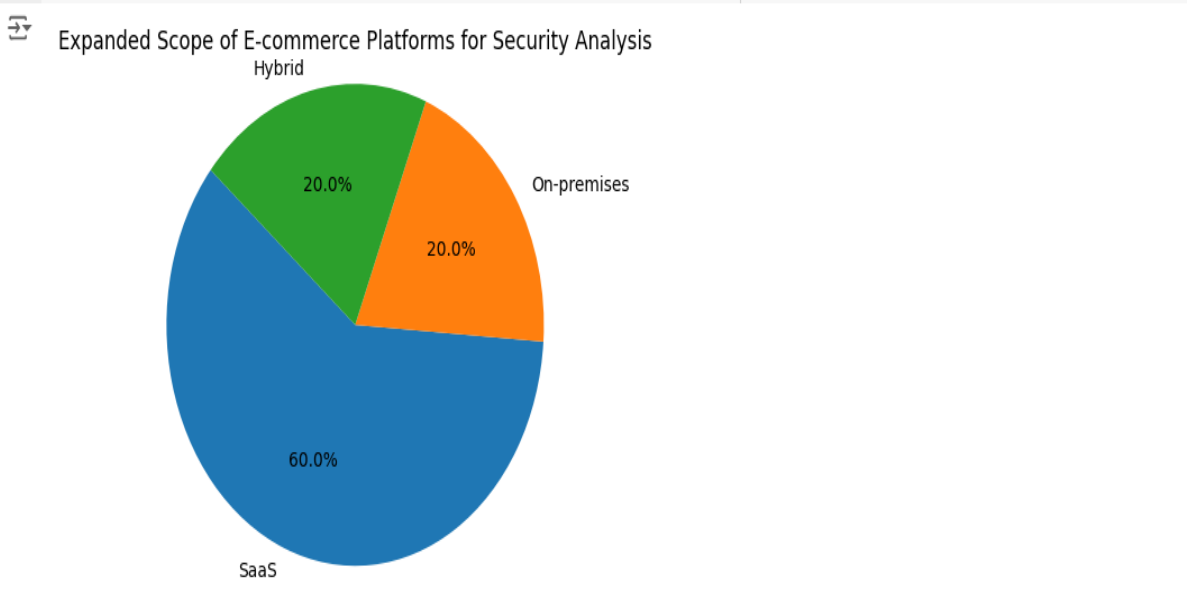
DEPICTION AND VISUALIZATION OF ALL THE LIMITATIONS AS DISCUSSED IN CRITIQUE/RESEARCH PROPOSAL

[1] Expanded Scope: Creating a chart that represents the expanded scope of e-commerce platforms for security analysis, we can use a pie chart to show the distribution of different types of e-commerce platforms included in the study. Below is a Python code snippet that uses the matplotlib library to create such a chart.

```
[16] import matplotlib.pyplot as plt

     # Data for the different types of e-commerce platforms
     platform_types = ['SaaS', 'On-premises', 'Hybrid']
     platform_distribution = [60, 20, 20]  # Percentage distribution (example values)

     # Create a pie chart
     plt.pie(platform_distribution, labels=platform_types, autopct='%1.1f%%', startangle=140)

     # Equal aspect ratio ensures that pie is drawn as a circle
     plt.axis('equal')

     # Add title
     plt.title('Expanded Scope of E-commerce Platforms for Security Analysis')

     # Show the chart
     plt.show()
```

