Roadmap

PhishVision - December 19th Final Competition Roadmap

🎯 Executive Summary

You have 19 days (November 30 → December 19) to transform PhishVision from a hackathon MVP to a competition-winning final product.

Final Project MVP Requirements (from competition brief):

- ✅ Template builder (DONE - custom templates exist)
- ⚠️ SOC response timeline (NEEDS WORK)
- ✅ NLP analyzer (DONE - comprehensive)
- ⚠️ Full dashboard (NEEDS ENHANCEMENT)

Current State vs Competition Requirements

| Requirement | Current Status | Gap | Priority |

|---|---|---|---|

| Template builder | ✅ Custom templates + AI generation | Minor polish | LOW |

| SOC response timeline | ❌ Missing | Need to build | HIGH |

| NLP analyzer | ✅ 719 lines, comprehensive | Add ML model | MEDIUM |

| Full dashboard | ⚠️ Basic stats exist | Add SOC metrics | HIGH |

📊 Market Analysis Summary

Top Competitors

| Platform | Market Share | Pricing | Key Strength |

|----------|-------------|---------|--------------|

| KnowBe4 | 26.6% | $18-39/user/year | Largest template library |

| Proofpoint | 9.8% | $18+/user/year | Threat intelligence |

| Cofense | Enterprise | Custom | SOC integration |

| Gophish | Open Source | Free | Full customization |

| Barracuda | Mid-market | $14.40/user/year | Multi-vector (SMS, Voice) |

Market Trends 2024-2025

1. 82.6% of phishing emails are now AI-generated
2. 1,265% surge in AI-linked phishing attacks since ChatGPT
3. SOC integration is becoming mandatory for enterprise
4. Behavioral analytics is the new standard
5. Multi-vector attacks (SMS, Voice, USB) are growing

What Makes Winners Stand Out

1. AI-powered personalization (KnowBe4 Diamond)
2. SOC/SIEM integration (Cofense)
3. Behavioral risk scoring (Barracuda - 16,000+ data points)
4. Just-in-time training (All leaders)
5. Compliance reporting (SOC 2, ISO 27001)

🏗️ Development Roadmap (19 Days)

Week 1: Dec 1-7 - SOC Features (HIGH PRIORITY)

Day 1-2: SOC Response Timeline

What it is: A timeline showing security incidents and response actions

Why it matters: Direct competition requirement

Implementation:

```
1. Create new database model: SecurityIncident

- id, type, severity, description, created_at

- status: detected, investigating, contained, resolved

- response_time (calculated field)



2. Create backend route: /api/soc/

- GET /timeline - Get all incidents with timeline

- POST /incident - Log new incident

- PUT /incident/<id>/status - Update incident status

- GET /metrics - MTTR, detection rate, etc.



3. Create frontend component: SOCTimeline.js

- Visual timeline with Recharts

- Filter by date range, severity, status

- Real-time updates
```

Files to create/modify:

- `backend/models.py` - Add SecurityIncident model
- `backend/routes/soc_routes.py` - New file

- `frontend/src/components/SOCTimeline.js` - New file
- `frontend/src/components/Dashboard.js` - Add SOC section

Day 3-4: Enhanced Dashboard

What it is: Full SOC-style dashboard with comprehensive metrics

Why it matters: Competition requirement + industry standard

New Metrics to Add:

```
SOC Metrics:

├── Mean Time to Detect (MTTD)

├── Mean Time to Respond (MTTR)

├── Incident count by severity

├── Response efficiency score

└── Security posture score



Risk Metrics:

├── User risk scores (individual)

├── Department risk heatmap

├── Trend analysis (improving/declining)

└── Top 10 at-risk users
```

```
Campaign Metrics:

├── Success rate by template type

├── Time-to-click analysis

├── Repeat offender tracking

└── Training completion rates
```

New Visualizations:

- Risk heatmap by department
- Incident timeline chart
- User vulnerability scatter plot
- Response time trends

