3.33 排他制御に関する教訓（T33）

教訓 T33

入念な方式設計と多段階の確認は当たり前、個人情報を扱う場合には特に排他制御に気をつけて

問題

イベント運営会社Ｉは、顧客からのイベント参加予約を複数の代理店を介してネットで受け付け、イベント管理システムによりその情報を一元的に処理している。各代理店は、それぞれの登録された顧客からのイベント予約を仲介している。（図 3.33－1 参照）

ある日、代理店Ａの担当者が、予約が締め切られたばかりのイベントの予約者一覧リストをＩ社のイベント管理システムからダウンロードしたところ、リストに同代理店の顧客以外の情報が含まれていた。同じ頃、代理店Ｂの担当者も同様の作業を行ったところ、同代理店の顧客以外の情報がダウンロードしたリストに含まれていた。それらの情報には、個人情報として扱われるものも含まれていた。

不審に思った両代理店の担当者は、前後してイベント管理会社Ｉのコールセンターにその状況を伝えた。同時期に２つの代理店からダウンロード情報の異常を伝えられたＩ社のシステム担当は、情報漏えい（セキュリティ・インシデント）が発生したと認識し、同社の情報セキュリティ規定に基づく対応を開始した。

まず、イベント管理システムのダウンロード機能を即座に停止した後、発生原因の調査を開始した。また、同様の事象が過去に発生していた可能性がないか、システムログの解析を行った。並行して、同社幹部が両代理店に出向き、今回の件について謝罪するとともに、ダウンロードしたリスト情報の廃棄を依頼し、その実施を確認した。その後、Ｉ社は、規定に基づき、今回のインシデントの発生をホームページで公表した。

幸い、過去に同様の事象が発生していた形跡は確認できなかった。また、発生原因が突き止められ、改修の後、発生から３日後にダウンロード機能の提供を再開した。

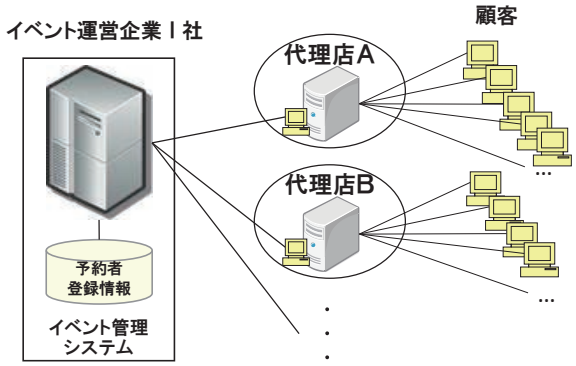

図 3.33－1　システムの全体構成概要

原因

(1) 直接原因

本セキュリティ・インシデントの直接原因は、I社のイベント管理システムのソフトウェアの不具合であった。イベント予約者一覧リストのダウンロード処理において、リスト情報転送用の一時ファイルの競合対策が不十分であった。具体的には、次のように処理が行われていた。（図 3.33－2 参照）

代理店からのリストのダウンロード要求に対し、一旦、ダウンロードデータを一時ファイルに書き込んだ後、所定の契機で一時ファイルの内容を要求元に転送する処理になっていた。ところが、複数の代理店から同タイミングで要求されるケースへの対策が不十分であり、要求の競合を考慮せずに一つの一時ファイルを排他制御無しで使用するようになっていた。そのため、ほぼ同時にダウンロード要求のあった 2 つの代理店の予約者情報が、同じ一時ファイルに混在して書き込まれてしまった。その後、その内容がそのまま両代理店に転送された。

