План реагирования на фишинговую атаку

1. Обнаружение

Цель: Выявить атаку, определить ее масштаб и собрать доказательства.

Действия:

1. **Анализ поступивших отчетов**:
   - Проверить жалобы сотрудников на подозрительные письма.
   - Использовать DLP-системы и SIEM для поиска подозрительной активности.
2. **Идентификация вредоносных ссылок и писем**:
   - Извлечь подозрительные URL из отчетов.
   - Использовать sandbox-аналитику для анализа содержимого.
3. **Мониторинг сетевого трафика и логов**:
   - Анализ DNS-запросов и HTTP-трафика.
   - Проверить журналы почтовых серверов (SMTP, IMAP, Exchange).
4. **Поиск компрометированных учетных записей**:
   - Анализ попыток входа с необычных IP-адресов.
   - Проверить использование украденных учетных данных.

2. Локализация

Цель: Ограничить воздействие атаки и минимизировать риски распространения.

Действия:

1. **Идентификация зараженных систем**:
   - Поиск рабочих станций, взаимодействовавших с вредоносными ссылками.
   - Изолирование потенциально скомпрометированных устройств.
2. **Ограничение доступа**:
   - Заблокировать вредоносные домены через прокси и файрвол.
   - Отключить скомпрометированные учетные записи и сбросить пароли.
3. **Изоляция почтовых серверов**:
   - Удалить вредоносные письма из почтовых ящиков сотрудников.
   - Временное ограничение исходящей почты, если есть подозрение на утечку.

3. Ликвидация

Цель: Устранить угрозу и восстановить безопасность.

Действия:

1. **Удаление вредоносных элементов**:
   - Очистка зараженных рабочих станций (антивирус, EDR-решения).
   - Удаление вредоносных макросов, скриптов, вложений.
2. **Обновление политик безопасности**:
   - Настроить фильтрацию входящих писем (DMARC, DKIM, SPF).

- Ужесточить политики паролей и внедрить MFA.
3. **Повторный аудит учетных записей**:
   - Пересмотр прав доступа для сотрудников.
   - Отслеживание подозрительной активности после сброса паролей.

4. Восстановление

Цель: Восстановить нормальную работу и предотвратить повторные атаки.

Действия:

1. **Обучение сотрудников**:
   - Проведение тренингов по кибер-гигиене.
   - Имитация фишинговых атак для тестирования осведомленности.
2. **Усиление мониторинга**:
   - Внедрение SOC для постоянного наблюдения.
   - Настройка SIEM для автоматического выявления аномалий.
3. **Разбор инцидента**:
   - Проведение ретроспективного анализа.
   - Корректировка стратегии реагирования на будущие атаки.

Вывод

Фишинговая атака была обнаружена и нейтрализована. Внедрены дополнительные меры защиты, предотвращающие повторные инциденты.

сценарий на Python

Вот сценарий на Python, который автоматизирует анализ журналов, выявление подозрительных попыток входа и блокировку вредоносных IP-адресов.

```
import re

import json

import requests

from datetime import datetime



# Настройки

LOG_FILE = "auth.log"  # Файл журнала аутентификации
```

```
THREAT_INTEL_API = "https://api.abuseipdb.com/api/v2/check"  # API для проверки IP

API_KEY = "YOUR_API_KEY"  # Замените на ваш ключ

BLOCKLIST_FILE = "blocklist.txt"  # Файл для блокировки IP


# Фильтр подозрительных попыток входа

FAILED_LOGIN_PATTERN = re.compile(r"Failed password for (invalid user )?(\w+) from ([\d\.]+) port")




def parse_logs(log_file):

    suspicious_ips = {}

    with open(log_file, "r") as file:

        for line in file:

            match = FAILED_LOGIN_PATTERN.search(line)

            if match:

                username = match.group(2)

                ip = match.group(3)

                if ip not in suspicious_ips:

                    suspicious_ips[ip] = {"count": 0, "users": set()}

                suspicious_ips[ip]["count"] += 1

                suspicious_ips[ip]["users"].add(username)

    return suspicious_ips




def check_threat_intel(ip):

    headers = {
```

```
        "Key": API_KEY,
        "Accept": "application/json"
    }
    params = {"ipAddress": ip, "maxAgeInDays": 30}
    response = requests.get(THREAT_INTEL_API, headers=headers, params=params)
    if response.status_code == 200:
        data = response.json()
        return data["data"].get("abuseConfidenceScore", 0)  # Уровень угрозы
    return 0



def block_ip(ip):
    with open(BLOCKLIST_FILE, "a") as file:
        file.write(ip + "\n")
    print(f"IP {ip} добавлен в список блокировки")



def simulate_incidents():
    test_logs = [
        "Failed password for root from 192.168.1.10 port 22",
        "Failed password for admin from 203.0.113.45 port 22",
        "Failed password for invalid user guest from 185.199.110.153 port 22",
        "Failed password for user from 192.168.1.15 port 22",
        "Failed password for invalid user test from 203.0.113.45 port 22",
        "Failed password for root from 185.199.110.153 port 22"
    ]
```

```
    with open(LOG_FILE, "w") as file:
        for log in test_logs:
            file.write(log + "\n")


if __name__ == "__main__":
    simulate_incidents()  # Генерация тестовых данных
    logs = parse_logs(LOG_FILE)
    for ip, info in logs.items():
        if info["count"] > 2:  # Порог срабатывания
            threat_score = check_threat_intel(ip)
            print(f"Проверка IP {ip}: {threat_score}")
            if threat_score > 50:  # Порог угрозы
                block_ip(ip)
```

