WannaBetV2 Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Immeas

December 24, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

WannaBet is a peer-to-peer wagering system that enables two parties (`maker` and `taker`) to create and participate in trustless bets with third-party adjudication by a nominated `judge`. The protocol consists of two primary contracts:

- `BetFactory` deploys individual bet instances using the minimal clone pattern (EIP-1167)
- `Bet` implementation contract that manages the lifecycle and funds for each wager

A bet is initiated when a `maker` specifies a designated `taker`, a trusted `judge`, the wagering asset, stake amounts for both parties, and deadline parameters. The `maker's` stake is transferred upon bet creation, after which the `taker` has until the `acceptBy` timestamp to accept by depositing their stake. Once both parties have committed funds, the bet becomes active and awaits resolution by the designated `judge`, who must declare a winner before the `resolveBy` deadline. If either deadline passes without the required action, the bet expires and can be cancelled by anyone, returning funds to their respective depositors.

The protocol integrates with Aave V3 to generate yield on deposited funds while bets await resolution. The factory owner can configure Aave pool addresses on a per-token basis; when configured bet contracts automatically supply deposited assets to Aave. Upon resolution the winner receives the combined stakes, while any accrued yield is directed to the protocol treasury. This design:

- allows participants' funds to remain productive rather than sitting idle during the betting period
- provides the betting service at no cost to the participants
- provides an income stream for the protocol from the generated yield

5 Audit Scope

The audit scope was limited to:

```
contracts/src/Bet.sol
contracts/src/BetFactory.sol
```

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the WannaBetV2 smart contracts provided by WannaBet. In this period, a total of 23 issues were found.

The findings consist of 3 Medium and 4 Low severity issues with the remainder being informational and gas optimizations.

Of the 3 Mediums:

- 7.1.1 allowed anyone to cancel active bets resulting in denial of service
- 7.1.2 could result in user betting amounts becoming permanently stuck inside the `Bet` contract though with low likelihood
- 7.1.3 allowed the owner of the `BetFactory` contract to subtly rug-pull anyone calling `BetFactory::createBet`, stealing their betting amount

Centralization Risks

Both `BetFactory` and `Bet` contracts are immutable so their functionality can't be changed after deployment. `BetFactory` has an owner who can update:

- Aave pool address associated with given token which only affects future bets (see findings M-3 and I-9)
- `Bet` implementation address which only affects future bets
- treasury address which only affects future bets and has no significant impact

Generally the contracts have been designed to be as trustless as possible, allowing decentralized betting between users with minimal admin interference or control.

The protocol intends to support stablecoins which have blacklisting and freezing functionality so this is a known, very minimal risk.

Summary

Project Name	WannaBetV2
Repository	wannabet-v2
Commit	d73333695488...
Fix Commit	c18aae04ef57...
Audit Timeline	Nov 27th - Nov 28th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	4
Informational	11
Gas Optimizations	5
Total Issues	23

Summary of Findings

[M-1] Active and pending bets can be cancelled by anyone	Resolved
[M-2] In `Bet::cancel` if one transfer reverts but the other succeeds, one users's tokens are permanently locked in the `Bet` contract	Resolved
[M-3] Owner of `BetFactory` can subtly rug-pull any caller to `BetFactory::createBet` stealing their tokens	Resolved
[L-1] Use `SafeERC20` functions instead of standard ERC20 functions	Resolved
[L-2] Judge can designate arbitrary winner who is neither maker nor taker	Resolved
[L-3] Salt used by `BetFactory::createBet` excludes important parameters preventing multiple bets with same timestamps between the same entities	Acknowledged
[L-4] Aave rewards incentives lost as there is no way to claim them	Acknowledged
[I-01] Use named mappings to explicitly indicate the purpose of keys and values	Resolved
[I-02] Use `Ownable2Step` instead of `Ownable`	Resolved
[I-03] Remove obsolete `return` statements when already using named return variables	Resolved
[I-04] Fee-on-transfer and rebasing tokens break accounting	Acknowledged
[I-05] In `Bet::accept,resolve,cancel` update `Bet` state prior to external calls	Resolved
[I-06] `BetFactory::createBet` should revert in a number of scenarios to prevent abuse and ensure bets can be resolved	Resolved
[I-07] `BetResolved` event in `Bet::resolve` will have the wrong `totalWinnings` when Aave pool is used	Resolved
[I-08] Taker receives Aave yield for cancelled pending bets	Resolved
[I-09] `BetFactory::setPool` should validate input pool is legitimate AaveV3 pool and supports input token	Resolved
[I-10] Unused error `IBet::InvalidAmount`	Resolved
[I-11] Unresolved developer comments	Resolved
[G-1] Use named return variables where this can eliminate local variables	Resolved