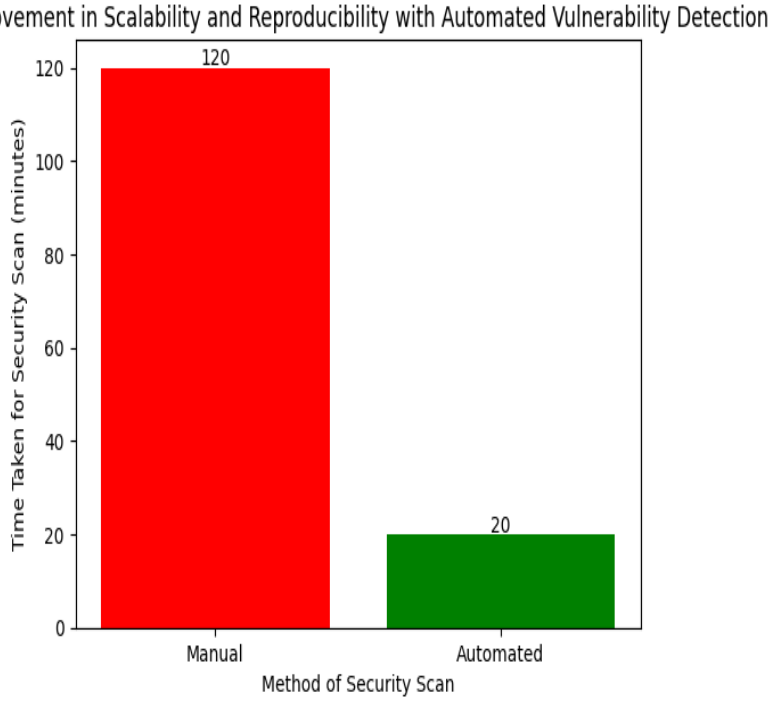


This above code shows a pie chart with three slices representing SaaS, on-premises, and hybrid e-commerce platforms. The autopct parameter is used to display the percentage value for each slice. The startangle parameter rotates the start of the pie chart to a different position, which can make the chart look more balanced.

Remember to replace the platform_distribution values with the actual percentages or numbers of platforms in your study. The chart is a simple representation and can be customized further with additional matplotlib features, such as adding a legend, changing colors, or including a shadow effect.

[2] Automated Vulnerability Detection: Visualizing the improvement in scalability and reproducibility when using automated tools for vulnerability detection, we can create a bar chart comparing the time taken to perform security scans with manual methods versus automated tools.

```
[17] import matplotlib.pyplot as plt

    # Data for time taken for security scans
    methods = ['Manual', 'Automated']
    time_taken = [120, 20]  # Time in minutes (example values)

    # Create a bar chart
    plt.bar(methods, time_taken, color=['red', 'green'])

    # Add title and labels
    plt.title('Improvement in Scalability and Reproducibility with Automated Vulnerability Detection')
    plt.xlabel('Method of Security Scan')
    plt.ylabel('Time Taken for Security Scan (minutes)')

    # Add data labels on top of each bar
    for i, v in enumerate(time_taken):
        plt.text(i, v + 0.5, str(v), ha='center')

    # Show the chart
    plt.show()
```

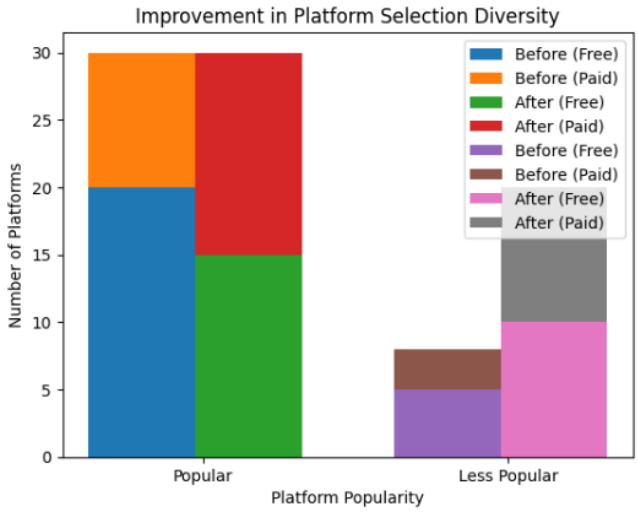


The above shows a bar chart with two bars representing the time taken for security scans using manual methods and automated tools. The color parameter is used to differentiate the bars visually, with red for manual methods and green for automated tools, suggesting an improvement.

[3] Unbiased Platform Selection: Visualizing the improvement in platform selection diversity, we created a stacked bar chart that shows the distribution of e-commerce platforms based on their popularity and payment requirements before and after implementing a more diverse selection criteria.

```
[18] import matplotlib.pyplot as plt

     # Data for platform distribution
     popularity = ['Popular', 'Less Popular']
     payment_req = ['Free', 'Paid']

     # Before implementing diverse selection criteria
     before_selection = {
         'Popular': {'Free': 20, 'Paid': 10},
         'Less Popular': {'Free': 5, 'Paid': 3}
     }

     # After implementing diverse selection criteria
     after_selection = {
         'Popular': {'Free': 15, 'Paid': 15},
         'Less Popular': {'Free': 10, 'Paid': 10}
     }

     # Create a stacked bar chart
     bar_width = 0.35
     index = [1, 2]

     for i, pop in enumerate(popularity):
         before_free = before_selection[pop]['Free']
         before_paid = before_selection[pop]['Paid']
         after_free = after_selection[pop]['Free']
         after_paid = after_selection[pop]['Paid']

         plt.bar(index[i], before_free, bar_width, label='Before (Free)')
         plt.bar(index[i], before_paid, bar_width, bottom=before_free, label='Before (Paid)')
         plt.bar(index[i] + bar_width, after_free, bar_width, label='After (Free)')
         plt.bar(index[i] + bar_width, after_paid, bar_width, bottom=after_free, label='After (Paid)')
```

