the transaction/mempool model and notations used in the rest of the paper. We then describe the model checking, attack patterns and actual exploit generation.

We implement the mempool model in the PlusCal/TLA+ language and use the model-checking tool TLC [8] to find

violations as exploit patterns. We set different settings when checking the model: We use two initial states S_0: a full mempool (F) or an empty mempool ($\varnothing$). We also turn on and off $PP3$, because $PP3$ is on in OpenEthereum and off in Geth.

TABLE 1: Found attack patterns: The trace is shown in the initial state, be it full (F) or empty ($\varnothing$), followed by the sequence of transactions, each represented by its price. With/without [] is the price of an invalid/valid transaction. ✓/✗/– means the feature is required/not required/irrelevant in the success of found patterns. Note that $B = 2$ and the total price of a mempool full of normal transactions is $3B = 6$.

Found patterns	Violation		Required features	
	Trace	Total price	PP2	PP3
ED1	F,[4, 4]	$0 < 6$	–	–
ED2	F,4, [4]	$4 < 6$	–	–
ED3	F,[5, 5]	$0 < 6$	–	✗
ED4	F,5, [4]	$5 < 6$	✓	–
LD1	$\varnothing$, [4, 4]	$0 < 6$	–	–
LD2	$\varnothing$, 1, [4]	$1 < 6$	–	✓
LD3	$\varnothing$, [4, 4]	$0 < 6$	–	✗
LD4	$\varnothing$, 2, [4]	$2 < 6$	✓	✓
LD5	$\varnothing$, 1, 4	$5 < 6$	–	✓

3.2. Step 1: Seed Attack Patterns

We manually found nine unique patterns by checking the mempool model. The model-checking settings that lead to the discovery of each pattern are described in Table 1. Before describing each attack pattern, we first introduce the new attack design "dimensions" and choices discovered.

- **Dimension DD1: Evict or lock mempool**: We found there are two general approaches that can disable a mempool at low cost: By evicting the normal transactions residing in an initially full mempool (i.e., mempool eviction, denoted by ◐) or by populating an initially empty mempool to full so that subsequent normal transactions are declined (i.e., mempool locking, denoted by ◑).
  Note that the DETER attacks fall under the category of "evict-mempool" design.
- **Dimension DD2: How to craft transactions**: A successful DETER+ attack entails occupying the mempool with asymmetrically low-cost (or zero-cost) transactions; there are different ways to craft these transactions: The attacker directly crafts invalid transactions in her payload (the "straight invalid" design, denoted by ◴), or crafts valid transactions that turn existing normal transactions into invalid ones (the "valid-turned-invalid" design, denoted by ◒), or crafts a sequence of low-price valid transactions from the same sender (the "low-price valid" design, denoted by ◔).
  Note that the DETER attacks fall under the category of "straight invalid" design.
- **Dimension DD3: What invalid transactions**: An DETER+ attacker can aim to generate or create invalid transactions in the mempool. There are two types of invalid transactions in Ethereum: future transactions (denoted by ◓) and latent overdraft transactions (denoted by ◒). Both are exploited in the DETER attacks [6].

The nine attack patterns and their design choices are listed in Table 2. Note that patterns ED1 and ED2 correspond to the known attacks of DETER-X and -Z variants [6]. The other seven patterns are new and previously unknown.

TABLE 2: Discovered attack patterns: the first two patterns (ED1/ED2) are known (same with DETER [6]), while the other seven patterns are new and previously unknown ones.

Patterns	DD1	DD2	DD3	Description
ED1	◐	◴	◓	Evict mempool by straight future transactions
ED2	◐	◴	◒	Evict mempool by straight latent overdraft transactions
ED3	◐	◔	◓	Evict mempool by valid-turned-future transactions
ED4	◐	◔	◒	Evict mempool by valid-turned-latent overdraft txs
LD1	◑	◴	◓	Lock mempool by straight future transactions
LD2	◑	◴	◒	Lock mempool by straight latent overdraft transactions
LD3	◑	◔	◓	Lock mempool by valid-turned-future transactions
LD4	◑	◔	◒	Lock mempool by valid-turned-latent overdraft transactions
LD5		◒	◒	Lock mempool by low-cost valid transactions

We describe the attack patterns as follows. Here, we extend the patterns discovered under the 2-slot mempool model to the more generic case with a B-slot mempool. In particular, transactions can be crafted by as many as M accounts.