[G-2] Don't copy entire structs from `storage` to `memory` when only a few slots are required	Acknowledged
[G-3] Cache storage slots to prevent identical storage reads	Resolved
[G-4] Use `type(uint256).max` when withdrawing from Aave	Resolved
[G-5] Remove redundant timestamp check in `Bet::resolve`	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Active and pending bets can be cancelled by anyone

Description: `Bet::cancel` allows any address (except the maker) to cancel a bet while it is in the `ACTIVE` or `PENDING` state:

```
/// @dev Anybody can cancel an expired bet and send funds back to each party. The maker can cancel a
↪  pending bet.
function cancel() external {
    IBet.Bet memory b = _bet;

    // Can't cancel a bet that's already completed
    if (b.status >= IBet.Status.RESOLVED) {
        revert InvalidStatus();
    } else {
        // Pending or active bets at this point
        // The maker can cancel a pending bet, so block them from cancelling an active bet
        // @audit-issue this only prevents `maker` from cancelling, anyone including the taker can
        ↪  cancel an active bet
        if (b.maker == msg.sender && b.status != IBet.Status.PENDING) {
            revert InvalidStatus();
        }
    }
```

The logic only prevents the maker from cancelling an `ACTIVE` bet or anyone from cancelling after the bet has reached a final state. In practice, this means any arbitrary address (including the `taker`) can cancel both `PENDING` and `ACTIVE` bets at any time.

Impact: Allowing any user to cancel a bet enables griefing and, more critically, lets the taker unilaterally back out of a bet they have already accepted. For example, in a sports bet, once the taker observes that their side is likely to lose, they can simply call `cancel()` to unwind the bet and reclaim their stake, undermining the integrity of the betting mechanism.

Proof of Concept: Add following test to `BetFactory.t.sol` which shows that the `taker` can cancel a non-expired `ACTIVE` bet:

```
// Anyone can cancel a non-expired ACTIVE bet
function test_ActiveBetCanBeCancelledByAnyone() public {
    // 1. Have the taker accept the bet so it becomes ACTIVE
    vm.startPrank(taker);
    usdc.approve(address(betNoPool), 1000);
    betNoPool.accept();
    vm.stopPrank();

    // Sanity check: bet is ACTIVE
    assertEq(uint(betNoPool.bet().status), uint(IBet.Status.ACTIVE));

    // 2. Warp to a time before resolveBy so the bet is still non-expired
    IBet.Bet memory state = betNoPool.bet();
    vm.warp(uint256(state.resolveBy) - 1);
    assertEq(uint(betNoPool.bet().status), uint(IBet.Status.ACTIVE));

    uint256 makerBalanceBefore = usdc.balanceOf(maker);
    uint256 takerBalanceBefore = usdc.balanceOf(taker);

    // 3. taker cancels the bet
    vm.prank(taker);
    betNoPool.cancel();

    // 4. After cancellation, the bet is marked CANCELLED and funds are refunded
```

```
    assertEq(uint(betNoPool.bet().status), uint(IBet.Status.CANCELLED));
    assertEq(usdc.balanceOf(maker) - makerBalanceBefore, 1000);
    assertEq(usdc.balanceOf(taker) - takerBalanceBefore, 1000);
}
```

Recommended Mitigation: Consider not allowing anyone to cancel the bet once it's active, alternatively, only allow the judge to cancel active bets.

WannaBet: Fixed in commit cdf3d64.

Cyfrin: Verified.

7.1.2 In `Bet::cancel` if one transfer reverts but the other succeeds, one users's tokens are permanently locked in the `Bet` contract

Description: `Bet::cancel` does this when refunding token transfers:

```
try IERC20(b.asset).transfer(b.maker, makerRefund) {} catch {}
try IERC20(b.asset).transfer(b.taker, takerRefund) {} catch {}
```

But if one of the `transfer` calls reverts but the other succeeds, the transaction still successfully executes. This:

- moves the bet into the `CANCELLED` state
- keeps one user's tokens inside the `Bet` contract

Impact: For the user whose transfer reverted during the cancellation, there is no way for them to withdraw their tokens since `Bet::cancel` can't be called again, the `Bet` contract is immutable and there's no other functions that can rescue the tokens.