Day 5-7: User Risk Scoring System

What it is: Score each user 0-100 based on their phishing susceptibility

Why it matters: Industry standard, impressive for judges

Algorithm:

```
def calculate_user_risk_score(user_id):

# Get user's campaign history

targets =
CampaignTarget.query.filter_by(email=user_email).all()



total_received = len(targets)
```

```
total_opened = sum(1 for t in targets if t.opened_at)

total_clicked = sum(1 for t in targets if t.clicked_at)



if total_received == 0:

return 50 # Neutral score for new users



# Base score (higher = more risk)

click_rate = (total_clicked / total_received) * 100

open_rate = (total_opened / total_received) * 100



# Weighted score

risk_score = (click_rate * 0.7) + (open_rate * 0.3)



# Time decay - recent failures weight more

# Difficulty adjustment - hard templates = less penalty



return min(100, max(0, risk_score))
```

Frontend:

- User risk dashboard
- Individual user profiles with history
- Risk trend charts
- Recommendations for high-risk users

Week 2: Dec 8-14 - NLP Enhancement + Polish

Day 8-10: ML-Based Email Classification

What it is: Upgrade from regex to machine learning

Why it matters: Major differentiator, impressive technology

Implementation:

```
# Option 1: Simple ML with scikit-learn

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_extraction.text import TfidfVectorizer



class MLEmailClassifier:

def __init__(self):

self.vectorizer = TfidfVectorizer(max_features=5000)

self.classifier = RandomForestClassifier(n_estimators=100)



def train(self, emails, labels):
```

```
X = self.vectorizer.fit_transform(emails)

self.classifier.fit(X, labels)



def predict(self, email_text):

X = self.vectorizer.transform([email_text])

proba = self.classifier.predict_proba(X)[0]

return {

'safe': proba[0],

'suspicious': proba[1],

'malicious': proba[2]

}
```

Training Data Sources:

- Kaggle phishing email datasets
- Generate synthetic data from your templates
- Use existing analysis history

Integration:

- Keep NLP analyzer as fallback
- ML provides probability scores
- Ensemble both for best results

Day 11-12: Sentiment Analysis

What it is: Detect emotional manipulation in emails

Why it matters: Sophisticated detection, unique feature

Implementation:

```
# Use TextBlob or VADER for sentiment

from textblob import TextBlob



def analyze_emotional_manipulation(text):

blob = TextBlob(text)



# Overall sentiment

sentiment = blob.sentiment # polarity, subjectivity



# Detect urgency words

urgency_patterns = ['immediately', 'urgent', 'now',
'deadline']



# Detect fear words

fear_patterns = ['suspended', 'locked', 'unauthorized',
'breach']
```

```
# Detect greed words

greed_patterns = ['winner', 'prize', 'free', 'bonus',
'reward']



return {

'sentiment_score': sentiment.polarity,

'urgency_level': count_patterns(text, urgency_patterns),

'fear_manipulation': count_patterns(text, fear_patterns),

'greed_manipulation': count_patterns(text, greed_patterns),

'emotional_risk': calculate_emotional_risk(...)

}
```

Day 13-14: Just-in-Time Training

What it is: Immediate training popup when user clicks phishing link

Why it matters: Industry standard, proven effective

Current Flow:

```
User clicks link → Training page redirect → Generic message
```

Enhanced Flow:

```
User clicks link → Dynamic training page with:

- What they missed (specific red flags in that email)

- Interactive quiz (3 questions)

- Video micro-lesson (2 min)

- "I understand" confirmation

- Points/badges for completion
```

Implementation:

- Modify tracking_routes.py to pass campaign data to training page
- Create dynamic training page component
- Add quiz logic
- Track training completion in database

Week 3: Dec 15-18 - Final Polish + Demo Prep

Day 15-16: Template Builder Enhancement

What it is: Visual drag-and-drop template editor

Why it matters: Explicitly mentioned in requirements

Current: Text-based HTML input

Enhanced:

```
Visual Editor with:

├── Drag-and-drop blocks (header, body, button, image)
```

```
├── Pre-built components library
├── Live preview
├── Variable insertion ({{name}}, {{tracking_link}})
├── Difficulty auto-detection
└── Mobile preview
```

Libraries to use:

- React-DnD for drag-and-drop
- GrapesJS or Unlayer for visual editor

Day 17: Testing & Bug Fixes

- Full end-to-end testing
- Fix any broken features
- Performance optimization
- Mobile responsiveness check
- Cross-browser testing