- **Pattern ED1 (a.k.a., DETER-X [6])**: *It evicts a full mempool by directly crafting future transactions*. In our abstract model, the initial mempool is full of B normal transactions. This pattern contains B future transactions, each of which satisfies nonce $n \geq 2$ (recall initially the attacker's account does not have any transaction in the mempool or blockchain). The future transactions are priced high, such as $p = 4$ in Table 1.
- **Pattern ED2 (a.k.a., DETER-Z [6])**: *It evicts a full mempool by directly crafting latent overdraft transactions.* Against an initially full mempool, the pattern contains M sequences, each of B/M latent overdraft transactions; in each sequence, the parent transaction satisfies $f = i \in [1, M], v = B$ (i.e., sent from an account of index i and spending the full account balance, B Ether (recall we set the initial account balance at a value equal to the mempool length B). The $B/M - 1$ child transactions satisfy $f = i, v > 0$. In other words, after the parent transaction depletes the account balance, the child transactions cause overdraft. All transactions including parent and child ones are priced high, such as $p = 4$ in Table 1.
- **Pattern ED3**: *It evicts a full mempool by crafting valid-turned-future transactions.* Against an initially full mempool of normal transactions, the pattern contains a sequence of $B + M$ valid-turned-future transactions, in

which the first B transactions are M sequences of valid transactions; for the i-th sequence, it contains a parent transaction satisfying $f = i \in [1, M], n = 1, p = 4$ and $B/M - 1$ child transactions satisfying $f = i, n \in [2, B/M], p = 5$. The last M transactions in the pattern satisfy $f \notin [1, M], p = 5$. The attack is successful if the last M transactions evict the M parent transactions (i.e., $n = 1, p = 4$) in the first B transactions.

An ED3 attacker sends valid transactions at the time of admission, and after admission, turns them into invalid ones. This is different from and stealthier than the known ED1 attacker who directly sends invalid transactions at the time of admission. For instance, in a 2-slot mempool, ED1 or DETER-X (the straight-future" design) entails sending two transactions $\langle f1, n2, p4\rangle, \langle f2, n2, p4\rangle$, while ED3 entails sending three transactions $\langle f1, n1, p4\rangle, \langle f1, n2, p5\rangle, \langle f2, n1, p5\rangle$. While both can evict the mempool, ED3 does so by sending valid-looking transactions and can bypass the latest Ethereum's mitigation scheme that restricts the eviction by future transactions (see § 3.4).

- **Pattern ED4**: *It evicts the mempool by crafting valid-turned-latent-overdraft transactions.* Against an initially full of normal transactions, the pattern contains $B + M$ valid-turned-latent-overdraft transactions, which satisfy the following pattern: The first B transactions satisfy $f \in [1, \ldots, M], n1, v = 1, p3$ or $f \in [1, \ldots, M], n \in [2, \ldots, B/M], v = 1, p4$, and the last M transactions satisfy $f \in [1, \ldots, M], n1, v = B, p5$. The attack succeeds if the last M transactions *replace* the first M transactions of nonce 1.
- **Pattern LD1**: *It locks the mempool by directly crafting future transactions.* On an initially empty mempool, it sends the same transactions with Pattern ED1.
- **Pattern LD2**: *It locks the mempool by directly crafting latent overdraft transactions.* On an initially empty mempool, it sends transactions the same with $ED2$ except that the transaction of nonce $n = 1$ is priced at $p = 1$ instead of $p = 4$.

