Anatomy of a Logic Flaw

Presented by:
David Byrne
Managing Consultant
dbyrne@trustwave.com
Vulnerabilities

- "Traditional" Vulnerabilities
  - Standardized definitions
  - Security requirements common to all applications

- "Logic" Flaws
  - Violations of business rules; may be rules unique to a company or industry

- All vulnerabilities are violations of security rules
SQL Injection

• Requirement:
  Do not allow users to execute arbitrary SQL commands

• Vulnerability:
  Users can execute arbitrary SQL commands
Authentication Bypass

• **Requirement:**
  Verify a user’s identity before allowing access to the application

• **Vulnerability:**
  Access can be obtained without proving identity
Cross-Site Scripting

• **Requirement:**
  Do not allow users to define browser-side scripts

• **Vulnerability:**
  Users can define browser-side scripts
Vulnerabilities

• "Traditional" Vulnerabilities
  • Standardized definitions
  • Security requirements common to all applications

• "Logic" Flaws
  • Violations of business rule
  • Rules are often unique to a company, industry, or type of application

• All vulnerabilities are violations of security rules
Payment Bypass

• Requirement:
  Customers must pay for goods & services

• Vulnerability:
  Customers are not required to pay for goods & services
Client-Side Price Fixing

• Requirement: Only the business can set the price of goods

• Vulnerability: Customers can set the price of goods
Vulnerabilities

• "Traditional" Vulnerabilities
  • Standardized definitions
  • Security requirements common to all applications

• "Logic" Flaws
  • Violations of business rule
  • Rules are often unique to a company, industry, or type of application

• All vulnerabilities are failure to enforce rules
Root Causes of Logic Flaws

- Failure to anticipate threats
- Insufficient documentation of business rules
- Poor design practices (no SDLC)
- Poor understanding of underlying technologies
- Bad production management
Examples

• All real world examples

• Most are from real Trustwave tests, but client identity is well protected

• These are not rare flaws; we find them on a regular basis
Two factor – one factor = one factor
Two factor – one factor = one factor

1. To access the URL on the location line of your browser: https://
You can also link to from the home page (www."
com) from the Account Login menu.

2. Enter your user name and password, including hyphens, periods, and capitalization. Your password consists of a 5-8 digit Personal Identification Number (PIN) followed by the 6-digit SecurID Number:

User Name: Password (PIN + SecurID No.)

[SecurID No.]

= 543604
Two factor – one factor = one factor

by the 6-digit SecurID Number:

User Name: Password (PIN + SecurID No.)

[SecurID No.]
Two factor – one factor = one factor

Root causes:
• Insufficient documentation of business rules
• Poor understanding of underlying technologies

History
• Conflicting business priorities: customer security vs. customer convenience
• Someone said "use two factor"

Prevention
• Better documentation
• Security interests being represented throughout the process
Account Manipulation

Consider a banking application...

• **Functional requirement to allow wire transfers**
  • Only allow transfers between accounts the logged in user owns

• **Banking application generated a drop-down list of accounts to transfer funds to and from**

• **User selects the accounts to transfer from and to and clicks the "transfer" button**

• **The form details were submitted as a post parameters for the server-side to process**
Account Manipulation

Root cause:
- Failure to anticipate threats
- Poor understanding of technology

History
- This was the initial roll-out of the application developed by a third-party. Unfortunately, it was basically a case of the development team being unaware of secure coding techniques.

Prevention
- Understand how the technology works
Complex Price Manipulation

eyDigJ1pdGVTijogeyAidGl0bGUioiDigJ1IYWNraW5nIGZvciBE
dW1taWVzIiwgYiI9yIijogeyAidGl0bGUioiAiUyIsIO
KAnUNodWNrIEhlbmRlc3NvbiI6IHsgIkdsb3NzRW50cnkiOiB7
ICJJRCI6ICJTR01MIiwgIlNvcnRBcyI6ICJTR01MIiwgIkFjcm
9ueW0iOiAiU0dNTCI6IOKAnVByaWNlIjog4oCdMTU4NeKAnX0g
fSB9IH0gIAoK

