Open-Core Monetization Model

Overview

Implement a sustainable open-core business model that maintains Audityzer's commitment to open source while generating revenue through premium enterprise features, advanced services, and value-added offerings. This model ensures long-term sustainability while keeping core security testing capabilities freely available.

Open-Core Architecture

Core vs. Premium Feature Split

```
┌──────────────────────────────────────────────────────────────┐
│                    Audityzer Ecosystem                     │
├──────────────────────────────────────────────────────────────┤
│                      Premium Features                        │
│                                                            │
│  ┌─────────────┐ ┌─────────────┐ ┌─────────────┐           │
│  │ Enterprise  │ │  Advanced   │ │ Professional│           │
│  │   Suite     │ │  Analytics  │ │   Support   │           │
│  │             │ │             │ │             │           │
│  └─────────────┘ └─────────────┘ └─────────────┘           │
├──────────────────────────────────────────────────────────────┤
│                    Core Features (MIT)                       │
│                                                            │
│  ┌─────────────┐ ┌─────────────┐ ┌─────────────┐           │
│  │ Vulnerability│ │    Basic    │ │   Plugin    │            │
│  │  Detection  │ │  Reporting  │ │   System    │           │
│  └─────────────┘ └─────────────┘ └─────────────┘           │
│                                                            │
└──────────────────────────────────────────────────────────────┘
```

Feature Classification Framework

```
// Feature classification system
class FeatureClassification {
  constructor() {
    this.categories = {
      'core': {
        license: 'MIT',
        description: 'Essential security testing functionality',
        target: 'Individual developers, small teams, open source projects'
      },
      'professional': {
        license: 'Commercial',
        description: 'Advanced features for professional use',
        target: 'Professional auditors, medium-sized teams'
      },
      'enterprise': {
        license: 'Commercial',
        description: 'Enterprise-grade features and support',
        target: 'Large organizations, enterprise customers'
      }
    };
  }

  classifyFeature(feature) {
    const coreFeatures = [
      'basic_vulnerability_detection',
      'standard_reporting',
      'cli_interface',
      'plugin_system',
      'community_support',
      'basic_ci_integration'
    ];

    const professionalFeatures = [
      'advanced_analytics',
      'custom_rule_engine',
      'priority_support',
      'advanced_reporting',
      'team_collaboration',
      'api_access'
    ];

    const enterpriseFeatures = [
      'sso_integration',
      'audit_trails',
      'compliance_reporting',
      'dedicated_support',
      'on_premise_deployment',
      'custom_integrations'
    ];

    if (coreFeatures.includes(feature)) return 'core';
    if (professionalFeatures.includes(feature)) return 'professional';
    if (enterpriseFeatures.includes(feature)) return 'enterprise';

    return 'core'; // Default to core for new features
  }
}
```

Revenue Streams

1. Professional Subscriptions

Subscription Tiers

```yaml
subscription_tiers:
  professional:
    price: "$99/month per user"
    annual_discount: "20%"
    features:
      - "Advanced vulnerability detection rules"
      - "Custom reporting templates"
      - "Priority email support"
      - "Team collaboration tools"
      - "API access (10,000 calls/month)"
      - "Advanced analytics dashboard"
      - "Custom rule engine"
      - "Integration with premium tools"

    limits:
      users: 10
      projects: 50
      api_calls: 10000
      storage: "100GB"
      support_response: "24 hours"

  enterprise:
    price: "$499/month per organization"
    annual_discount: "25%"
    features:
      - "All Professional features"
      - "SSO integration (SAML, OIDC)"
      - "Audit trails and compliance reporting"
      - "On-premise deployment option"
      - "Dedicated customer success manager"
      - "Custom integrations"
      - "Advanced security features"
      - "White-label options"

    limits:
      users: "unlimited"
      projects: "unlimited"
      api_calls: "unlimited"
      storage: "1TB"
      support_response: "4 hours"
      sla: "99.9% uptime"

  enterprise_plus:
    price: "Custom pricing"
    features:
      - "All Enterprise features"
      - "Custom feature development"
      - "Dedicated infrastructure"
      - "24/7 phone support"
      - "On-site training and consulting"
      - "Custom SLA agreements"
      - "Multi-region deployment"
```