  Unlike LD1 sending identical transactions with ED1, LD2 sends transactions with lower-price than those in ED2. Specifically, an LD2 attacker exploits a certain admission policy (PP3) to lock the mempool at a cost lower than ED2. For instance, in a 2-slot mempool, an ED2 trace is $\langle f1, n1, p4\rangle$, $\langle f1, n2, p4\rangle$, while an LD2 trace is $\langle f1, n1, p1\rangle$, $\langle f1, n2, p4\rangle$. With the mempool supporting PP3, while both traces can disable the mempool, LD2 incurs only $\frac{1}{4}$ of the Ether cost of ED2.
- **Pattern LD3**: *It locks the mempool by crafting valid-turned-future transactions.* On an initially empty mempool, it sends the same transactions with Pattern ED3.
- **Pattern LD4**: *It locks the mempool by crafting valid-turned-latent-overdraft transactions.* On an initially empty mempool, the pattern generates transactions similar to ED4 except that the nonce-1 transactions are first priced at $p = 1$ (instead of $p = 4$) and then the replacing/evicting transactions of nonce $n = 1$ are priced at $p = 2$.
- **Pattern LD5**: Unlike the previous patterns that exploit invalid transactions for extremely low costs (i.e., either one or zero transaction's fee per mempool), LD5 exploits only valid transactions that are harder to defend against. LD5 *locks the mempool by crafting cheap valid transactions.* The attacker sends a sequence of transactions from the same sender where the last transaction (i.e., the transaction with the largest nonce) has a higher price than normal transactions while all other transactions have the minimal price (e.g., $p = 1$).

  Note that exploiting valid transactions to lock may still incur lower costs as will be seen. By contrast, exploiting valid transactions to evict does not constitute a low-cost attack where the attacker pays higher fees than those of victim transactions.

3.3. Step 2: Concrete Exploit Generation

3.3.1. Search Algorithm. The model checking described above abstract away implementation-level details. As a result, the discovered attack pattern is not immediately applicable to actual mempool implementation. For instance, the checkable model is set at 2 transactions, while real mempool implementation commonly stores thousands of transactions (e.g., 5120 transactions in Geth). On the one hand, exhausting the design space of 5120-transaction traces poses computational hardness. On the other hand, naively generating exploit traces under a given pattern may be ineffective because there are too many implementation-level ad-hoc defenses that cannot be modeled or checked as in the previous step. For instance, transaction admission atomicity (i.e., a transaction is admitted to the mempool in an all-or-nothing manner) is a property that is not checked in our abstract model but can be violated in real Ethereum clients due to flawed implementation.

In this subsection, we tackle the challenge of bypassing complex implementation-level defenses and, from the discovered patterns, generate functional exploit traces on real Ethereum clients (recall Figure 3). At the high level, we propose a search algorithm that mutates implementation-level API knobs, synthesizes multiple patterns in generating exploits, and adaptively bypasses defenses.

Concretely, our exploit generation algorithm takes the discovered attack patterns and a target Ethereum client as input. It produces the output of initial states and transaction traces (i.e., the exploits) that can cause successful DETER+ attacks on the client. The algorithm emits a successful exploit if the target Ethereum client processing the exploit reaches one of two following mempool states (i.e., test oracles): 1) Transaction-evicted mempool in which a mempool initially full of normal transactions ends up storing only the transactions in the exploit that has a lower cost than the initial state, 2) Locked mempool in which a mempool declines any normal transactions and stores only the transactions in the exploit that have a lower cost than B normal transactions.

We use a depth-first search strategy in designing our algorithm. That is, the algorithm continues the current decision in exploring the next "bit"/transaction if it makes

progress in reaching successful DETER+ attacks (e.g., a step to evict one more transaction or to lower the attacker cost in the mempool). Otherwise, upon encountering failure, it backtracks and switches to different API knobs (e.g., a mempool receives incoming transactions in variable-length messages) or different attack patterns. By this means, it explores different knobs and synthesizes different patterns in generating an exploit.

Specifically, the algorithm iteratively searches for the exploit in a workflow depicted in Figure 4. This is a two-level nested loop: The inner loop generates the next message under a given setting (meaning fixed pattern, message size, and transaction-generating accounts), appends the message to the current trace, replays the trace against an initial state on the target mempool, and monitors the mempool end state as the feedback. If the latest message is a success (either in evicting existing transactions or in being admitted) and the end state does not meet test oracle, the inner loop continues to the next iteration. Otherwise, if the latest message fails, the algorithm discards it from the current trace and goes to the outer loop to try the next setting. Alternatively, if the end state meets the test oracle, the algorithm emits the current trace as the output exploit and goes to the outer loop.

