Module 2: Liquidation Detection Systems

Duration: 75 minutes | **Level:** Beginner-Intermediate | **Author:** Obelisk Core

Learning Objectives

By the end of this module, you will be able to:
- Understand liquidation mechanics in lending protocols
- Build real-time monitoring systems for at-risk positions
- Calculate optimal liquidation amounts and profits
- Implement liquidation bots with proper risk management
- Navigate ethical considerations in liquidations

Understanding Liquidation Mechanics

Liquidation in DeFi occurs when a borrower's collateral ratio falls below the required maintenance margin. This typically happens when the value of borrowed assets increases relative to collateral, making the position under-collateralized.

Key Protocols and Their Mechanics

Aave Protocol
- **Health Factor**: Primary metric for liquidation risk
- **Liquidation Threshold**: Typically 80-85% for most assets
- **Liquidation Bonus**: 5-10% incentive for liquidators
- **Gas Costs**: Moderate (~$20-40 for liquidation transactions)

Compound Protocol
- **Collateral Factor**: Percentage of asset value that can be borrowed
- **Liquidation Incentive**: 8% for most assets
- **Close Factor**: Maximum percentage of debt that can be liquidated per transaction
- **Gas Costs**: Lower than Aave (~$15-25)

MakerDAO
- **Liquidation Ratio**: 150% for most collateral types
- **Liquidation Penalty**: 13% penalty for under-collateralized positions
- **Auction System**: English auctions for collateral liquidation
- **Gas Costs**: Higher due to complex auction mechanics

Health Factor Calculations

Aave Health Factor Formula:

```
Health Factor = (Total Collateral Value in ETH × Liquidation
Threshold) / Total Borrow Value in ETH
```

Example Calculation:
- User deposits 10 ETH as collateral
- Current ETH price: $2,000
- Borrows 16,000 USDC ($1 each)
- Liquidation threshold: 80%

```
Health Factor = (10 × $2,000 × 0.80) / 16,000 = 16,000 / 16,000 = 1.0
```

Risk Levels:
- Health Factor > 1.5: Safe
- Health Factor 1.2-1.5: Caution zone
- Health Factor 1.0-1.2: At risk
- Health Factor < 1.0: Liquidatable