Subscription Management System

```
class SubscriptionManager {
  constructor() {
    this.tiers = ['free', 'professional', 'enterprise', 'enterprise_plus'];
    this.billingCycles = ['monthly', 'annual'];
  }

  async createSubscription(userId, tier, billingCycle) {
    const subscription = {
      id: this.generateSubscriptionId(),
      userId,
      tier,
      billingCycle,
      status: 'active',
      createdAt: new Date(),
      currentPeriodStart: new Date(),
      currentPeriodEnd: this.calculatePeriodEnd(billingCycle),
      features: this.getTierFeatures(tier),
      limits: this.getTierLimits(tier),
      pricing: this.getTierPricing(tier, billingCycle)
    };

    await this.processPayment(subscription);
    await this.activateFeatures(userId, subscription.features);
    await this.sendWelcomeEmail(userId, subscription);

    return subscription;
  }

  async upgradeSubscription(subscriptionId, newTier) {
    const subscription = await this.getSubscription(subscriptionId);
    const prorationAmount = this.calculateProration(subscription, newTier);

    // Process upgrade payment
    await this.processUpgradePayment(subscription, prorationAmount);

    // Update subscription
    subscription.tier = newTier;
    subscription.features = this.getTierFeatures(newTier);
    subscription.limits = this.getTierLimits(newTier);
    subscription.updatedAt = new Date();

    await this.updateSubscription(subscription);
    await this.activateFeatures(subscription.userId, subscription.features);

    return subscription;
  }

  async handleUsageOverage(userId, resource, usage) {
    const subscription = await this.getUserSubscription(userId);
    const limit = subscription.limits[resource];

    if (usage > limit) {
      const overage = usage - limit;
      const overageRate = this.getOverageRate(subscription.tier, resource);
      const overageCharge = overage * overageRate;

      await this.processOverageCharge(subscription, resource, overageCharge);
      await this.notifyUserOfOverage(userId, resource, overage, overageCharge);
    }
  }
}
```

2. Enterprise Services

Professional Services Offerings

```
class ProfessionalServices {
  constructor() {
    this.services = {
      'security_audit': {
        name: 'Comprehensive Security Audit',
        description: 'Full security assessment by certified experts',
        duration: '2-4 weeks',
        pricing: {
          small_project: 15000,
          medium_project: 35000,
          large_project: 75000,
          enterprise_project: 150000
        },
        deliverables: [
          'Detailed vulnerability report',
          'Risk assessment and prioritization',
          'Remediation recommendations',
          'Executive summary',
          'Follow-up consultation'
        ]
      },

      'custom_rule_development': {
        name: 'Custom Detection Rules',
        description: 'Develop custom vulnerability detection rules',
        duration: '1-2 weeks',
        pricing: {
          simple_rule: 2500,
          complex_rule: 7500,
          rule_suite: 20000
        },
        deliverables: [
          'Custom detection rules',
          'Test cases and validation',
          'Documentation',
          'Integration support'
        ]
      },

      'integration_consulting': {
        name: 'Integration Consulting',
        description: 'Help integrate Audityzer into existing workflows',
        duration: '1-3 weeks',
        pricing: {
          basic_integration: 5000,
          advanced_integration: 15000,
          enterprise_integration: 35000
        },
        deliverables: [
          'Integration architecture',
          'Custom connectors',
          'Workflow automation',
          'Training and documentation'
        ]
      },

      'training_program': {
        name: 'Security Training Program',
        description: 'Comprehensive Web3 security training',
        duration: '1-5 days',
        pricing: {
          workshop_half_day: 2500,
          workshop_full_day: 5000,
          multi_day_program: 15000,
          custom_curriculum: 25000
        },
        deliverables: [
          'Custom training curriculum',
          'Hands-on workshops',
          'Certification program',
          'Ongoing support'
        ]
      }
    };
  }

  async requestService(serviceType, requirements) {
    const service = this.services[serviceType];
    if (!service) {
      throw new Error('Service not available');
    }

    const quote = await this.generateQuote(service, requirements);
    const proposal = await this.createProposal(service, requirements, quote);

    return {
      service,
      quote,
      proposal,
      estimatedDelivery: this.calculateDeliveryDate(service, requirements)
    };
  }

  generateQuote(service, requirements) {
    const basePrice = this.calculateBasePrice(service, requirements);
    const complexity = this.assessComplexity(requirements);
    const timeline = this.estimateTimeline(service, requirements);

    const quote = {
      basePrice,
      complexityMultiplier: complexity,
      totalPrice: basePrice * complexity,
      timeline,
      validUntil: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000), // 30 days
      terms: this.getServiceTerms(service)
    };

    return quote;
  }
}
```