The outer loop is to iterate through different settings. To find the next setting, it records all the tried settings on the current trace and sets the next setting to be an untried one. If all settings are tried, it removes the latest message in the current trace and backtracks to a previous state until all possibilities are covered. In other words, the outer loop runs a depth-first search (DFS) on the "setting" space. The pseudocode implementing this nested-loop search algorithm is listed in Figure 5.

3.4. Attack Evaluation

This subsection evaluates the effectiveness of newly discovered DETER+ attacks on multiple versions of Ethereum clients. Due to the complex nature of the attacks and defense, our evaluation is conducted from different angles and focuses on answering three research questions, as will be presented. Before that, we first introduce the evaluation platform used.

3.4.1. Evaluation Platform. Our evaluation is set up in the following framework: We run three nodes in a local network, each of which runs the same tested Ethereum client, be it Geth or OpenEthereum. The first node is an attacker who is connected to the second node as a victim and sends the victim crafted transactions. The victim node is also connected to the third node, as the workload generator, and receives from it the normal transactions at a certain rate. There is no connection between the workload-generator node and the attacker node. During the experiment, we let both the workload generator and attacker node send their transactions to the victim node, turn on the mining on the victim node, and observe the blocks generated by the victim node.

The framework is designed to measure the effectiveness of an DETER+ attack. Intuitively, the more the normal transactions are discarded from generated blocks, the more successful an DETER+ attack is. Note that fewer normal transactions included in blocks can be caused by the mempool being occupied by malicious invalid transactions or malicious valid transactions (the latter leads to malicious transactions included in the blocks). Based on this intuition, we propose the first metric: Attack success rate, that is, as shown in Equation 1, the complement of the rate of the number of normal transactions included in the blocks under an attack divided by that without the attack.

$$\text{Success rate} = 1 - \frac{\text{No. of normal txs included under attacks}}{\text{No. of normal txs included without attacks}} \quad (1)$$

Attack success rates only measure an DETER+ attack's interference to the inclusion of normal transactions in blocks. It does not capture the "cost-effectiveness" of the attack. For instance, a baseline transaction spamming attack can achieve high attack success rate but burden the attacker to pay a large amount of Ether. We thus propose the second metric, namely the attack cost, to measure the amount of Gas/Ether paid by the attacker divided by the number of blocks affected by the attack.

To measure the attack cost precisely and realistically, we generate normal transactions using the real-world price value. Specifically, we collect Ethereum transactions from the mempool of a node we run in the mainnet. The collected transactions' price is used as the price of the normal transactions to be generated. Statistically, we found that of 99.99% of the transactions on the Ethereum mainnet, the prices are lower than 1000 Gwei, and we use 1000 Gwei to set the price of evicting/replacing transitions in the attack payload.

Note that to cause disruption in real Ethereum networks, a real-world DETER+ attacker would need to discover critical service nodes in the network to the attack targets, such as top mining pools or RPC nodes. Discovering these critical nodes in real Ethereum networks has been demonstrated feasible in the existing research [6].

3.4.2. Evading Protection in OpenEthereum.

RQ1. How effective are DETER+ attacks in evading the defense of the latest OpenEthereum client?