Day 18: Demo Preparation

- Create compelling demo data
- Prepare 10 realistic campaigns
- Generate fake users with varied risk scores
- Set up impressive dashboard visualizations
- Write demo script (5-7 minutes)

Day 19 (Competition Day): Final Touches

- Last-minute fixes only
- Practice demo 3x
- Prepare backup plan (screenshots, video)

- Arrive early, test equipment

🎨 UI Enhancements Checklist

Dashboard Improvements

- [ ] Add SOC metrics section
- [ ] User risk heatmap
- [ ] Incident timeline widget
- [ ] Response time charts
- [ ] Risk trend sparklines
- [ ] Department comparison chart

New Pages to Add

- [ ] SOC Timeline page
- [ ] User Risk Profiles page
- [ ] Training Results page
- [ ] Compliance Reports page

Visual Polish

- [ ] Loading skeletons
- [ ] Better error states
- [ ] Empty state illustrations
- [ ] Micro-animations
- [ ] Toast notifications
- [ ] Confirmation dialogs (already have)

📁 Files to Create

Backend

```
backend/
```

```
├── routes/
│ └── soc_routes.py # NEW - SOC timeline & metrics
├── services/
│ ├── ml_classifier.py # NEW - ML email classification
│ ├── sentiment_analyzer.py # NEW - Emotional manipulation detection
│ └── risk_scorer.py # NEW - User risk scoring
└── models.py # UPDATE - Add SecurityIncident, UserRiskScore
```

Frontend

```
frontend/src/
├── components/
│ ├── SOCTimeline.js # NEW - SOC response timeline
│ ├── UserRiskDashboard.js # NEW - Risk scoring dashboard
│ ├── RiskHeatmap.js # NEW - Department risk visualization
│ ├── TrainingModule.js # NEW - Just-in-time training
│ └── TemplateBuilder.js # ENHANCE - Visual editor
└── pages/
```

```
├── SOCPage.js # NEW - SOC operations page

└── ReportsPage.js # NEW - Compliance reports
```

📊 Database Schema Updates

New Models

```
class SecurityIncident(db.Model):

__tablename__ = 'security_incidents'



id = db.Column(db.String(36), primary_key=True,
default=lambda: str(uuid.uuid4()))

type = db.Column(db.String(50)) # phishing_click,
credential_entered, malware_download

severity = db.Column(db.String(20)) # low, medium, high,
critical

description = db.Column(db.Text)

user_email = db.Column(db.String(200))

campaign_id = db.Column(db.String(36),
db.ForeignKey('campaigns.id'), nullable=True)



# Timeline tracking
```

```
detected_at = db.Column(db.DateTime, default=datetime.utcnow)

acknowledged_at = db.Column(db.DateTime, nullable=True)

contained_at = db.Column(db.DateTime, nullable=True)

resolved_at = db.Column(db.DateTime, nullable=True)



status = db.Column(db.String(20), default='detected') #
detected, investigating, contained, resolved

response_notes = db.Column(db.Text, nullable=True)




class UserRiskScore(db.Model):

__tablename__ = 'user_risk_scores'



id = db.Column(db.String(36), primary_key=True,
default=lambda: str(uuid.uuid4()))

email = db.Column(db.String(200), unique=True)

department = db.Column(db.String(100), nullable=True)



risk_score = db.Column(db.Float, default=50.0) # 0-100
```

```
campaigns_received = db.Column(db.Integer, default=0)

campaigns_opened = db.Column(db.Integer, default=0)

campaigns_clicked = db.Column(db.Integer, default=0)

training_completed = db.Column(db.Integer, default=0)



last_incident_at = db.Column(db.DateTime, nullable=True)

last_training_at = db.Column(db.DateTime, nullable=True)

score_updated_at = db.Column(db.DateTime,
default=datetime.utcnow)
```

🏆 Competition Winning Tips

What Judges Look For

1. **Does it solve the problem?** (Phishing simulation + SOC dashboard)
2. **Is it technically impressive?** (AI/ML, real-time, sophisticated)
3. **Is the demo compelling?** (Story, flow, "wow" moments)
4. **Is the code quality good?** (Clean, documented, tested)
5. **What's the market potential?** (Your research shows $620M market)

