SECURITY AUDIT REPORT

Butter Network — Smart Contract Ecosystem

Client	Butter Network (butterswap.io)
Auditor	ByteScan Security Research Lab
Scope	butter-router-contracts / butter-mos-contracts
Date	April 2026
Version	1.0 — Public Release

01 — Executive Summary

ByteScan Security Research Lab performed an independant security review of the Butter Network smart contract ecosystem, covering both the **butter-router-contracts** and **butter-mos-contracts** GitHub repositorys. The review was focused on live mainnet deployments accross BSC, Base, Arbitrum, Optimism, Polygon, Linea, zkSync, and the MAP Relay Chain.

The audit identified **8 security findings** across a range of severities. All findings affect contracts that are currently live on mainnet. The most impactful deployed finding is a complete bypass of fee collection logic inside **ButterRouterV31.swapAndBridge()**, which causes the protocol to collect zero fees on every single cross-chain bridge transaction across all 7 supported chains. This has been occuring since the contract was first deployed.

8	2	5	1
Total	**High**	**Medium**	**Low**

02 — Audit Scope

Repository	Contract	Chains	Deployed
butter-router-contracts	`ButterRouterV31`	BSC, Base, Arb, Op, Polygon, Linea, zkSync	Yes
butter-router-contracts	`ButterRouterV3`	All EVM chains	Yes
butter-mos-contracts	`BridgeAndRelay`	MAP Relay Chain	Yes
butter-mos-contracts	`MAPOmnichainServiceV2`	Multiple EVM	Yes
butter-mos-contracts	`RelayExecutor`	MAP Relay Chain	Yes
butter-mos-contracts	`AffiliateFeeManager`	MAP Relay Chain	Yes

03 — Findings

F-01	MEDIUM	swapAndBridge() Bypasses Fee Collection Entirely	Deployed
Contract	ButterRouterV31 — 0xEE0319cF0BCa5d09333f9F6277743E8De31bD69A	Chains	BSC, Base, Arbitrum, Optimism, Polygon, Linea, zkSync

Description

The ButterRouterV31 contract has two main entry points: swapAndCall() and swapAndBridge(). When you look at swapAndCall(), it correctly calls _collectFee() before doing anything else, which deducts the protocol and integrator fee from the input amount. However in swapAndBridge(), this call is simply missing. The function only calls _getReferrerFromFeeData() to extract the referrer address, but never actually collects any fee. This means every single cross-chain bridge transaction routed through this contract — accross 7 chains — generates zero fee revenue for the protocol. This has been the case since the contract was first deployed.

Vulnerable Code

```
// swapAndCall() — CORRECT behaviour:
(fd, swapTemp.swapAmount, swapTemp.referrer) = _collectFee(
swapTemp.srcToken,
swapTemp.srcAmount,
_feeData // fee is deducted here
);

// swapAndBridge() — VULNERABLE (fees never collected):
swapTemp.referrer = _getReferrerFromFeeData(_feeData); // only reads referrer
// _collectFee() is NEVER called
(orderId, swapTemp.toChain, receiver) = _doBridge(
msg.sender, swapTemp.swapToken, swapTemp.swapAmount, _bridgeData
// full amount is bridged — no fee deducted
);
```

Recommendation

Add a call to _collectFee() inside swapAndBridge() before the _doBridge() call, mirroring the exact same pattern already used in swapAndCall(). Update swapTemp.swapAmount with the returned remainder value so the bridged amount is correctly reduced by the fee.

F-03	HIGH	fusionReceiver Hardcoded to address(0) — Function Permanently Bricked	Deployed

Contract	BridgeAndRelay — MAP Relay Chain	Chains	MAP Relay Chain

Description

The BridgeAndRelay contract declares fusionReceiver as an immutable constant set to the zero address. There is a public function swapOutTokenWithOrderId() that is gated with require(msg.sender == fusionReceiver). Since msg.sender can never equal address(0) in any normal transaction, this function is permanently unreachable. The comment in the code says 'replace with FusionReceiver address' which strongly suggests this was a deployment mistake where a placeholder was committed instead of the real address. Any integration or protocol flow that relies on order-based bridging through this entry point is completley broken and cannout be used.