なお、I社のイベント管理システムは、5 年以上運用していたが、今回のような問題が発生したのは初めてであった。

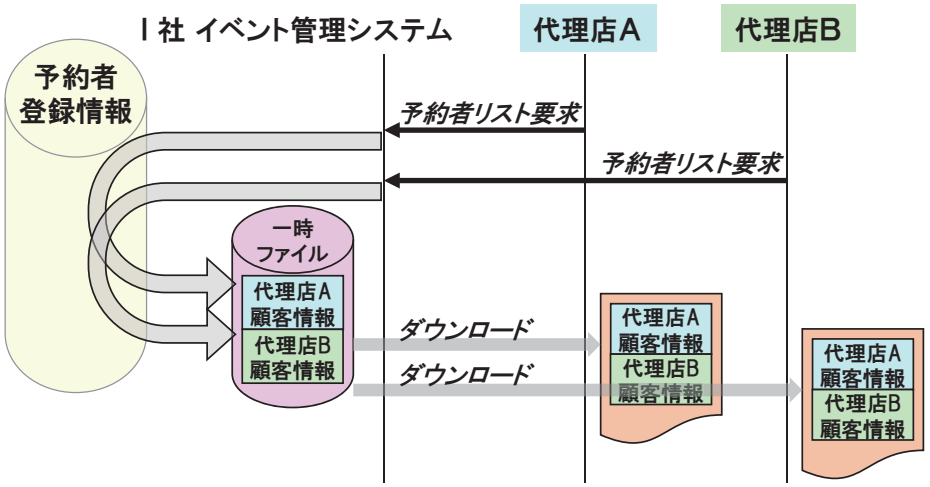

図 3.33－2　問題となった処理の流れ概要

(2) 不具合の混入原因

一時ファイルの競合対策が不十分であった原因を分析するため、関係者にヒアリングを行った。その結果、システム開発当時の当該処理の設計担当者には、自身のそれまでのシステム開発経験に基づく思い込みがあり、それが今回の問題を引き起こす要因となったことが判明した。

その担当者のそれまでの設計経験では、オペレーティングシステム（OS）あるいはミドルウェアがリソース競合対策の機能を有していたので、排他制御の機能をアプリケーションレベルで実装する必要はなかった。今回の対象システムの動作環境では彼の経験とは異なり、排他制御のための処理を担当プログラムに組み込む必要があったが、上記の理由により、それを自ら作り込むということには思い

が至らなかった。

また、当時の開発チーム編成時には、開発対象システムの内容や開発プロセス・環境に照らした、各担当者のスキルや経験についての細かな確認が行われていなかったことも分かった。

(3) 不具合が開発時に摘出できなかった理由

システム設計時には、第一に、リソース競合対策（排他制御）を始めとする処理方式等が慎重に考慮されるべきである。個人情報を取り扱う場合には特に、細心の注意が必要である。次に、たとえ設計時にリソース競合対策（排他制御）を適切に実装していなくとも、その後のレビューやテストにおいて、そのことは発見されるべきである。それが今回の問題発生まで発見されなかった理由を追求するため、I社では、システム開発時の膨大な記録を分析した。その結果、以下の（a）～（d）に示すような問題点が判明した。

（a）I社のシステム開発標準における設計指針には、リソース競合対策に関する記載がなかった。

過去の経験から、一時ファイルの競合対策（排他制御）を担当プログラムに実装することに思いが及ばなかった設計担当者は、設計指針によっても、その必要性に気づかされることはなかった。

リソース競合対策は、情報処理システムにおける常識ではあるものの、念のための注意喚起の意味から、少なくとも項目だけでも設計指針の中に記載されるべきである。また、リソース競合対策を含む方式設計を入念に行った上で、アプリケーション設計に進むべきである。

（b）I社のシステム開発標準プロセスでは、設計指針／コーディング指針と設計レビュー用チェック指針／コードレビュー用チェック指針とは、それぞれ全く同じものを兼用し、指針以外の観点でのレビューはなされなかった。

競合対策が不十分な設計がなされたが、設計指針にはしたがっていたため、設計レビューで不備が指摘されることはなかった。コーディングについても、指針にしたがって設計書の内容をそのままコード化したため、コードレビューで不備が指摘されることはなかった。