Building Monitoring Systems

Real-Time Position Monitoring

```
import asyncio
import aiohttp
import logging
from dataclasses import dataclass
from typing import List, Dict, Optional
from datetime import datetime, timedelta

@dataclass
class PositionRisk:
    user_address: str
    protocol: str
    health_factor: float
    collateral_value: float
    borrow_value: float
    liquidation_threshold: float
    bonus_percentage: float
    estimated_profit: float
    timestamp: float
    risk_level: str

class LiquidationMonitor:
    def __init__(self, web3_provider: str, private_key: str):
        self.w3 = Web3(Web3.HTTPProvider(web3_provider))
        self.private_key = private_key
        self.account = self.w3.eth.account.from_key(private_key)

        # Protocol addresses (Ethereum Mainnet)
        self.aave_lending_pool =
"0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9"
        self.aave_Collateral_manager =
"0x8dFf5E27EA6b7AC08EbFdf9eB090F32ee9a30fcf"

        self.comp_comptroller =
"0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B"
        self.comp_price_oracle =
"0x922018674c19a8aeAdeE3d388269722C42162D03"

        # Monitoring configuration
        self.min_profit_threshold = 50  # Minimum $50 profit
        self.max_gas_price = 100  # Maximum 100 gwei
        self.min_health_factor = 1.1  # Monitor positions below this
```

```
    async def monitor_aave_positions(self) -> List[PositionRisk]:
        """Monitor Aave positions for liquidation opportunities"""
        positions = []

        # Aave user account data ABI (simplified)
        aave_abi = [
            {
                "inputs": [{"internalType": "address", "name": "user",
"type": "address"}],
                "name": "getUserAccountData",
                "outputs": [
                    {"internalType": "uint256", "name":
"totalCollateralETH", "type": "uint256"},
                    {"internalType": "uint256", "name": "totalDebtETH",
"type": "uint256"},
                    {"internalType": "uint256", "name":
"availableBorrowsETH", "type": "uint256"},
                    {"internalType": "uint256", "name":
"currentLiquidationThreshold", "type": "uint256"},
                    {"internalType": "uint256", "name": "ltv", "type":
"uint256"},
                    {"internalType": "uint256", "name": "healthFactor",
"type": "uint256"}
                ],
                "stateMutability": "view",
                "type": "function"
            }
        ]

        try:
            # Get all Aave users (simplified - in practice, you'd
iterate through events)
            users = await self.get_aave_users()

            lending_pool = self.w3.eth.contract(
                address=self.aave_lending_pool,
                abi=aave_abi
            )

            for user in users[:100]:  # Limit for demo
                try:
                    user_data =
```

```
lending_pool.functions.getUserAccountData(user).call()

                    if len(user_data) >= 6 and user_data[5] <
self.min_health_factor * 1e18:
                        health_factor = user_data[5] / 1e18
                        collateral_eth = user_data[0] / 1e18
                        debt_eth = user_data[1] / 1e18
                        threshold = user_data[3] / 1e4

                        # Estimate profit potential
                        max_liquidation = min(debt_eth * 0.5,
collateral_eth * 0.95)
                        bonus_rate = 0.075  # 7.5% liquidation bonus
                        estimated_profit = max_liquidation * bonus_rate
* 2000  # ETH price ~$2000

                        position = PositionRisk(
                            user_address=user,
                            protocol="Aave",
                            health_factor=health_factor,
                            collateral_value=collateral_eth,
                            borrow_value=debt_eth,
                            liquidation_threshold=threshold,
                            bonus_percentage=bonus_rate,
                            estimated_profit=estimated_profit,
                            timestamp=datetime.now().timestamp(),

risk_level=self.assess_risk_level(health_factor)
                        )

                        if estimated_profit >=
self.min_profit_threshold:
                            positions.append(position)

                except Exception as e:
                    logging.warning(f"Error monitoring user {user}:
{e}")
                    continue

        except Exception as e:
            logging.error(f"Error in Aave monitoring: {e}")
```

```
        return positions

    async def monitor_compound_positions(self) -> List[PositionRisk]:
        """Monitor Compound positions for liquidation opportunities"""
        positions = []

        # Compound Comptroller ABI (simplified)
        comp_abi = [
            {
                "inputs": [{"internalType": "address", "name":
"account", "type": "address"}],
                "name": "getAccountLiquidity",
                "outputs": [
                    {"internalType": "uint256", "name": "", "type":
"uint256"},
                    {"internalType": "uint256", "name": "", "type":
"uint256"},
                    {"internalType": "uint256", "name": "", "type":
"uint256"}
                ],
                "stateMutability": "view",
                "type": "function"
            }
        ]

        try:
            comptroller = self.w3.eth.contract(
                address=self.comp_comptroller,
                abi=comp_abi
            )

            users = await self.get_compound_users()

            for user in users[:100]:
                try:
                    liquidity_data =
comptroller.functions.getAccountLiquidity(user).call()
                    liquidity = liquidity_data[1]  # Liquidity in USD
(18 decimal)

                    if liquidity == 0:  # User might be at risk
                        # Get user's positions
```

```
                        user_positions = await
self.get_user_compound_positions(user)

                        for position in user_positions:
                            if position['liquidity'] < 0:
                                # Calculate liquidation potential
                                debt_value = abs(position['liquidity'])
                                collateral_value =
position['collateral_value']

                                if collateral_value > 0:
                                    health_ratio = collateral_value /
(collateral_value + debt_value)

                                    if health_ratio < 0.85:
# Close to liquidation
                                        liquidation_bonus = 0.08  # 8%
bonus
                                        estimated_profit = debt_value *
liquidation_bonus

                                        if estimated_profit >=
self.min_profit_threshold:
                                            position_risk =
PositionRisk(
                                                user_address=user,
                                                protocol="Compound",

health_factor=health_ratio,

collateral_value=collateral_value,

borrow_value=debt_value,

liquidation_threshold=0.85,

bonus_percentage=liquidation_bonus,

estimated_profit=estimated_profit,

timestamp=datetime.now().timestamp(),
```

```
risk_level=self.assess_risk_level(health_ratio)
                                                )

positions.append(position_risk)

                except Exception as e:
                    logging.warning(f"Error monitoring Compound user
{user}: {e}")
                    continue

        except Exception as e:
            logging.error(f"Error in Compound monitoring: {e}")

        return positions

    def assess_risk_level(self, health_factor: float) -> str:
        """Assess risk level based on health factor"""
        if health_factor < 1.0:
            return "CRITICAL"
        elif health_factor < 1.1:
            return "HIGH"
        elif health_factor < 1.2:
            return "MEDIUM"
        else:
            return "LOW"
```