Vulnerable Code

```
// Hardcoded constant — cannot be changed after deployment:
address constant fusionReceiver = 0x0000000000000000000000000000000000000000;
// comment says: 'replace with FusionReceiver address'

function swapOutTokenWithOrderId(
address _initiator,
address _token,
bytes memory _to,
uint256 _amount,
uint256 _toChain,
bytes32 orderId,
bytes calldata _bridgeData
) external payable nonReentrant whenNotPaused returns (bytes32) {
require(msg.sender == fusionReceiver); // always fails: 0x000 != any EOA
return _swapOutToken(orderId, _initiator, _token, _to, _amount, _toChain, _bridgeData);
}
```

Recommendation

Replace the constant with a mutable state variable and add a restricted setter function. Then set the correct FusionReceiver address either at deployment or via the setter before any integrations attempt to use this function.

F-04	HIGH	_checkBridgeable() Skipped for External Chain Messages	Deployed

Contract	BridgeAndRelay — MAP Relay Chain	Chains	MAP Relay Chain

Description

When a cross-chain swap arrives at the relay chain from an external source chain (fromChain != selfChainId), the bridgeability check is completely skipped in the _swapRelay() function. The check only fires when the swap originates from the relay chain itself. This means an attacker or misconfigured frontend can route a swap through an unregistered token or destination chain combination. The relay chain will mint and then burn the relay token, but the destination chain never receives anything because the token is not registered there. The funds are permanently lost with no recovery path.

Vulnerable Code

```
function _swapRelay(...) internal {
if (_relay && _inEvent.swapData.length != 0) {
// relay execute ...
}

// check ONLY fires when coming from the relay chain itself:
if(_inEvent.fromChain == selfChainId)
_checkBridgeable(_inEvent.token, _inEvent.toChain);
// if fromChain != selfChainId — NO check is performed
// unregistered token/chain combos proceed silently

_inEvent.amount = _collectTochainFee(_initiator, _inEvent);
// tokens get burned on relay chain with no delivery on dest chain
}
```

Recommendation

Move the _checkBridgeable() call outside of the if-block so it applies to all incoming messages regardless of their originating chain. The bridgeability of a token to a destination chain should always be validated before any fees are collected or tokens are burned.

F-05	MEDIUM	Caller-Supplied orderId Disconnected from Proof in swapIn()	Deployed

Contract	`MAPOmnichainServiceV2 — Multiple EVM chains`	Chains	Ethereum, BSC, Polygon and others

Description

The swapIn() function accepts a caller-supplied _orderId parameter and checks it against the orderList mapping to prevent replay attacks. However, the actual orderId used for the replay protection inside _swapIn() comes from the verified proof (outEvent.orderId), not from the caller's parameter. These two IDs are never compared against each other. This means the top-level check is effectivley a no-op for replay protection purposes. A griefing scenario is possible where an attacker submits a valid proof for order A while passing the ID of a known future order B as _orderId, poisoning order B's slot in the orderList.

Vulnerable Code

```
function swapIn(
uint256 _chainId,
uint256 _logIndex,
bytes32 _orderId, // supplied by CALLER
bytes calldata _receiptProof
) external nonReentrant whenNotPaused {
require(!orderList[_orderId], 'order exist'); // checks CALLER id
...
(bool success, string memory message, ILightVerifier.txLog memory log) =
lightNode.verifyProofDataWithCache(false, _logIndex, _receiptProof);
require(success, message);
_swapInVerifiedWithIndex(log, _logIndex);
// inside here: checkOrder(outEvent.orderId) — checks PROOF id
// the two IDs are never compared — totally independent checks
}
```

Recommendation

Remove the top-level require(!orderList[_orderId]) check since it provides no actual replay protection. The real protection is already correctly handled by checkOrder(outEvent.orderId) inside _swapIn(). Alternatively, add a check that _orderId == outEvent.orderId after the proof is verified to ensure they match.