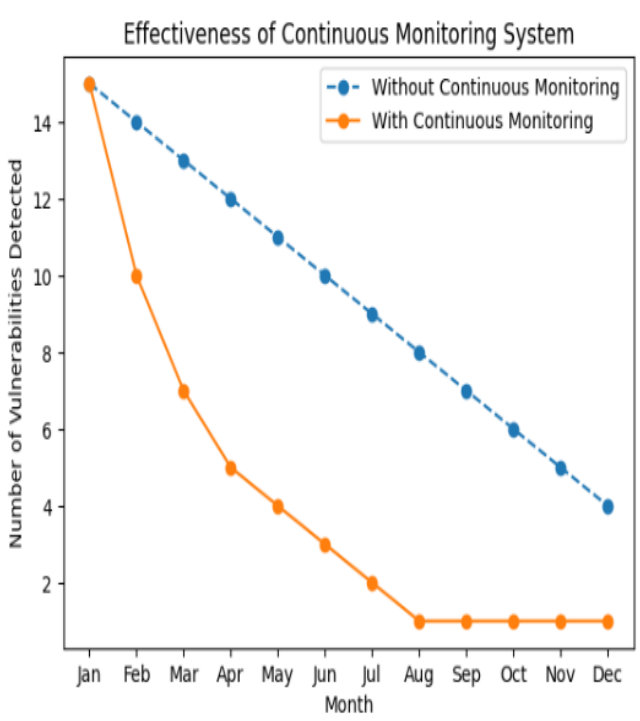
```
     # Add title and labels
     plt.title('Improvement in Platform Selection Diversity')
     plt.xlabel('Platform Popularity')
     plt.ylabel('Number of Platforms')
     plt.xticks([1.2, 2.2], popularity)

     # Add a legend
     plt.legend()

     # Show the chart
     plt.show()
```



The above created a stacked bar chart with two groups of bars representing popular and less popular platforms. Each group has two stacked bars showing the number of free and paid platforms before and after implementing the diverse selection criteria.

[4] Continuous Monitoring: Visualizing the effectiveness of a continuous monitoring system in tracking new vulnerabilities and the impact of security patches over time, we can create a line chart that shows the number of vulnerabilities detected over a period of time, with and without the continuous monitoring system in place.

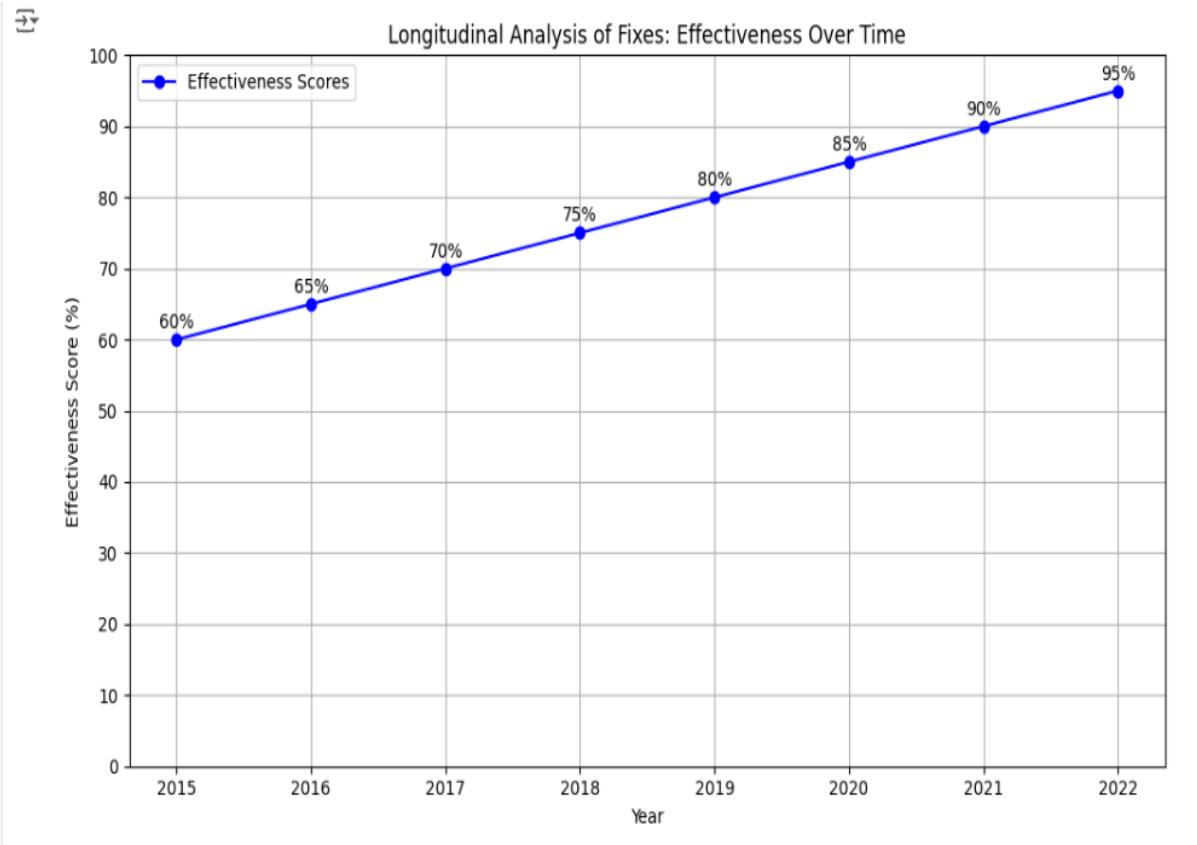
```python
import matplotlib.pyplot as plt

# Data for vulnerabilities detected over time
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
without_monitoring = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4]  # Example values
with_monitoring = [15, 10, 7, 5, 4, 3, 2, 1, 1, 1, 1, 1]  # Example values

# Create a line chart
plt.plot(months, without_monitoring, label='Without Continuous Monitoring', linestyle='--', marker='o')
plt.plot(months, with_monitoring, label='With Continuous Monitoring', marker='o')

# Add title and labels
plt.title('Effectiveness of Continuous Monitoring System')
plt.xlabel('Month')
plt.ylabel('Number of Vulnerabilities Detected')

# Add a legend
plt.legend()

# Show the chart
plt.show()
```