Recommended Mitigation: The simplest option is to remove the silent `catch` and revert the entire transaction if one of the `transfer` calls reverts; either both users are refunded or neither are.

A more complicated option is when cancelling a bet:

- update storage `Bet::makerStake,takerStake` to deduct the refunded amount if the transfer succeeded
- add a new function that allows the `maker` or `taker` of a given bet to withdraw their tokens if the bet has been cancelled but the `Bet` record shows they didn't receive a refund because the transfer failed.

WannaBet: Fixed in commit f1750a9.

Cyfrin: Verified.

7.1.3 Owner of `BetFactory` can subtly rug-pull any caller to `BetFactory::createBet` stealing their tokens

Description: Owner of `BetFactory` can rug-pull any caller to `createBet` stealing their tokens by front-running to call `BetFactory::setPool` with the address of a malicious contract that implements the same interface as Aave `supply` function and just transfers tokens to itself.

This works since `Bet::initialize` gives max approval to the pool then calls the `supply` function on it:

```
// If the pool is set, approve and supply the funds to the pool
if (pool != address(0)) {
    IERC20(initialBet.asset).approve(pool, type(uint256).max);

    _aavePool.supply(
        initialBet.asset,
        initialBet.makerStake,
        address(this),
        0
    );
}
```

Recommended Mitigation: In `BetFactory::createBet,predictBetAddress` include `tokenToPool[asset]` in the salt passed to `Clones.cloneDeterministic`. This way if it has changed the new `Bet` contract will be created at a different address which the caller has not approved, the first token transfer in `Bet::initialize` will revert causing the entire transaction to revert.

The recommended fix for I-9 also resolves this issue, since it prevents the owner from setting the Aave pool associated with a token to an illegitimate pool.

WannaBet: Initially we fixed this in commit fbd8016 by including the aave pool in the salt. But after implementing the fixes for I-9 (pool validation) in commit 70e1565 we no longer included the pool in the salt since the pool can only be set to a valid one.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Use `SafeERC20` functions instead of standard ERC20 functions

Description: Use SafeERC20 functions `safeTransfer`, `safeTransferFrom`, `forceApprove` etc instead of standard ERC20 functions to ensure support for ERC20 tokens with non-standard behavior:

```
Bet.sol
63:         IERC20(initialBet.asset).transferFrom(
71:             IERC20(initialBet.asset).approve(pool, type(uint256).max);
114:         IERC20(b.asset).transferFrom(msg.sender, address(this), b.takerStake);
150:         IERC20(b.asset).transfer(winner, totalWinnings);
155:             IERC20(b.asset).transfer(_treasury, remainder);
193:         try IERC20(b.asset).transfer(b.maker, makerRefund) {} catch {}
194:         try IERC20(b.asset).transfer(b.taker, takerRefund) {} catch {}
```

WannaBet: Fixed in commit b571b26.

Cyfrin: Verified.

7.2.2 Judge can designate arbitrary winner who is neither maker nor taker

Description: When calling `Bet::resolve`, the nominated judge can designate an arbitrary winner to receive the winnings.

Impact: The judge can send the winnings to themselves or to any other non-related party who didn't participate in the bet, even though the bet is a contract between the `maker` and `taker`.

Recommended Mitigation: `Bet::resolve` should enforce that the winner is either the `maker` or the `taker`.

WannaBet: Fixed in commit 74654b5.

Cyfrin: Verified.

7.2.3 Salt used by `BetFactory::createBet` excludes important parameters preventing multiple bets with same timestamps between the same entities

Description: Multiple entities (especially programs) may wish to "batch create" multiple bets between themselves that all have the same timestamps.

This is impossible since `BetFactory::createBet` will revert as the salt used is composed of `maker, taker, acceptBy, resolveBy` but doesn't incorporate other fields such as `asset`, `makerStake`, `takerStake` or `betCount`.