Position Data Collection

```
class PositionDataCollector:
    def __init__(self, w3: Web3):
        self.w3 = w3

    async def get_user_assets(self, user_address: str, protocol: str) -
> Dict:
        """Get user's assets across different protocols"""
        if protocol == "Aave":
            return await self.get_aave_user_assets(user_address)
        elif protocol == "Compound":
            return await self.get_compound_user_assets(user_address)
        else:
            return {}

    async def get_aave_user_assets(self, user_address: str) -> Dict:
        """Get user's Aave positions"""
        # Get user's reserve data
        lending_pool = self.w3.eth.contract(
            address="0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9",
            abi=self.get_aave_abi()
        )

        try:
            # Get all user reserves
            user_reserves =
lending_pool.functions.getUserReservesData(user_address).call()

            positions = {
                'collateral': [],
                'borrowed': [],
                'total_collateral_usd': 0,
                'total_borrowed_usd': 0
            }

            for reserve_data in user_reserves:
                asset = reserve_data[0]
                scaled_balance = reserve_data[1] / 1e18
                scaled_debt = reserve_data[2] / 1e18

                if scaled_balance > 0:
                    # Get asset price
                    oracle = self.w3.eth.contract(
```

```
address="0xA50ba011c48123De63b5785Fb408e8fD5831949F",
                        abi=self.get_aave_oracle_abi()
                    )

                    asset_price =
oracle.functions.getAssetPrice(asset).call() / 1e8
                    position_value = scaled_balance * asset_price

                    positions['collateral'].append({
                        'asset': asset,
                        'amount': scaled_balance,
                        'value_usd': position_value
                    })
                    positions['total_collateral_usd'] += position_value

                if scaled_debt > 0:
                    oracle = self.w3.eth.contract(

address="0xA50ba011c48123De63b5785Fb408e8f5831949F",
                        abi=self.get_aave_oracle_abi()
                    )

                    asset_price =
oracle.functions.getAssetPrice(asset).call() / 1e8
                    debt_value = scaled_debt * asset_price

                    positions['borrowed'].append({
                        'asset': asset,
                        'amount': scaled_debt,
                        'value_usd': debt_value
                    })
                    positions['total_borrowed_usd'] += debt_value

            return positions

        except Exception as e:
            logging.error(f"Error getting Aave positions for
{user_address}: {e}")
            return {}
```

Profitability Analysis

Liquidation Profit Calculations

Aave Liquidation Profit Formula:

```
Profit = (Liquidation Amount × Bonus Rate × Collateral Price) - Gas
Costs
```

Example Calculation:
- Position: 5 ETH collateral, $16,000 debt
- Health Factor: 1.05 (liquidatable)
- Current ETH Price: $2,000
- Liquidation Bonus: 7.5%
- Gas Cost: $30

```
Max Liquidation = <span class="math-inline" style="display:
inline;"><math xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mn>16</mn><mo>&#x0002C;</mo><mn>000</mn><mi>×</
mi><mn>0.5</mn><mo>&#x0003D;</mo></mrow></math></span>8,000
ETH Amount = <span class="math-inline" style="display: inline;"><math
xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mn>8</mn><mo>&#x0002C;</mo><mn>000</
mn><mo>&#x0002F;</mo></mrow></math></span>2,000 = 4 ETH
Bonus Received = 4 ETH × 0.075 = 0.3 ETH
Gross Profit = 0.3 ETH × <span class="math-inline" style="display:
inline;"><math xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mn>2</mn><mo>&#x0002C;</mo><mn>000</
mn><mo>&#x0003D;</mo></mrow></math></span>600
Net Profit = <span class="math-inline" style="display: inline;"><math
xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mn>600</mn><mo>&#x02212;</mo></mrow></math></
span>30 = $570
```