F-06	MEDIUM	depositToken() Never Assigns orderId — MessageIn Event Silently Skipped	Deployed

Contract	BridgeAndRelay — MAP Relay Chain	Chains	MAP Relay Chain

Description

In the depositToken() function of BridgeAndRelay, the return variable orderId is declared but never actually assigned a value. It stays as bytes32(0) throughout the function. When _deposit() is called internally, it receives this zero value and then checks if(_orderId != bytes32('')) — which evaluates to false — so the MessageIn event is never emitted. Light clients and any off-chain monitoring infrastructure that tracks deposits by listening for MessageIn events will silently miss all deposits made through this path. This could cause deposits to appear as pending or lost in tracking systems.

Vulnerable Code

```
function depositToken(
address _token,
address _to,
uint256 _amount
) external payable override nonReentrant whenNotPaused
returns (bytes32 orderId) { // orderId declared here
address from = msg.sender;
address token = _tokenTransferIn(_token, from, _amount, true, false);
_deposit(token, Helper._toBytes(from), _to, _amount, orderId, selfChainId);
// orderId is still bytes32(0) — never assigned!
// inside _deposit():
// if (_orderId != bytes32('')) { <-- false, skips MessageIn emit
// emit MessageIn(...);
// }
}
```

Recommendation

Add orderId = _getOrderId(...) with the appropriate parameters before the _deposit() call. This is the same pattern correctly used in swapOutToken() and should be straightforward to add.

F-07	MEDIUM	depositWhitelist Address Hardcoded as Immutable Constant	Deployed
Contract	`BridgeAndRelay — MAP Relay Chain`	Chains	MAP Relay Chain

Description

The BridgeAndRelay contract hardcodes the DepositWhitelist contract address as an immutable constant at the top of the contract. There is also a typo in the variable name (depositWhitelsit). Because its a constant, this address cannot be changed after deployment. If the whitelist contract ever needs to be upgraded, or if the address is incorrect on a newly deployed chain, all depositToken() calls will interact with the wrong address. Depending on what code lives at that address on different chains, this could cause silent failures or unexpected behaviour. The rest of the codebase correctly uses configurable state variables for similar contracts.

Vulnerable Code

```
// Hardcoded — cannot be updated post-deployment:
address constant depositWhitelsit = // note: typo in variable name
0x27172dA6b48DB586B5261ff90D6D1D5F2C1c1363;

function _deposit(...) internal {
if(IDepositWhitelist(depositWhitelsit) // always calls this fixed address
.checkTokenAmountAndWhitelist(_token, _to, _amount)){
...
} else {
require(_orderId != bytes32(''));
_tokenTransferOut(_token, _to, _amount, true);
}
}
```

Recommendation

Replace the constant with a storage variable and add a restricted setter (e.g. via the existing restricted modifier pattern used elsewhere in the contract). Also fix the typo in the variable name while making this change.

F-08	MEDIUM	Silent Fee Failure Passes Zero Amount Downstream in RelayExecutor	Deployed

Contract	`RelayExecutor — MAP Relay Chain`	Chains	MAP Relay Chain

Description

In the RelayExecutor contract, the _collectFee() function wraps the fee collection call in a try/catch block. If the feeManager call reverts for any reason, the catch block does nothing and the amountOut variable stays at its default value of zero. This zero amount then flows into the next step of _relayExecute() where it is used for swapping or forwarding tokens. Calling a swap with amount zero will typically revert or produce unexpected results downstream. The intended behaviour on a fee collection failure should be to pass through the full original amount without any fee deduction, not to silently return zero.