Этот скрипт анализирует журналы на предмет неудачных попыток входа, проверяет IP-адреса на сервисе AbuseIPDB и блокирует их, если уровень угрозы превышает установленный порог. Его можно расширить интеграцией с SIEM или файрволами для автоматического предотвращения атак.

проверка сценария автоматического реагирования на инциденты.

```
import re
import json
import requests
```

```
from datetime import datetime


# Настройки

LOG_FILE = "auth.log"  # Файл журнала аутентификации

THREAT_INTEL_API = "https://api.abuseipdb.com/api/v2/check"  # API для проверки IP

API_KEY = "YOUR_API_KEY"  # Замените на ваш ключ

BLOCKLIST_FILE = "blocklist.txt"  # Файл для блокировки IP


# Фильтр подозрительных попыток входа

FAILED_LOGIN_PATTERN = re.compile(r"Failed password for (invalid user )?(\w+) from ([\d\.]+) port")



def parse_logs(log_file):
    suspicious_ips = {}
    with open(log_file, "r") as file:
        for line in file:
            match = FAILED_LOGIN_PATTERN.search(line)
            if match:
                username = match.group(2)
                ip = match.group(3)
                if ip not in suspicious_ips:
                    suspicious_ips[ip] = {"count": 0, "users": set()}
                suspicious_ips[ip]["count"] += 1
                suspicious_ips[ip]["users"].add(username)
    return suspicious_ips
```

```
def check_threat_intel(ip):
    headers = {
        "Key": API_KEY,
        "Accept": "application/json"
    }
    params = {"ipAddress": ip, "maxAgeInDays": 30}
    response = requests.get(THREAT_INTEL_API, headers=headers, params=params)
    if response.status_code == 200:
        data = response.json()
        return data["data"].get("abuseConfidenceScore", 0)  # Уровень угрозы
    return 0



def block_ip(ip):
    with open(BLOCKLIST_FILE, "a") as file:
        file.write(ip + "\n")
    print(f"IP {ip} добавлен в список блокировки")



def simulate_incidents():
    test_logs = [
        "Failed password for root from 192.168.1.10 port 22",
        "Failed password for admin from 203.0.113.45 port 22",
        "Failed password for invalid user guest from 185.199.110.153 port 22",
```

```
        "Failed password for user from 192.168.1.15 port 22",
        "Failed password for invalid user test from 203.0.113.45 port 22",
        "Failed password for root from 185.199.110.153 port 22"
    ]

    with open(LOG_FILE, "w") as file:
        for log in test_logs:
            file.write(log + "\n")


if __name__ == "__main__":
    simulate_incidents()  # Генерация тестовых данных
    logs = parse_logs(LOG_FILE)
    for ip, info in logs.items():
        if info["count"] > 2:  # Порог срабатывания
            threat_score = check_threat_intel(ip)
            print(f"Проверка IP {ip}: {threat_score}")
            if threat_score > 50:  # Порог угрозы
                block_ip(ip)
```