Advanced Profit Optimization

```
class ProfitOptimizer:
    def __init__(self, w3: Web3):
        self.w3 = w3
        self.gas_calculator = GasCalculator()

    def calculate_optimal_liquidation(self, position: PositionRisk) ->
Dict:
        """Calculate optimal liquidation amount"""

        # Get current gas price
        current_gas_price = self.w3.eth.gas_price

        # Calculate gas cost for different liquidation amounts
        amounts_to_test = [0.1, 0.2, 0.3, 0.4, 0.5]
# Fractions of max debt
        best_profit = 0
        optimal_amount = 0

        for fraction in amounts_to_test:
            liquidation_amount = position.borrow_value * fraction
            bonus_amount = liquidation_amount *
position.bonus_percentage
            gross_profit = bonus_amount * 2000  # Assuming ETH price

            # Estimate gas cost
            gas_cost = self.gas_calculator.estimate_liquidation_gas(
                position.protocol,
                liquidation_amount
            ) * current_gas_price * 1e-9 * 2000  # Convert to USD

            net_profit = gross_profit - gas_cost

            if net_profit > best_profit:
                best_profit = net_profit
                optimal_amount = liquidation_amount

        return {
            'optimal_amount': optimal_amount,
            'expected_profit': best_profit,
            'profit_margin': best_profit / position.borrow_value if
position.borrow_value > 0 else 0
        }
```

```
    def assess_execution_risk(self, position: PositionRisk) -> Dict:
        """Assess execution risks"""
        risks = {
            'gas_risk': 'LOW',
            'competition_risk': 'MEDIUM',
            'slippage_risk': 'LOW',
            'protocol_risk': 'LOW'
        }

        # Gas risk assessment
        current_gas_price = self.w3.eth.gas_price
        if current_gas_price > 100e9:  # 100 gwei
            risks['gas_risk'] = 'HIGH'
        elif current_gas_price > 50e9:  # 50 gwei
            risks['gas_risk'] = 'MEDIUM'

        # Competition risk (based on health factor)
        if position.health_factor < 1.05:
            risks['competition_risk'] = 'HIGH'
        elif position.health_factor < 1.1:
            risks['competition_risk'] = 'MEDIUM'

        return risks
```

Execution Strategies

Liquidation Bot Architecture

```
class LiquidationBot:
    def __init__(self, config: Dict):
        self.monitor = LiquidationMonitor(config['web3_provider'],
config['private_key'])
        self.executor = LiquidationExecutor(config['private_key'])
        self.risk_manager = RiskManager(config)
        self.profit_optimizer =
ProfitOptimizer(config['web3_provider'])
        self.performance_tracker = PerformanceTracker()

    async def run_liquidation_strategy(self):
        """Main liquidation strategy loop"""
        while True:
            try:
                # Monitor for opportunities
                aave_opportunities = await
self.monitor.monitor_aave_positions()
                compound_opportunities = await
self.monitor.monitor_compound_positions()

                all_opportunities = aave_opportunities +
compound_opportunities

                # Filter and rank opportunities
                filtered_opportunities = await
self.filter_opportunities(all_opportunities)

                # Execute top opportunities
                for opportunity in filtered_opportunities[:3]:
# Top 3 opportunities
                    success = await
self.execute_liquidation(opportunity)

                    # Track performance

self.performance_tracker.record_execution(opportunity, success)

                await asyncio.sleep(1)  # Check every second

            except Exception as e:
                logging.error(f"Error in liquidation strategy: {e}")
                await asyncio.sleep(5)
```

```
    async def filter_opportunities(self, opportunities:
List[PositionRisk]) -> List[PositionRisk]:
        """Filter and rank liquidation opportunities"""
        filtered = []

        for opp in opportunities:
            # Risk assessment
            risks = self.risk_manager.assess_execution_risk(opp)

            if risks['gas_risk'] == 'HIGH':
                continue

            # Profit optimization
            optimal_amount =
self.profit_optimizer.calculate_optimal_liquidation(opp)

            if optimal_amount['expected_profit'] >= 50:  # Minimum $50
profit
                opp.estimated_profit =
optimal_amount['expected_profit']
                opp.optimal_amount = optimal_amount['optimal_amount']
                filtered.append(opp)

        # Sort by profit potential
        return sorted(filtered, key=lambda x: x.estimated_profit,
reverse=True)

    async def execute_liquidation(self, opportunity: PositionRisk) ->
bool:
        """Execute liquidation for a specific opportunity"""
        try:
            # Prepare transaction
            if opportunity.protocol == "Aave":
                tx_hash = await
self.executor.execute_aave_liquidation(opportunity)
            elif opportunity.protocol == "Compound":
                tx_hash = await
self.executor.execute_compound_liquidation(opportunity)
            else:
                return False
```

```
            # Wait for confirmation
            receipt = self.w3.eth.wait_for_transaction_receipt(tx_hash)

            if receipt.status == 1:
                logging.info(f"Liquidation successful:
{opportunity.user_address}")
                return True
            else:
                logging.warning(f"Liquidation failed:
{opportunity.user_address}")
                return False

        except Exception as e:
            logging.error(f"Error executing liquidation: {e}")
            return False
```