設計者／プログラマは、設計指針／コーディング指針に基づいて作業する。その作業結果に対し、設計レビュー用チェック指針／コードレビュー用チェック指針に基づいてレビューされるわけであるが、設計指針と設計レビュー用チェック指針とが全く同じものであると、設計指針に基づいて忠実に設計する限り、レビューにおいて指摘されることはあり得ない。コーディングについても同様である。レビューでは、設計指針／コーディング指針にしたがっているかという観点に加え、より多様な観点からチェックされるべきである。

（c）I社のテスト項目抽出基準は、設計書の内容のみに基づいてテスト項目を導くものであった。

設計書には共有リソース競合対策（排他制御）に関する記載が一切なかったので、その処理に対応するテスト項目が抽出されることはなかった。今回問題となった事象が発生するのは、複数の代理店がほぼ同時にダウンロード要求を行った場合である。この場合には、システム実装上の共有リソースの競合が発生し得る。通常、設計書の記載とは関係なく、このようなシナリオや

ユースケースを想定して共有リソース競合対策に問題がないかを確認するテストを行う。すなわち、「複数代理店からランダムに要求が来るということは、リソースの競合が起き得るな」ということを想定するはずであり、そう想定したならば、リソース競合対策の妥当性を確認するためのテスト項目を設定するはずである。今回は、そのテストが実施されなかった。また、テスト項目を含むテスト仕様書に対するレビューは行われなかった。

なお、このような微妙なタイミングに絡むケースのテストを実地で行うことは一般に困難であるが、疑似的な環境で負荷テストを長時間実施することにより、発生確率を高めることはできる。

(d) 対象システムの開発においては、第三者によるレビューが実施されていなかった。

レビューは開発チーム内で行われ、その主な観点は、「上位設計書の通りに実装されているか?」てあった。そのため、今回は上位設計書に誤りがあったが、それが摘出されなかった。レビュー者のうち一人でも、複数のダウンロード要求を同時に扱う処理のレビューを行っているということを意識していれば、その処理における共有リソースの有無やその競合対策について思いが及んだものと思われる。

一般に、知見や経験の豊富な第三者やシステム運用担当者などが、それらに基づいてチェックすることにより、誤りの摘出率は高くなる。レビュー対象がどのようなシナリオ（ユースケース）に対応する処理かということを意識しさえすれば，それに対応した注意事項や関連した過去のトラブル事例に思い当たることは間違いない。

(4) 保守時での不具合発見の機会逸失（セキュリティの観点）

I社のイベント管理システムでは、その初期開発時には、代理店によるイベント予約者一覧リストのダウンロード機能において個人情報を扱ってはいなかった。ところが初期リリースから数年後に、当該機能を更新していた。このときの更新内容は、ダウンロード対象項目の拡張であり、ここで初めて、ダウンロード情報にいわゆる個人情報が含まれることとなり、セキュリティに対する要求レベルが上がった。にもかかわらず、I社では十分な影響確認が行われなかった。もし、この時点で影響確認がセキュリティ要求レベルにふさわしい内容で実施され、それによってソフトウェアの不具合が発見されていれば、今回のインシデントは発生しなかった。

実際には、単なる情報項目の追加に対して、一般に、それを扱う処理のロジックは影響されないことから、当該システムにおいて、この機能更新時には十分な確認が行われなかった。もし、「セキュリティ（個人情報の取扱い）」という点を重視し、更新対象の情報項目にかかわる処理パスを方式設計にまで遡ってすべて徹底的にチェックしていれば、予約者一覧リスト情報を一時ファイルに書き込む処理の妥当性確認も確認対象となったはずであり、あるいは、競合対策漏れの不具合を発見できたかもしれない。