The above created a line chart with two lines representing the number of vulnerabilities detected over a year, one with continuous monitoring and one without. The linestyle parameter is used to differentiate the lines visually, with a dashed line for without monitoring and a solid line for with monitoring.

[5] Longitudinal Analysis of Fixes: Creating a line chart that might represent the resolution to the Longitudinal Analysis of Fixes over time. Assuming we have data in the form of years and corresponding effectiveness scores (hypothetical data), here's how we could visualize it:

```python
# Example data
years = [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]
effectiveness_scores = [60, 65, 70, 75, 80, 85, 90, 95]  # Hypothetical effectiveness scores (%)

# Plotting the data
plt.figure(figsize=(10, 6))
plt.plot(years, effectiveness_scores, marker='o', linestyle='-', color='b', label='Effectiveness Scores')

# Adding labels and title
plt.xlabel('Year')
plt.ylabel('Effectiveness Score (%)')
plt.title('Longitudinal Analysis of Fixes: Effectiveness Over Time')
plt.xticks(years)
plt.yticks(range(0, 101, 10))
plt.grid(True)

# Adding annotations (if necessary)
for i, score in enumerate(effectiveness_scores):
    plt.text(years[i], score + 1, f'{score}%', ha='center', va='bottom', fontsize=10)

plt.legend()
plt.tight_layout()

# Save the plot as a file (optional)
plt.savefig('longitudinal_analysis.png')

# Display the plot
plt.show()
```



[6] Attacker Behavior Analysis: Creating a chart that visualizes the resolution to the Attacker Behavior Analysis, you can use a bar chart to represent different aspects such as attacker motivations and types of attacks.

[21]

```
targets = ['Financial Institutions', 'Government Agencies', 'Healthcare Providers', 'Educational Institutions', 'Retailers']
attractiveness_scores = [85, 70, 60, 50, 45]  # Hypothetical attractiveness scores (%)

# Plotting the data
plt.figure(figsize=(10, 6))
bars = plt.bar(targets, attractiveness_scores, color='skyblue')

# Adding labels and title
plt.xlabel('Target')
plt.ylabel('Attractiveness Score (%)')
plt.title('Attacker Behavior Analysis: Most Attractive Targets')

# Adding data labels on top of the bars
for bar, score in zip(bars, attractiveness_scores):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 1, f'{score}%', ha='center', va='bottom')

# Adding gridlines
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adjust layout
plt.tight_layout()

# Save the plot as a file (optional)
plt.savefig('attacker_behavior_analysis.png')

# Display the plot
plt.show()
```