Gas Optimization

```
class GasOptimizer:
    def __init__(self, w3: Web3):
        self.w3 = w3

    def estimate_liquidation_gas(self, protocol: str, amount: float) ->
int:
        """Estimate gas cost for liquidation"""
        base_gas = {
            'Aave': 200000,
            'Compound': 150000,
            'MakerDAO': 300000
        }

        amount_gas = int(amount * 1000)
# Additional gas per dollar of liquidation

        return base_gas.get(protocol, 200000) + amount_gas

    def calculate_gas_price_strategy(self, urgency: str) -> Dict:
        """Calculate optimal gas price strategy"""
        current_gas = self.w3.eth.gas_price

        strategies = {
            'normal': {
                'max_fee': current_gas * 2,
                'max_priority_fee': current_gas * 1.2,
                'gas_limit': 'auto'
            },
            'urgent': {
                'max_fee': current_gas * 3,
                'max_priority_fee': current_gas * 2,
                'gas_limit': 'auto'
            },
            'budget': {
                'max_fee': current_gas * 1.5,
                'max_priority_fee': current_gas * 1.1,
                'gas_limit': 'conservative'
            }
        }

        return strategies.get(urgency, strategies['normal'])
```

Risk Management

Position Sizing and Exposure

```
class PositionSizer:
    def __init__(self, config: Dict):
        self.max_position_size = config.get('max_position_size', 0.1)
# 10% of portfolio
        self.max_daily_exposure = config.get('max_daily_exposure',
1000)  # $1000 daily
        self.daily_exposure = 0

    def calculate_position_size(self, opportunity: PositionRisk,
portfolio_value: float) -> float:
        """Calculate appropriate position size"""
        # Maximum position size based on portfolio
        max_portfolio_position = portfolio_value *
self.max_position_size

        # Maximum position size based on opportunity
        max_opportunity_position = opportunity.optimal_amount

        # Take the smaller value
        recommended_position = min(max_portfolio_position,
max_opportunity_position)

        # Check daily exposure limits
        if self.daily_exposure + recommended_position >
self.max_daily_exposure:
            recommended_position = self.max_daily_exposure -
self.daily_exposure

        return recommended_position

    def update_daily_exposure(self, amount: float):
        """Update daily exposure tracking"""
        self.daily_exposure += amount

        # Reset if new day
        if self.is_new_day():
            self.daily_exposure = amount

    def is_new_day(self) -> bool:
        """Check if it's a new trading day"""
        # Implementation for tracking trading days
        return False  # Simplified
```

Circuit Breakers

```
class CircuitBreaker:
    def __init__(self, config: Dict):
        self.max_consecutive_failures =
config.get('max_consecutive_failures', 5)
        self.consecutive_failures = 0
        self.cooldown_period = config.get('cooldown_period', 300)  # 5
minutes
        self.cooldown_end = 0

    def should_allow_trading(self) -> bool:
        """Check if trading should be allowed"""
        if time.time() < self.cooldown_end:
            return False

        return self.consecutive_failures <
self.max_consecutive_failures

    def record_failure(self):
        """Record trading failure"""
        self.consecutive_failures += 1

        if self.consecutive_failures >= self.max_consecutive_failures:
            self.cooldown_end = time.time() + self.cooldown_period
            logging.warning(f"Circuit breaker activated after
{self.consecutive_failures} failures")

    def record_success(self):
        """Record successful trade"""
        self.consecutive_failures = max(0, self.consecutive_failures -
1)
```

Ethical Considerations

Liquidation Ethics

While liquidations are a core mechanic of DeFi protocols, practitioners should consider:

Positive Aspects:
- Maintain protocol solvency
- Prevent bad debt accumulation

- Ensure system stability
- Provide emergency liquidity

Ethical Guidelines:
1. **Transparency**: Use public infrastructure, not private mempools
2. **Fairness**: Don't front-run other legitimate liquidators
3. **Proportionality**: Liquidate only what's necessary
4. **Communication**: Consider notifying users when possible

Alternative Approaches

Slow Liquidation Bots:
- Monitor positions approaching liquidation threshold
- Allow users time to add collateral or repay debt
- Lower competition but longer monitoring periods