Recommended Mitigation: The simplest solution is to just add `betCount` to the salt, but this introduces potential DoS vector via front-running. However this DoS vector is unlikely since it costs the attacker significant gas as they have to deploy the new contract, even though there is no gain apart from the temporary DoS.

Another solution is to add `asset`, `makerStake`, `takerStake` to the salt.

WannaBet: Acknowledged.

7.2.4 Aave rewards incentives lost as there is no way to claim them

Description: When a bet uses an Aave pool, funds are supplied to Aave and can accrue incentive rewards, but the contracts provide no way to claim or forward these rewards (no integration with the Aave incentives controller and no generic sweep of reward tokens).

Impact: All Aave incentive rewards earned by bet positions are effectively lost/stranded, leading to missed revenue over time and potentially unrecoverable reward token balances.

Recommended mitigation: Add a rewards-handling mechanism that can claim incentives via Aave's incentives controller and route them according to the protocol's economics, (treasury):

```
function aave_claimRewards(address reward) external {
    address[] memory assets = new address[](1);
    assets[0] = _bet.asset;
    rewardsController.claimRewards(
        assets,
        type(uint256).max,
        _treasury,
        reward
    );
}
```

WannaBet: Acknowledged.

7.3 Informational

7.3.1 Use named mappings to explicitly indicate the purpose of keys and values

Description: Use named mappings to explicitly indicate the purpose of keys and values:

```
BetFactory.sol
22:     mapping(address => address) public tokenToPool;
```

WannaBet: Fixed in commit a20d1e7.

Cyfrin: Verified.

7.3.2 Use `Ownable2Step` instead of `Ownable`

Description: Use Ownable2Step instead of `Ownable`.

WannaBet: Fixed in commit 4fb5f42.

Cyfrin: Verified.

7.3.3 Remove obsolete `return` statements when already using named return variables

Description: Remove obsolete `return` statements when already using named return variables:

- `Bet::bet`

WannaBet: Fixed in commit c116182.

Cyfrin: Verified.

7.3.4 Fee-on-transfer and rebasing tokens break accounting

Description: When recording a bet, the input data is saved directly to storage:

```
function initialize(
    IBet.Bet calldata initialBet,
    string calldata description,
    address pool,
    address treasury
) external initializer {

    _bet = initialBet;
```

This means that the bet amount recorded is the "full" amount, not what was actually received.

Impact: Bet cancelled and resolution will revert due to broken accounting, resulting in tokens being locked into the contract.

Recommended Mitigation: The easiest solution is to simply not support fee on transfer or rebasing tokens. Otherwise if desiring to support them, tokens should be transferred first then subtract the actual balance from the sent amount to see what was actually received after the fee was deducted, and store that in storage.

For rebasing tokens the mechanism would be similar to how aave balances are handled. Record the total in the contract and split it between maker and taker, for `cancel` and `resolve`. Any left overs can be sent to the `_treasury`.

WannaBet: Acknowledged; we decided to not support these and explicitly added a comment to the code.

7.3.5 In `Bet::accept,resolve,cancel` update `Bet` state prior to external calls

Description: It is wise to follow the Checks-Effects-Interactions [1, 2] pattern; in `Bet::accept,resolve,cancel` the following storage updates should occur prior to making external calls:

```
// `Bet::accept`
_bet.status = IBet.Status.ACTIVE;

// `Bet::resolve`
_bet.winner = winner;
_bet.status = IBet.Status.RESOLVED;

// `Bet::cancel`
_bet.status = IBet.Status.CANCELLED;
```

WannaBet: Fixed in commit 5cda880.

Cyfrin: Verified.

7.3.6 `BetFactory::createBet` should revert in a number of scenarios to prevent abuse and ensure bets can be resolved

Description: `BetFactory::createBet` should revert if:

- `treasury` is `address(0)`
- `msg.sender` and `taker` are the same
- `judge` is the same as `msg.sender` or `taker`

All of the scenarios represent errors either in initialization or bet creation so such transactions should revert. The first scenario where `treasury` is `address(0)` can prevent bet resolution for tokens which revert on transfer to `address(0)`, though cancelling will still work.

WannaBet: We allow maker/taker/judge to be same address as this is useful for testing and doesn't cause any technical issues.

In commits 5ec090f, f1750a9 implemented a fix such that if no treasury has been set any generated yield goes to the winner.

Cyfrin: Verified.