3. Marketplace & Partnerships

Plugin Marketplace Revenue

```
class PluginMarketplace {
  constructor() {
    this.revenueShare = {
      'free_plugins': 0,
      'paid_plugins': 0.30, // 30% platform fee
      'premium_plugins': 0.25, // 25% platform fee for verified developers
      'enterprise_plugins': 0.20 // 20% platform fee for enterprise partners
    };
  }

  async publishPlugin(developerId, plugin, pricing) {
    const listing = {
      id: this.generateListingId(),
      developerId,
      plugin,
      pricing,
      category: plugin.category,
      status: 'pending_review',
      publishedAt: null,
      downloads: 0,
      revenue: 0,
      rating: 0,
      reviews: []
    };

    // Review process
    const review = await this.reviewPlugin(plugin);
    if (!review.approved) {
      listing.status = 'rejected';
      listing.rejectionReason = review.reason;
      return listing;
    }

    // Publish plugin
    listing.status = 'published';
    listing.publishedAt = new Date();

    await this.addToMarketplace(listing);
    return listing;
  }

  async processPurchase(userId, pluginId, licenseType) {
    const plugin = await this.getPlugin(pluginId);
    const price = plugin.pricing[licenseType];
    const platformFee = price * this.revenueShare[plugin.category];
    const developerRevenue = price - platformFee;

    // Process payment
    const payment = await this.processPayment(userId, price);

    // Distribute revenue
    await this.distributePlatformRevenue(platformFee);
    await this.distributeDeveloperRevenue(plugin.developerId, developerRevenue);

    // Grant license
    const license = await this.grantLicense(userId, pluginId, licenseType);

    return {
      payment,
      license,
      plugin
    };
  }

  async calculateMarketplaceRevenue(timeframe) {
    const sales = await this.getSales(timeframe);

    const revenue = {
      totalSales: 0,
      platformRevenue: 0,
      developerRevenue: 0,
      transactionCount: sales.length,
      topPlugins: [],
      revenueByCategory: {}
    };

    for (const sale of sales) {
      revenue.totalSales += sale.amount;
      revenue.platformRevenue += sale.platformFee;
      revenue.developerRevenue += sale.developerRevenue;
    }

    return revenue;
  }
}
```

4. Training & Certification

Educational Revenue Streams

```
class EducationalServices {
  constructor() {
    this.offerings = {
      'certification_programs': {
        'security_tester': {
          price: 299,
          duration: '4 weeks',
          format: 'online',
          includes: ['course_materials', 'practice_exams', 'certification']
        },
        'defi_security_expert': {
          price: 599,
          duration: '8 weeks',
          format: 'online + live sessions',
          includes: ['advanced_materials', 'mentorship', 'project_review']
        },
        'enterprise_training': {
          price: 2500,
          duration: 'custom',
          format: 'on-site or virtual',
          includes: ['custom_curriculum', 'dedicated_instructor', 'ongoing_support']
        }
      },

      'premium_content': {
        'advanced_courses': {
          price: 99,
          access: 'lifetime',
          includes: ['video_content', 'hands_on_labs', 'community_access']
        },
        'masterclasses': {
          price: 199,
          access: '1 year',
          includes: ['expert_sessions', 'case_studies', 'networking']
        }
      },

      'corporate_training': {
        'team_workshops': {
          price: 5000,
          duration: '2 days',
          max_participants: 20,
          includes: ['custom_content', 'hands_on_training', 'follow_up']
        },
        'ongoing_program': {
          price: 15000,
          duration: '6 months',
          includes: ['monthly_sessions', 'progress_tracking', 'certification']
        }
      }
    };
  }

  async enrollInProgram(userId, programType, programId) {
    const program = this.offerings[programType][programId];
    if (!program) {
      throw new Error('Program not found');
    }
    const enrollment = {
      id: this.generateEnrollmentId(),
      userId,
      programType,
      programId,
      price: program.price,
      enrolledAt: new Date(),
      status: 'active',
      progress: 0,
      completionDate: null,
      certificateIssued: false
    };

    await this.processPayment(userId, program.price);
    await this.grantAccess(userId, program);
    await this.trackEnrollment(enrollment);

    return enrollment;
  }

  async calculateEducationalRevenue(timeframe) {
    const enrollments = await this.getEnrollments(timeframe);

    const revenue = {
      totalRevenue: 0,
      enrollmentCount: enrollments.length,
      completionRate: 0,
      revenueByProgram: {},
      averageRevenuePerUser: 0
    };

    for (const enrollment of enrollments) {
      revenue.totalRevenue += enrollment.price;

      if (!revenue.revenueByProgram[enrollment.programId]) {
        revenue.revenueByProgram[enrollment.programId] = 0;
      }
      revenue.revenueByProgram[enrollment.programId] += enrollment.price;
    }

    revenue.averageRevenuePerUser = revenue.totalRevenue / enrollments.length;

    return revenue;
  }
}
```