Collaborative Systems:
- Share profitable opportunities with other bots
- Coordinate to avoid gas wars
- Split profits based on detection time

Performance Analysis

Key Metrics

```
class LiquidationPerformanceTracker:
    def __init__(self):
        self.trades = []
        self.daily_stats = {}

    def track_execution(self, opportunity: PositionRisk, success: bool,
actual_profit: float = None):
        """Track liquidation execution metrics"""
        self.trades.append({
            'timestamp': datetime.now(),
            'protocol': opportunity.protocol,
            'user_address': opportunity.user_address,
            'health_factor': opportunity.health_factor,
            'estimated_profit': opportunity.estimated_profit,
            'actual_profit': actual_profit,
            'success': success,
            'gas_used': self.get_last_gas_used(),
            'execution_time': self.get_last_execution_time()
        })

        self.update_daily_stats()

    def calculate_metrics(self) -> Dict:
        """Calculate performance metrics"""
        if not self.trades:
            return {}

        successful_trades = [t for t in self.trades if t['success']]

        total_opportunities = len(self.trades)
        successful_executions = len(successful_trades)
        success_rate = successful_executions / total_opportunities if
total_opportunities > 0 else 0

        total_estimated_profit = sum(t['estimated_profit'] for t in
successful_trades)
        total_actual_profit = sum(t['actual_profit'] for t in
successful_trades if t['actual_profit'])

        return {
            'total_opportunities': total_opportunities,
            'successful_executions': successful_executions,
```

```
            'success_rate': success_rate,
            'total_estimated_profit': total_estimated_profit,
            'total_actual_profit': total_actual_profit,
            'average_profit_per_execution': total_actual_profit /
successful_executions if successful_executions > 0 else 0,
            'profit_accuracy': total_actual_profit /
total_estimated_profit if total_estimated_profit > 0 else 0
        }
```

Real-World Implementation

Production Deployment Checklist

1. **Infrastructure**
   - [ ] Redundant Web3 providers
   - [ ] Database for position tracking
   - [ ] Real-time monitoring dashboards
   - [ ] Alert systems for failures
2. **Security**
   - [ ] Secure private key management
   - [ ] Multi-signature wallets for large positions
   - [ ] Rate limiting and API protection
   - [ ] Transaction simulation before execution
3. **Compliance**
   - [ ] Legal review of liquidation activities
   - [ ] Tax reporting for profits
   - [ ] Regulatory compliance for different jurisdictions
4. **Testing**
   - [ ] Paper trading for strategy validation
   - [ ] Testnet deployment and testing
   - [ ] Stress testing under high network congestion

Quick Check Quiz

1. **What is the primary metric used by Aave to determine liquidation risk?**
   a) Collateral factor
   b) Health factor
   c) Liquidation ratio
   d) Loan-to-value ratio

2. **What typically happens when a position's health factor falls below 1.0?**
   a) Automatic liquidation
   b) Position becomes liquidatable
   c) Interest rate increases
   d) Position is frozen
3. **What percentage is a common liquidation bonus in Aave?**
   a) 3%
   b) 7.5%
   c) 15%
   d) 25%
4. **What should be considered when calculating liquidation profits?**
   a) Only the bonus amount
   b) Bonus minus gas costs
   c) Only market impact
   d) Just transaction fees
5. **What is an ethical concern in liquidation strategies?**
   a) Using public vs private mempools
   b) Maximizing profit extraction
   c) Competing with other liquidators
   d) Gas price optimization

Summary

Liquidation detection systems are a cornerstone of profitable MEV strategies. Key takeaways:

- **Health Monitoring**: Continuous monitoring of health factors across protocols
- **Profit Analysis**: Accurate calculation of liquidation profitability including all costs
- **Risk Management**: Proper position sizing and exposure limits
- **Execution Speed**: Fast detection and execution increase success rates
- **Ethical Considerations**: Balance profit with responsible protocol stewardship

In the next module, we'll explore gas optimization techniques that can significantly improve strategy profitability.

Next Module Preview

Module 3: Gas Optimization Techniques
- Understanding EIP-1559 gas mechanics
- Dynamic gas pricing strategies
- Transaction bundling and optimization
- Layer 2 solutions for gas savings
- Advanced techniques for gas-efficient MEV strategies

Author: Obelisk Core
Module Duration: 75 minutes
Prerequisites: Completion of Module 1, basic understanding of lending protocols
Resources: Monitoring tools, liquidation calculators, and testing frameworks