Vulnerable Code

```
function _collectFee(bytes32 _orderId, address _token,
uint256 _amount, bytes calldata _message)
internal returns (uint256 offset, uint256 amountOut) {
// amountOut starts as 0 (default uint256 value)
uint256 len = uint256(uint8(bytes1(_message[offset:(offset += 1)])));
if (len == 0) { return (offset, _amount); }
try feeManager.collectAffiliatesFee(...) returns (uint256 totalFee) {
IERC20(_token).safeTransfer(address(feeManager), totalFee);
amountOut = _amount - totalFee; // success case: correct
} catch (bytes memory) {
// amountOut is still 0 here — NOT set to _amount
// downstream receives 0 tokens
}
}
```

Recommendation

In the catch block, set amountOut = _amount so that on a fee collection failure the full amount is passed through unchanged. This matches the expected graceful degradation behaviour where fee failures should not block the underlying transaction.

F-09	LOW	AffiliateFeeManager feeData Parsing Inconsistency	Deployed

Contract	AffiliateFeeManager — MAP Relay Chain	Chains	MAP Relay Chain

Description

The AffiliateFeeManager contract has two functions that process the same feeData byte array but parse it differently. getAffiliatesFee() reads a leading one-byte length field at offset 0 to determine how many affiliates are encoded. collectAffiliatesFee() ignores this byte entirely and instead computes the count as feeData.length / 4. If feeData is constructed with the leading length byte (as the view function implies), collectAffiliatesFee() will misparse it — treating the length byte plus the first 3 affiliate bytes as the first record, producing garbage affiliate IDs and rates. This would cause fees to be assigned to wrong affiliates or not collected at all.

Vulnerable Code

```
// getAffiliatesFee() — reads length byte first:
uint256 len = uint256(uint8(bytes1(feeData[offset:(offset += 1)])));
// then loops len times over 4-byte records

// collectAffiliatesFee() — ignores length byte:
uint256 len = feeData.length / 4;
// starts reading from offset 0, not offset 1
// if feeData has a leading byte, record[0] is misaligned
for (uint256 i = 0; i < len; i++) {
uint16 id = uint16(bytes2(feeData[offset:(offset += 2)]));
uint16 rate = uint16(bytes2(feeData[offset:(offset += 2)]));
// id and rate are wrong if feeData has a leading length byte
}
```

Recommendation

Standardise the feeData format across both functions. Either add the leading length byte handling to collectAffiliatesFee(), or remove it from getAffiliatesFee(). Add a NatSpec comment documenting the exact expected byte layout of feeData to prevent future regressions.

04 — Recommendations Summary

ID	Severity	Action Required
F-01	MEDIUM	Add _collectFee() call in swapAndBridge() before _doBridge(). One line fix.
F-03	HIGH	Replace constant fusionReceiver with a configurable storage variable.
F-04	HIGH	Apply _checkBridgeable() to all relay paths regardless of fromChain value.
F-05	MEDIUM	Remove the redundant top-level orderList check on the caller-supplied _orderId.
F-06	MEDIUM	Assign orderId = _getOrderId(...) in depositToken() before calling _deposit().
F-07	MEDIUM	Replace hardcoded depositWhitelsit constant with a settable storage variable.
F-08	MEDIUM	Set amountOut = _amount in the catch block of _collectFee() in RelayExecutor.
F-09	LOW	Align feeData parsing format between getAffiliatesFee() and collectAffiliatesFee().

05 — Disclaimer

This report reflects the findings of an independent security review conducted by ByteScan Security Research Lab in April 2026. The scope of the review was limited to the code contained in the referenced GitHub repositorys at the time of review. It does not constitute a guarantee of complete security and does not rule out the existance of additional vulnerabilities outside the reviewed scope. Security is an ongoing process — a clean audit does not mean a contract is free of bugs found at a later date.

All findings were reported to the Butter Network team via responsible disclosure before this report was made public. ByteScan is not responsible for any losses incured as a result of the vulnerabilities described in this document.

Auditor	Contact	Website
ByteScan Security Research Lab	audit@bytescan.net	bytescan.net