7.3.7 `BetResolved` event in `Bet::resolve` will have the wrong `totalWinnings` when Aave pool is used

Description: In `Bet::resolve`, `BetResolved` is emitted with `totalWinnings = makerStake + takerStake` before checking the actual funds in Aave and capping `totalWinnings` to the contract's `aTokenBalance`:

```
uint256 totalWinnings = b.makerStake + b.takerStake;
emit BetResolved(winner, totalWinnings);

// If the funds are in Aave, withdraw them
if (address(_aavePool) != address(0)) {
    uint256 aTokenBalance = IERC20(_aavePool.getReserveAToken(b.asset)).balanceOf(address(this));

    // @audit totalWinnings changed
    totalWinnings = _min(totalWinnings, aTokenBalance);
    _aavePool.withdraw(b.asset, aTokenBalance, address(this));
}
```

When an Aave pool is used (with yield or loss), the event's `amount` field may not match the actual winnings transferred to the `winner`, making the event misleading for indexers and off-chain consumers.

Consider emitting the event after the Aave accounting has been applied.

WannaBet: Fixed as of commit c18aae0.

Cyfrin: Verified.

7.3.8 Taker receives Aave yield for cancelled pending bets

Description: If a bet uses an Aave pool but never becomes `ACTIVE` (taker never accepts), only the maker's stake is supplied to Aave. In `Bet::cancel`, the recovered Aave balance is still split using maker/taker logic:

```
uint256 aTokenBalance = IERC20(_aavePool.getReserveAToken(b.asset))
    .balanceOf(address(this));
_aavePool.withdraw(b.asset, aTokenBalance, address(this));

makerRefund = _min(makerRefund, aTokenBalance);
takerRefund = _min(takerRefund, aTokenBalance - makerRefund);
```

So the taker can receive part of the maker's accrued yield (or recovered principal) despite never having deposited. However this is balanced out by the maker having priority to be refunded, if for example there was a "negative yield" event in Aave which caused the total amount withdrawn from Aave to be less than what was deposited.

Consider whether:

1) the treasury should receive any Aave generated yield when the bet is cancelled
2) if the taker never accepted, whether they should still receive Aave yield

WannaBet: Fixed in commit f1750a9 such that:

- taker only receives refund if they deposited
- yield is sent either to treasury if it exists or otherwise to the maker

Cyfrin: Verified.

7.3.9 `BetFactory::setPool` should validate input pool is legitimate AaveV3 pool and supports input token

Description: `BetFactory::setPool` should validate input pool is legitimate AaveV3 pool and supports input token. This can be done by:

1) `BetFactory::constructor` should take as input the address of AaveV3 deployed `PoolAddressesProvider` contract and store it into an immutable variable `AAVE_ADDRESSES_PROVIDER`; on Base mainnet this is `0xe20fCBdBfFC4Dd138cE8b2E6FBb6CB49777ad64D`
2) `BetFactory::setPool` should verify that input `_pool` matches `PoolAddressesProvider::getPool`
3) `BetFactory::setPool` should verify that input `_token` is an associated underlying token of the pool by calling `Pool::getReserveAToken` and ensuring the return value is not `address(0)`
4) optionally also verify via `IAToken::UNDERLYING_ASSET_ADDRESS`

Recommended Mitigation: A potential solution which appears to work with the existing test suite, and also resolves finding M-3:

1) first add these additional includes to `BetFactory.sol`:

```
import {IPool} from "@aave-dao/aave-v3-origin/src/contracts/interfaces/IPool.sol";
import {IAToken} from "@aave-dao/aave-v3-origin/src/contracts/interfaces/IAToken.sol";
import {IPoolAddressesProvider} from
↪  "@aave-dao/aave-v3-origin/src/contracts/interfaces/IPoolAddressesProvider.sol";
```

2) Inside `BetFactory`, define this constant and three new errors:

```
contract BetFactory is Ownable {
    // Base mainnet Aave V3 PoolAddressesProvider
    IPoolAddressesProvider public constant AAVE_ADDRESSES_PROVIDER =
        IPoolAddressesProvider(0xe20fCBdBfFC4Dd138cE8b2E6FBb6CB49777ad64D);

    error InvalidPool();
    error TokenNotSupported();
```

```
    error ATokenMismatch();
```