なお、I社のシステム開発標準には、セキュリティにかかわる処理に関し、このようなチェックを行う具体的な規定はなかった。

対策

直接原因への対応として、I社は早急に、次の不具合修正を行った。

(1)　イベント管理システムのソフトウェアの不具合である、イベント予約者一覧リストのダウンロード処理における一時ファイルの競合対策処理（排他制御）を正しく修正

また、根本原因への対策として、I社は、全般的に、セキュリティ（特に、個人情報）を扱う場合には入念な設計・確認が必要な旨をシステム開発標準に明記した。その上で、各原因に対応し、システム開発標準を次のように改訂した。

(2)　開発対象システムの内容や開発プロセス・環境に照らして、開発チームメンバ候補のスキルや経験について確認し、全体として必要条件をカバーできるようチームを編成

(3a)　設計指針、及びコーディング指針に、リソース競合対策（排他制御）に関する事項を追記

(3b)　設計／コーディングレビュー用チェック指針に、過去のトラブルに基づく確認項目を追加

(3c)　テスト項目リストもレビュー対象に追加

(3d)　一定規模以上のシステムや重要度の高いシステムにおいては、知見・経験の豊かな第三者によるレビューを義務化

(4)　セキュリティ（特に、個人情報）にかかわる更新時には、当該システムの重要性が一定以上の場合に、更新対象のセキュリティにかかわる部分に関連する全処理の再トレースに基づく確認を義務化

以上の措置について、今回のインシデントの概要とともに、I社内の関係者に周知した。

効果

I社のイベント管理システムでは、対策実施後、セキュリティ・インシデントは発生していない。

また、システム開発標準の改訂後、I社の開発するシステムに同種のインシデントは発生していない。

なお、I社では、改訂したシステム開発標準に基づき、システム開発チームの編成時に、メンバの経験やスキルにより注意深く配慮するようになった。また、個人情報を取り扱う場合には、リソースの競合対策（排他制御）等について開発の各段階で確実にチェックする習慣ができていった。例えばレビューに関しては、どのようなシナリオやユースケースに対応する処理を確認しているのかを考慮するなど、レビュー対象範囲のみに集中するのではなく、より広い視点でレビューを行う傾向が高まっていった。

教訓

システムの開発段階で混入した不具合は、システム障害を引き起こしたり、出力の誤りという形で現れたりすることが多い。本教訓における問題は、個人情報の漏えいというセキュリティ・インシデントを招いたものである。最近では、個人情報の漏えいは社会的影響が非常に大きくなっており、その

十分な対策が求められている。したがって、個人情報を扱うシステムでは、その処理で重要な排他制御等について、開発の初期段階から保守段階に至るまで、その入念な設計・確認が必要である。

今回のケースでは、不具合の混入は担当者の思い込みによるものであるが、それがずっと摘出されず、システムのサービス開始以降何年も経過した後で初めてセキュリティ・インシデントという形で発現した。その理由としては、開発チーム編成や方式設計、レビュー方法、テスト項目等、様々な段階での要因が考えられる。

まず、開発の早期で入念な方式設計を行うことが最も重要である。特に、排他制御はセキュリティ対策の要の一つであり、動作環境を踏まえた慎重な設計が求められる。

不幸にしてこのような検討をすり抜けて混入した不具合は、開発チームあるいは第三者による多様な視点からのレビューにより、システムのリリース前に摘出することができる。特に、微妙なタイミングに絡むような実地テストが困難な処理については、レビューが欠かせない。第三者によるレビューが効果的であるが、コスト面の制約から小規模なシステムでは開発チーム内でレビューすることもある。多様な視点でのレビューを行うには、レビュアの知見や経験、関心や専門等が多様であることが重要である。すなわち、チーム編成やレビュアの選定がポイントの一つとなる。

特に、個人情報を取り扱うなどセキュリティがクリティカルなシステムにおいては、より慎重な確認が求められる。そのためには、方式設計時やレビュー時にセキュリティに着目したシナリオやユースケースを設定する必要がある。また、テスト項目に関する適切な抽出、レビューも重要かつ効果的である。

そして、これらを踏まえ、設計やレビュー等の関係者の拠り所となるシステム開発標準を適切に整備・保守することが重要である。