Pricing Strategy

Value-Based Pricing Model

```
class PricingStrategy {
  constructor() {
    this.pricingFactors = {
      'value_delivered': 0.4,
      'market_position': 0.3,
      'cost_structure': 0.2,
      'competitive_landscape': 0.1
    };
  }

  calculateOptimalPricing(product, market) {
    const valueScore = this.assessValueDelivered(product);
    const marketScore = this.assessMarketPosition(market);
    const costScore = this.assessCostStructure(product);
    const competitiveScore = this.assessCompetitiveLandscape(market);

    const weightedScore =
      (valueScore * this.pricingFactors.value_delivered) +
      (marketScore * this.pricingFactors.market_position) +
      (costScore * this.pricingFactors.cost_structure) +
      (competitiveScore * this.pricingFactors.competitive_landscape);

    const basePrice = this.getBasePrice(product);
    const optimalPrice = basePrice * (1 + weightedScore);

    return {
      basePrice,
      optimalPrice,
      factors: {
        valueScore,
        marketScore,
        costScore,
        competitiveScore
      },
      recommendation: this.generatePricingRecommendation(optimalPrice, market)
    };
  }

  assessValueDelivered(product) {
    // Calculate value based on:
    // - Time saved for users
    // - Risk reduction provided
    // - Cost of alternative solutions
    // - ROI for customers

    const timeSaved = product.metrics.averageTimeSaved; // hours per month
    const riskReduction = product.metrics.vulnerabilitiesFound; // number of issues
    const alternativeCost = product.metrics.competitorPricing; // market rate

    const valueScore = (timeSaved * 100) + (riskReduction * 50) + (alternativeCost * 0.
5);
    return Math.min(valueScore / 10000, 1.0); // Normalize to 0-1
  }

  generatePricingRecommendation(optimalPrice, market) {
    const recommendations = [];

    if (optimalPrice > market.averagePrice * 1.5) {
      recommendations.push({
        type: 'premium_positioning',
        message: 'Price supports premium positioning but may limit market penetration',
        action: 'Consider value-added features to justify premium'
      });
    }

    if (optimalPrice < market.averagePrice * 0.8) {
      recommendations.push({
        type: 'competitive_pricing',
        message: 'Competitive pricing may accelerate adoption',
        action: 'Ensure sufficient margin for growth investments'
      });
    }

    return recommendations;
  }
}
```