Demo Strategy

1. **Start with impact**: "82.6% of phishing is AI-generated. We built AI to fight AI."
2. **Show the SOC dashboard**: Real-time metrics, impressive visuals
3. **Run a live campaign**: Send phishing email, show tracking
4. **Demonstrate ML analysis**: Analyze suspicious email, show detection

5. **Show risk scoring**: "This user is 85% risk, here's why..."
6. **End with vision**: Multi-tenant, MSP market, enterprise integrations

Technical Demos to Prepare

1. Live email sending + tracking
2. NLP analyzer detecting malicious email
3. SOC timeline showing incident response
4. User risk dashboard with heatmaps
5. AI template generation

📅 Daily Task Breakdown

Week 1

Day	Date	Tasks	Hours
1	Dec 1	SecurityIncident model, SOC routes	6-8h
2	Dec 2	SOCTimeline.js component	6-8h
3	Dec 3	Dashboard SOC section	4-6h
4	Dec 4	User risk scoring backend	6-8h
5	Dec 5	Risk dashboard frontend	6-8h
6	Dec 6	Risk heatmap visualization	4-6h
7	Dec 7	Testing & bug fixes	4-6h

Week 2

Day	Date	Tasks	Hours
8	Dec 8	ML classifier setup	6-8h
9	Dec 9	ML training + integration	6-8h

| 10 | Dec 10 | ML testing + fallback | 4-6h |

| 11 | Dec 11 | Sentiment analysis | 4-6h |

| 12 | Dec 12 | Just-in-time training | 6-8h |

| 13 | Dec 13 | Training quiz/module | 6-8h |

| 14 | Dec 14 | Template builder enhancement | 6-8h |

Week 3

| Day | Date | Tasks | Hours |

|-----|------|-------|-------|

| 15 | Dec 15 | Template builder completion | 6-8h |

| 16 | Dec 16 | UI polish & animations | 4-6h |

| 17 | Dec 17 | Full testing | 4-6h |

| 18 | Dec 18 | Demo prep & practice | 4-6h |

| 19 | Dec 19 | COMPETITION DAY | - |

🚀 Quick Wins (Do First)

Immediate Impact, Low Effort

1. **Add 5 more templates** (2 hours) - Easy content win
2. **Add department field** to campaigns (1 hour) - Enables heatmap later
3. **Add loading states** everywhere (2 hours) - Polish
4. **Fix mobile responsiveness** (2 hours) - Professional look
5. **Add export buttons** (CSV/PDF) (3 hours) - Compliance feature

Commands to Run Today

```
# Backend dependencies for ML
```

```
pip install scikit-learn textblob pandas



# Frontend dependencies for visualizations

cd frontend

npm install recharts react-dnd
```

💡 Differentiators to Highlight

vs KnowBe4 ($18-39/user/year)

- "Open source alternative with AI"
- "Modern React UI vs clunky interface"
- "Multi-language support (AZ, TR, EN)"

vs Gophish (Free)

- "AI-powered template generation"
- "Built-in NLP analyzer"
- "SOC dashboard included"
- "No setup required"

vs Cofense (Enterprise)

- "SMB-friendly pricing potential"
- "Modern architecture (React + Flask)"
- "Easy deployment"

✅ Competition Checklist

Before Competition

- [ ] All features working
- [ ] Demo data prepared
- [ ] Demo script written
- [ ] Slides ready (if needed)
- [ ] Backup demo video recorded
- [ ] Test on presentation laptop

During Presentation

- [ ] Internet connection tested
- [ ] Backend running
- [ ] Frontend running
- [ ] Demo accounts ready
- [ ] Confident delivery

Questions to Prepare For

1. "How does your NLP work?" → Explain keyword detection + ML
2. "How do you track opens?" → 1×1 pixel tracking
3. "What's your tech stack?" → React + Flask + SQLite
4. "How does it scale?" → Easy to move to PostgreSQL + Redis
5. "What's the business model?" → Per-user SaaS pricing ($12-30/user)

📞 Support

If you need help during development:

- Review existing code in `TECHNICAL_GUIDE.md`
- Check `backend/services/` for implementation patterns
- Use AI template generator as reference for API integration

Good luck! You won the 24-hour hackathon - now go win the final! 🏆