We customize our evaluation platform with OpenEthereum $V3.3.4$ and run the experiments as described. Here, we present the summarized results in selected experiments. Figure 6 shows the results of selected DETER+ attacks (i.e., ED1, ED3 and LD1) on OpenEthereum V3.3.4. The three selected attack variants all exploit future transactions but vary in different patterns. Specifically, Figure 6a shows the number of included transactions in the generated blocks, and Figure 6b shows the Ether cost paid by the transactions in these blocks. It can be seen that the number of included transactions under attack ED1 is almost the same with that without attacks, showing the ineffectiveness of ED1 on OpenEthereum. ED3

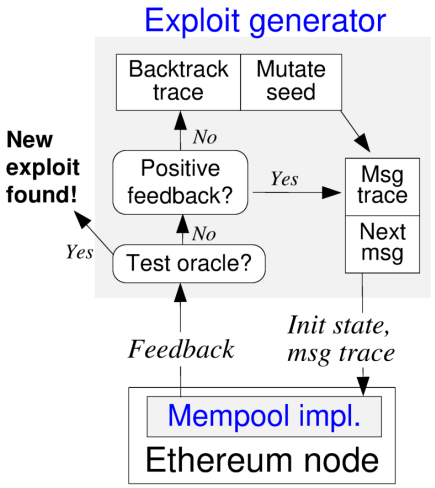
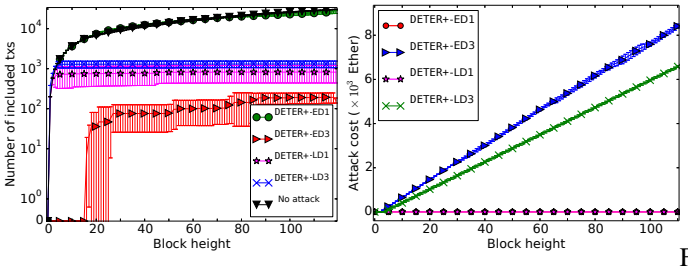


Figure 4: Exploit generator framework

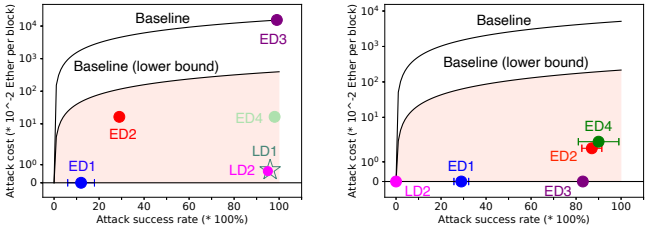
```
1 bool exploit_gen(){
2   //outer loop to iterate thru. different settings
3   while(1){
4     msgtrace.backtrack();
5     if(!nextSenderTried) {
6        seed.nextSender();
7        nextSenderTried=true;
8     } else {
9       nextSenderTried = false;
10      if(!seed.msgsize()<MAX_SIZE) {
11         seed.incMsgsize();
12      } else if(morePattern()) {
13         seed.nextPattern();
14      } else break;
15    } //inner loop to send txs under fixed settings
16    do{
17      msg = seed.nextMsg();
18      msgtrace.add(msg);
19      mempool.init();
20      for(Message msg : msgtrace)
21        feedback = mempool.sendTxs(msg);
22    } while(feedback.isPositive());}}
```

Figure 5: Simplied exploit-generation algorithm.



(a) Success rate (b) Attack costs

Figure 6: Attack timeline on OpenEthereum $V3.3.4$



(a) On OpenEthereum $V3.3.4$ (b) On Geth $V1.10.13$

Figure 7: Attack profiles on Geth and OpenEthereum: Red shade is the danger zone in which the presence of dots represent successful DETER+ attacks.

successfully evicts transactions and reduces the number of transactions included in the blocks, but incurs very high costs; actually, OpenEthereum declines the "turning" transaction in ED3's attack payload. LD1 is highly effective in reducing the number of included transactions and incurs zero Ether cost.

We characterize each attack by two metrics: The success rate (recall Equation 1) and the Ether cost per block. We plot the observed results on OpenEthereum in Figure 7a. The shaded area in the figure shows any instances that are successful DETER+ attacks, i.e., with either higher success rate or lower attack cost than the baseline transaction spamming [7].