Dynamic Pricing Implementation

```
class DynamicPricing {
  constructor() {
    this.pricingRules = [
      this.volumeDiscountRule,
      this.loyaltyDiscountRule,
      this.seasonalPricingRule,
      this.marketPenetrationRule
    ];
  }

  async calculatePrice(userId, product, context) {
    let basePrice = product.basePrice;
    let finalPrice = basePrice;
    let appliedDiscounts = [];

    for (const rule of this.pricingRules) {
      const discount = await rule.call(this, userId, product, context);
      if (discount.applicable) {
        finalPrice *= (1 - discount.percentage);
        appliedDiscounts.push(discount);
      }
    }

    return {
      basePrice,
      finalPrice,
      totalDiscount: basePrice - finalPrice,
      appliedDiscounts,
      validUntil: new Date(Date.now() + 24 * 60 * 60 * 1000) // 24 hours
    };
  }

  volumeDiscountRule(userId, product, context) {
    const userVolume = context.userMetrics.monthlyUsage;
    const volumeTiers = [
      { threshold: 1000, discount: 0.05 },
      { threshold: 5000, discount: 0.10 },
      { threshold: 10000, discount: 0.15 },
      { threshold: 50000, discount: 0.20 }
    ];

    for (const tier of volumeTiers.reverse()) {
      if (userVolume >= tier.threshold) {
        return {
          applicable: true,
          percentage: tier.discount,
          reason: `Volume discount for ${userVolume} monthly usage`,
          type: 'volume'
        };
      }
    }

    return { applicable: false };
  }

  loyaltyDiscountRule(userId, product, context) {
    const customerTenure = context.userMetrics.tenureMonths;
    const loyaltyTiers = [
      { threshold: 12, discount: 0.05 },
      { threshold: 24, discount: 0.10 },
      { threshold: 36, discount: 0.15 }
    ];

    for (const tier of loyaltyTiers.reverse()) {
      if (customerTenure >= tier.threshold) {
        return {
          applicable: true,
          percentage: tier.discount,
          reason: `Loyalty discount for ${customerTenure} months tenure`,
          type: 'loyalty'
        };
      }
    }

    return { applicable: false };
  }
}
```

Revenue Projections

Financial Modeling

```
class RevenueProjections {
  constructor() {
    this.projectionPeriod = 36; // months
    this.growthAssumptions = {
      userGrowth: 0.15, // 15% monthly
      conversionRate: 0.05, // 5% free to paid
      churnRate: 0.03, // 3% monthly
      expansionRevenue: 0.20 // 20% from upgrades
    };
  }

  async generateProjections(startingMetrics) {
    const projections = [];
    let currentMetrics = { ...startingMetrics };

    for (let month = 1; month <= this.projectionPeriod; month++) {
      const monthlyProjection = this.calculateMonthlyProjection(currentMetrics, month);
      projections.push(monthlyProjection);
      currentMetrics = this.updateMetrics(currentMetrics, monthlyProjection);
    }

    return {
      monthly: projections,
      summary: this.generateSummary(projections),
      scenarios: this.generateScenarios(startingMetrics)
    };
  }

  calculateMonthlyProjection(metrics, month) {
    const newUsers = Math.floor(metrics.totalUsers * this.growthAssump-
tions.userGrowth);
    const newPaidUsers = Math.floor(newUsers * this.growthAssumptions.conversionRate);
    const churnedUsers = Math.floor(metrics.paidUsers * this.growthAssump-
tions.churnRate);
    const upgrades = Math.floor(metrics.paidUsers * 0.02); // 2% upgrade monthly

    const subscriptionRevenue = this.calculateSubscriptionRevenue(metrics);
    const servicesRevenue = this.calculateServicesRevenue(metrics);
    const marketplaceRevenue = this.calculateMarketplaceRevenue(metrics);
    const educationRevenue = this.calculateEducationRevenue(metrics);

    return {
      month,
      users: {
        total: metrics.totalUsers + newUsers,
        paid: metrics.paidUsers + newPaidUsers - churnedUsers + upgrades,
        new: newUsers,
        churned: churnedUsers
      },
      revenue: {
        subscription: subscriptionRevenue,
        services: servicesRevenue,
        marketplace: marketplaceRevenue,
        education: educationRevenue,
        total: subscriptionRevenue + servicesRevenue + marketplaceRevenue + education-
Revenue
      },
      metrics: {
        arpu: subscriptionRevenue / (metrics.paidUsers || 1),
        ltv: this.calculateLTV(metrics),
        cac: this.calculateCAC(metrics),
        churnRate: churnedUsers / (metrics.paidUsers || 1)
      }
    };
  }

  generateScenarios(startingMetrics) {
    const scenarios = {
      conservative: this.generateConservativeScenario(startingMetrics),
      optimistic: this.generateOptimisticScenario(startingMetrics),
      pessimistic: this.generatePessimisticScenario(startingMetrics)
    };

    return scenarios;
  }

  generateConservativeScenario(startingMetrics) {
    const conservativeAssumptions = {
      userGrowth: 0.08, // 8% monthly
      conversionRate: 0.03, // 3% free to paid
      churnRate: 0.05, // 5% monthly
      expansionRevenue: 0.10 // 10% from upgrades
    };

    return this.runProjection(startingMetrics, conservativeAssumptions);
  }

  generateOptimisticScenario(startingMetrics) {
    const optimisticAssumptions = {
      userGrowth: 0.25, // 25% monthly
      conversionRate: 0.08, // 8% free to paid
      churnRate: 0.02, // 2% monthly
      expansionRevenue: 0.30 // 30% from upgrades
    };

    return this.runProjection(startingMetrics, optimisticAssumptions);
  }
}
```