3) Use this new version for `BetFactory::setPool`:

```
    function setPool(address _token, address _pool) external onlyOwner {
        // Allow setting to zero (disable Aave for this token)
        if (_pool == address(0)) {
            tokenToPool[_token] = address(0);
            emit PoolConfigured(_token, address(0));
            return;
        }

        // Validate against canonical registry
        if (_pool != AAVE_ADDRESSES_PROVIDER.getPool()) {
            revert InvalidPool();
        }

        // Verify token is listed
        address aToken = IPool(_pool).getReserveAToken(_token);
        if (aToken == address(0)) {
            revert TokenNotSupported();
        }

        // Verify bidirectional relationship
        if (IAToken(aToken).UNDERLYING_ASSET_ADDRESS() != _token) {
            revert ATokenMismatch();
        }

        tokenToPool[_token] = _pool;
        emit PoolConfigured(_token, _pool);
    }
```

WannaBet: Fixed in commit 70e1565.

Cyfrin: Verified.

7.3.10 Unused error `IBet::InvalidAmount`

Description: The custom error `InvalidAmount` is declared in `IBet` but never used anywhere in the implementation. Consider removing it or using it in the amount check it's intended to validate.

WannaBet: Fixed in commit 6bddf7f.

Cyfrin: Verified.

7.3.11 Unresolved developer comments

Description: There are leftover developer comments that should be removed or resolved:

`Bet::initialize#L62`:

```
// Maybe can skip this and send it striaght to Aave ?
```

and

`IBbet.Bet#L29`

```
// TODO: I feel like there's a more efficient way to represent the maker:taker ratio
```

The first is misleading (the contract must custody funds itself, not send them directly to Aave as the Aave pool might not be present), and the second is an unresolved `TODO`. Consider removing or clarifying these.

WannaBet: Fixed in commit 6bddf7f.

Cyfrin: Verified.

7.4 Gas Optimization

7.4.1 Use named return variables where this can eliminate local variables

Description: Use named return variables where this can eliminate local variables:

- `BetFactory::createBet`
- `Bet::_status`

WannaBet: Fixed in commit 1c5fb9d.

Cyfrin: Verified.

7.4.2 Don't copy entire structs from `storage` to `memory` when only a few slots are required

Description: Don't copy entire structs from `storage` to `memory` when only a few slots are required:

- `Bet::cancel` only needs to read 5 slots `status,maker,makerStake,takerStake,asset` from storage, so save 2 storage reads

Consider caching the required storage slots as they are needed, so that if a revert check triggers the code didn't waste a ton of gas reading a bunch of storage slots that never get used.

WannaBet: Acknowledged.

7.4.3 Cache storage slots to prevent identical storage reads

Description: Cache storage slots to prevent identical storage reads if the values don't change during execution.

For example, `Bet::accept,cancel,resolve` should cache `_aavePool` if it is likely to be non-zero since this saves 1 storage read in `accept` and 2 storage reads in `resolve,cancel`.

WannaBet: Fixed in commit b8b4863.

Cyfrin: Verified.

7.4.4 Use `type(uint256).max` when withdrawing from Aave

Description: When unwinding Aave positions in `Bet::resolve` and `cancel`, the contract withdraws using the current aToken balance as the `amount` parameter:

```
uint256 aTokenBalance = IERC20(_aavePool.getReserveAToken(b.asset))
    .balanceOf(address(this));
_aavePool.withdraw(b.asset, aTokenBalance, address(this));
```

Aave's recommended pattern for fully closing a position is to pass `type(uint256).max`, which is more robust against rounding/indexing edge cases. This also removes one external call as `withdraw` returns the amount withdrawn:

```
uint256 aTokenBalance = _aavePool.withdraw(b.asset, type(uint256).max, address(this));
```

WannaBet: Fixed in commit b060cf6.

Cyfrin: Verified.

7.4.5 Remove redundant timestamp check in `Bet::resolve`

Description: `Bet::resolve` has this revert check:

```
// Make sure the bet is active
if (_status(b) != IBet.Status.ACTIVE || block.timestamp > b.resolveBy) {
    revert InvalidStatus();
}
```

But the call to `_status(b)` already checks `block.timestamp > b.resolveBy` and returns `EXPIRED` status which triggers the revert, so having the same timestamp check here again is redundant.

WannaBet: Fixed in commit 45afa44.

Cyfrin: Verified.