The result shows that OpenEthereum is secured against ED3 and LD3 (as both attacks are outside the shaded area). Because OpenEthereum $V3.3.4$ declines the turning transactions in ED3 and LD3. Most lock-based attacks (including LD1, LD2, LD4 and LD5) are successful, because the OpenEthereum's policy to decline turning transactions can be exploited to enable mempool locking. ED4 is also successful on OpenEthereum and can evict 100% of the mempool (at a relatively low cost, $100 * 21000$ Gas per block). This is due to that OpenEthereum does not restrict the replacement transaction if it causes latent overdraft. ED2 and ED1 are more successful than baseline attacks, but they don't achieve 100% of the success rates because of OpenEthereum $V3.3.4$'s policy to decline future and latent overdraft transactions on a full mempool.

In particular, it may seem counter-intuitive that ED4 has higher cost than LD5 in Figure 7a, since ED4 exploits invalid transactions while LD5 only exploits valid transactions. This result can be explained as follows: Consider attacking a block using ED4 on OpenEthereum. To do so, the attacker sends 8192 high-price transactions from at least 102 accounts (to evade its constraint on the number of transactions from the same sender). Therefore, the attacker's cost is from 102 high-price transactions per block. By contrast, for LD5 on OpenEthereum, to attack a block, the attacker can send 380 transactions to exhaust a block's Gas limit (which is set at 8 million Gas in the experiment). This amounts to 5 high-price transactions and 375 low-price transactions. Therefore, although LD5 entails sending more transactions at more Gas to attack a block, its Ether cost is much lower than that of ED4.

3.4.3. Evading Protection in Geth.

RQ2. How effective are DETER+ attacks, both exploiting a single pattern and combining multiple patterns, in evading the defense of the latest Geth client?

We similarly conduct evaluation against the latest Geth client ($V1.10.13$). Unlike OpenEthereum, Geth's policy does allow the eviction of a low-price parent transaction, which makes all lock-based attacks of the same cost as the eviction-based attacks. We thus only label eviction based attacks. The attacks are characterized in Figure 7b. Attacks ED1 and ED3 have zero cost, but don't have high success rates (80% for ED3 and 30% for ED1). Attacks ED2 and ED4 achieve higher but smaller-than-100% success rates, and incur medium costs. By combining multiple patterns together, DETER+ attacks can achieve 100% success rates and zero cost, such as combining ED1 and ED3.

3.4.4. Tx Propagation on Testnet (ED4).

RQ3. How effective are DETER+ attacks (ED4) in propagating malicious transactions to the entire network and in victimizing normal transactions and miner revenue in a testnet?

We run two Geth nodes in the Ethereum Rinkeby testnet, respectively as a measurement node and an attacker node. The attacker node sends crafted transactions to the testnet, and the measurement node is configured to passively[1] receive transactions from its neighbors. There is no direct connection between the two nodes, and the attacker node is configured to connect three neighbors. We remove the limit of peers/neighbors on the measurement node. After running the measurement node in the testnet for seven straight days, we found the number of its neighbors became stable and reached 290; we believe the measurement node is a supernode connecting to all nodes in the testnet.

The measurement node runs an instrumented Geth client to log the received messages from different neighbors; these messages include those of transactions and of transaction hashes. The measurement node could receive the same transaction from different neighbors, and the log stores the transaction-neighbor pairs.

To do an experiment, we make the attacker node send ED4 transactions using 384 accounts. That is, each account sends 16 valid pending transactions of increasing nonces, followed by a replacement transaction. In total, the attacker node sends 6144 valid transactions and 384 replacement transactions. The Gas prices of the valid transactions and the replacement transactions are set to be 40 Gwei and 50 Gwei, respectively. In one setting, we split the 6528 transactions into six messages (i.e., the first five being 1229 transactions each and the last one with 384 transactions). We send these messages in order and wait for 2 seconds between two consecutive messages. For comparison, in another setting, we send the 6144 valid transactions in one message and the 384 replacement transactions in another message. Finally, we wrap up the experiment by waiting after the attacker node sends all messages until all replacement transactions are included in the blockchain. We then turn off the measurement node's logging and search for the presence of all 6528 transactions across neighbor nodes in the log. Given a transaction nonce, we report the percentage of neighbor nodes in the testnet that have propagated it to the measurement node.