Revenue Targets

```yaml
revenue_targets:
  year_1:
    q1:
      subscription_revenue: "$50,000"
      services_revenue: "$25,000"
      marketplace_revenue: "$5,000"
      education_revenue: "$10,000"
      total: "$90,000"

    q2:
      subscription_revenue: "$150,000"
      services_revenue: "$75,000"
      marketplace_revenue: "$20,000"
      education_revenue: "$30,000"
      total: "$275,000"

    q3:
      subscription_revenue: "$350,000"
      services_revenue: "$150,000"
      marketplace_revenue: "$50,000"
      education_revenue: "$75,000"
      total: "$625,000"

    q4:
      subscription_revenue: "$600,000"
      services_revenue: "$250,000"
      marketplace_revenue: "$100,000"
      education_revenue: "$125,000"
      total: "$1,075,000"

  year_2:
    annual_target: "$5,000,000"
    breakdown:
      subscription: "70%"
      services: "20%"
      marketplace: "6%"
      education: "4%"

  year_3:
    annual_target: "$15,000,000"
    breakdown:
      subscription: "65%"
      services: "25%"
      marketplace: "7%"
      education: "3%"
```

Success Metrics

Key Performance Indicators

- **Monthly Recurring Revenue (MRR)**: Target $1M by end of Year 1
- **Annual Recurring Revenue (ARR)**: Target $12M by end of Year 1
- **Customer Acquisition Cost (CAC)**: Keep below $500

- **Customer Lifetime Value (LTV)**: Target $5,000+
- **LTV/CAC Ratio**: Maintain above 10:1
- **Gross Revenue Retention**: Target 95%+
- **Net Revenue Retention**: Target 120%+

Conversion Metrics

- **Free to Paid Conversion**: Target 5% within 90 days
- **Trial to Paid Conversion**: Target 25%
- **Upgrade Rate**: Target 15% annually
- **Churn Rate**: Keep below 3% monthly

Market Penetration

- **Market Share**: Target 10% of addressable market
- **Enterprise Penetration**: 100+ enterprise customers
- **Geographic Expansion**: 5+ regions with local presence
- **Partner Channel Revenue**: 30% of total revenue

Implementation Timeline

Phase 1: Foundation (Months 1-3)

- [ ] Implement subscription management system
- [ ] Launch Professional tier
- [ ] Set up payment processing and billing
- [ ] Create basic enterprise features

Phase 2: Growth (Months 4-6)

- [ ] Launch Enterprise tier
- [ ] Implement plugin marketplace
- [ ] Start professional services offering
- [ ] Launch certification programs

Phase 3: Scale (Months 7-9)

- [ ] Expand enterprise features
- [ ] Launch partner program
- [ ] Implement dynamic pricing
- [ ] Scale professional services

Phase 4: Optimization (Months 10-12)

- [ ] Optimize pricing based on data
- [ ] Expand international markets
- [ ] Launch advanced enterprise features
- [ ] Achieve profitability targets

This open-core monetization model provides a sustainable path to profitability while maintaining Audityzer’s commitment to open source and community-driven development.