We run experiments of six-message and two-message propagation, repeat each experiment by 3 times, and report the average result of node percentage in Figure 8a and Figure 8b. Figure 8a shows the transaction propagation with six messages. More than 90% of the testnet nodes have received all of the 15 valid transactions in a sequence and the replacement transactions. The only exception is the relatively low percentage for the 16-th transaction of the maximal nonce. By comparison, when sending the transactions in two messages,the propagation fails as shown in Figure 8b where almost all nodes did not propagate any transaction with nonce larger than 5.

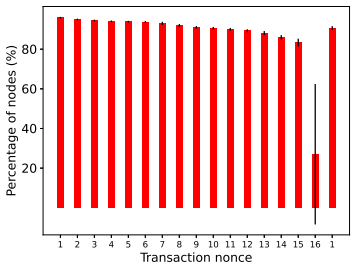
Additionally, Figure 8c shows the generated blocks in the experiment propagating six messages. We took the screenshot from etherscan.io and labeled it in red with information regarding the attack. Before the attack begins, the Rinkeby testnet normally utilizes $8-12\%$ of the Gas in a block. For ethical consideration, we limit our attack to be short, specifically, three blocks long. Right after the attack is launched, the Gas utilization drops significantly to 0.34% in the first block generated (i.e., block number 11186200). In this block, 26.8% of the included transactions labeled in red are the replacement transactions sent in ED4, which, together with the $6144 - 384 = 5760$ child transactions, successfully occupy the mempool. Note that the 5760 child transactions are latent overdraft and are not included in the block. On the second block 11186201, 26.81% of included transactions are replacement transactions in ED4, and the rest 16.54% are normal transactions. On the last block 11186202 before the attack ends, the block utilization drops to 1.83%. After the attack, there is a residual effect such as block 11186204 of utilization 4.01%. Overall, of the three blocks during the attack, the average Gas utilization per block drops to 6.24%, while the pre-attack utilization is consistently above 8.9%. Notice that in our experiments, we didn't target our attacks to the top miner nodes as the existing DETER attacks entail.

4. Related Work

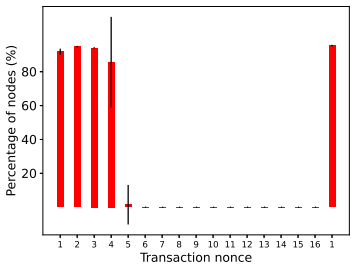
The existing literature has examined a blockchain system's DoS security at different layers, including P2P networks [9]–[12], mining-based consensus [13], [14], transaction processing [7], [15]–[17], and application-level extensions such as smart contracts [4], [18], [19] and DApp (decentralized application) services [20].

On the blockchain mempool security, a baseline DoS is by sending "spam" transactions [7], which incurs high costs (e.g., more than $5,000$ USD just to empty a single Ethereum block [6]) and may not be practical. Existing work [16] analyzes the effect of spamming on transaction

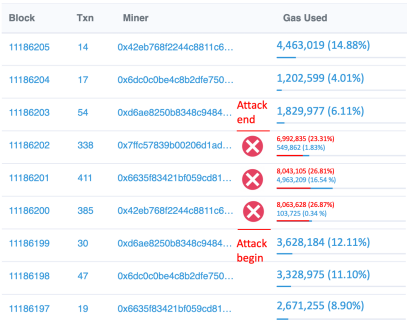
1. We turn off the transaction-propagation capability on the measurement node to avoid missing any received transactions.



(a) Transaction propagation in six messages



(b) Transaction propagation in two messages



(c) Etherscan screenshot of the blocks generated during the attack on Rinkeby (txs in two messages)

Figure 8: Mounting ED4 attacks on Rinkeby: Emptying blocks and propagating malicious transactions to the entire network.

fees and presents countermeasures on Bitcoin mempool. Recent work, DETER [6], has demonstrated the possibility of disabling Ethereum mempool at zero or very low costs. Such damaged nodes cause the global disruption of the real Ethereum networks, including various testnets, which are forced to produce empty blocks.

5. Responsible Disclosure

We have disclosed the discovered bugs to Ethereum client developer communities. Specifically, we have sent bug reports to bounty programs for Geth (Ethereum Foundation) and OpenEthereum. The bugs have been confirmed, and bug fixes are currently in progress.

6. Conclusion

This work tackles the comprehensive understanding and systematic hardening of Ethereum mempool security against DETER+ attacks, that is, asymmetric denial of mempool service attacks. First, we presents an automatic exploit-generation tool-chain and discover seven new attack patterns. High success rates and low attack costs have been shown on the recent Ethereum clients. Furthermore, the discovered attacks can be propagated to the entire network as evaluated on the testnet. Second, we design online mitigation schemes and show with a prototype that all discovered attacks can be mitigated.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," May 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction. Princeton University Press, 2016. [Online]. Available: http://press.princeton.edu/titles/10908.html

[3] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10401. Springer, 2017, pp. 357–388. [Online]. Available: https://doi.org/10.1007/978-3-319-63688-7_12

[4] D. Pérez and B. Livshits, "Broken metre: Attacking resource metering in EVM," in 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/broken-metre-attacking-resource-metering-in-evm/

[5] T. Chen, X. Li, Y. Wang, J. Chen, Z. Li, X. Luo, M. H. Au, and X. Zhang, "An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks," in ISPEC 2017, 2017, pp. 3–24. [Online]. Available: https://doi.org/10.1007/978-3-319-72359-4_1

[6] K. Li, Y. Wang, and Y. Tang, "DETER: denial of ethereum txpool services," in CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 1645–1667. [Online]. Available: https://doi.org/10.1145/3460120.3485369

[7] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, "Stressing out: Bitcoin "stress testing"," in Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers, ser. Lecture Notes in Computer Science, J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. S. Wallach, M. Brenner, and K. Rohloff, Eds., vol. 9604. Springer, 2016, pp. 3–18. [Online]. Available: https://doi.org/10.1007/978-3-662-53357-4_1

[8] L. Lamport, Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2002. [Online]. Available: http://research.microsoft.com/users/lamport/tla/book.html

[9] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in USENIX Security 2015, Washington, D.C., USA, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15

[10] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," IACR Cryptology ePrint Archive, vol. 2018, p. 236, 2018. [Online]. Available: http://eprint.iacr.org/2018/236

[11] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in IEEE Symposium on SP 2017, 2017, pp. 375–392. [Online]. Available: https://doi.org/10.1109/SP.2017.29

[12] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network," in To appear in Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P), 2020.

[13] M. Mirkin, Y. Ji, J. Pang, A. Klages-Mundt, I. Eyal, and A. Juels, "Bdos: Blockchain denial of service," 2019.

[14] "Irreversible transactions: Finney attack," https://en.bitcoin.it/wiki/Irreversible_Transactions\#Finney_attack, Retrieved July, 1, 2022.

[15] "Memoria 700 million stuck in 115,000 unconfirmed bitcoin transactions," https://www.ccn.com/700-million-stuck-115000-unconfirmed-bitcoin-transactions/, Retrieved July, 1, 2022.

[16] M. Saad, L. Njilla, C. A. Kamhoua, J. Kim, D. Nyang, and A. Mohaisen, "Mempool optimization for defending against ddos attacks in pow-based blockchain systems," in IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019. IEEE, 2019, pp. 285–292. [Online]. Available: https://doi.org/10.1109/BLOC.2019.8751476

[17] "Report: Bitcoin (btc) mempool shows backlogged transactions, increased fees if so?" https://goo.gl/LsU6Hq, Retrieved May, 5, 2021.

[18] V. Buterin, "Eip150: Gas cost changes for io-heavy operations." [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-150.md

[19] "Known attacks - ethereum smart contract best practices," https://consensys.github.io/smart-contract-best-practices/known_attacks/\#dos-with-block-gas-limit, Retrieved May, 5, 2021.

[20] K. Li, J. Chen, X. Liu, Y. R. Tang, X. Wang, and X. Luo, "As strong as its weakest link: How to break blockchain dapps at RPC service," in 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021. The Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/as-strong-as-its-weakest-link-how-to-break-blockchain-dapps-at-rpc-service/