{ "item": { "title": "Hacking for Dummies", "Author": { "name": "S", "Chuck Henderson": { "GlossEntry": { "ID": "SGML", "SortAs": "SGML", "Acronym": "SGML", "Price": "1585" } } } }
Complex Price Manipulation

Root cause:
• Poor understanding of underlying technologies

History
• This was an otherwise secure application
• The application framework obscured what data was sent to the client

Prevention
• Avoid niche application frameworks
• Popular frameworks have better documentation
• If a niche product is needed, dig into its internals
Private Performances

• Online theater seat reservation system

• Put seats into a cart, then checkout later

• Once seats are in a cart, they are held so that seats are not overbooked

• Using multiple browsers
  1. Put the seats you want into a cart
  2. Put the remaining open seats into a the second cart
  3. Complete the checkout of the first cart
  4. Never complete the checkout of the second cart.
Private Performances

Root causes:
• Failure to anticipate threats
• Poorly documented business rules
• Poor design practices

History
• Likely similar to the earlier examples of programmers used to private applications

Prevention
• A lot
Eat for (almost) Free

• Online system to place restaurant orders for delivery

• **Standard online order process**
  1. You select your meal
  2. Enter your address
  3. Pay your bill
  4. Food arrives

• **A third party handled the credit card transaction**
  • Redirected to a third party to handle the credit card purchase
  • Redirected back to the primary site after approval
Eat for (almost) Free

Minha Bandeja

<table>
<thead>
<tr>
<th>Valor do pedido:</th>
<th>3.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taxa de entrega:</td>
<td>6.00</td>
</tr>
<tr>
<td>Total do pedido:</td>
<td>9.50</td>
</tr>
</tbody>
</table>

| QTD. PROD | VLR |

My Tray

| Order value: | 3.50 |
| Delivery Rate: | 6.00 |
| Total Order: | 9.50 |

| QTD. PROD | VLR |

Receipt:

- **COO do documento vinculado:**
  - Valor da compra R$:
  - Valor do pagamento R$:
  - **REDECARD** 51******** A:064254
  - **VALOR:** 0.50
Eat for (almost) Free

Root causes:

• Insufficient documentation of business rules: The restaurant’s novice developers assumed that the processor was providing a secure service.
• Failure to anticipate threats: User tampering should always be prevented

History

• The payment processor did not provide a way to detect user tampering

Prevention

• Clearly define security responsibilities when integrating with a third party.
• Detect user tampering with cryptographic signing
Static Entropy

- Effective random number generation relies on a strong entropy source

```csharp
using System;
public class RandomGenerator
{
    Random random = new Random(3212351);
    public int getNext()
    {
        return random.Next();
    }
}
```
Static Entropy

Wicca vocalist, Indonesia razor
(in Freeport Roche of admits youngsters
Amsworth right, rerunning not
the googing

Wicca vocalist, Indonesia razor
(in Freeport Roche of admits youngsters
Amsworth right, rerunning not
the googing

Wicca vocalist, Indonesia razor
(in Freeport Roche of admits youngsters
Amsworth right, rerunning not
the googing
Static Entropy

Root causes:
• Poor understanding of underlying technologies

History
• The developers didn’t understand how random number generators worked

Prevention
• Educate developers
When Queries Collide

Welcome to Online Patient History
From this site you can access any of the resources available via the links to the left.

Patient Payment History

Patient Number: [ ]
(16-digits)

Patient Last Name: [ ]

Submit
When Queries Collide
When Queries Collide

Online Patient History

Patient Name Lookup Result

<table>
<thead>
<tr>
<th>Patient Name</th>
<th>Patient Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael Pachetti</td>
<td>2451 2497 3844 8854</td>
</tr>
</tbody>
</table>

- Home
- Preferences
- Patient Query
- Billing History
- Billing Help
- Support
- Logout
When Queries Collide

Welcome to Online Patient History
From this site you can access any of the resources available via the links to the left.

Patient Payment History
Patient Number: 2451 2497 3844 8854 (16-digits)
Patient Last Name: [Redacted]

Submit
When Queries Collide

Patient Payment History

Michael Perl
SSN: 893-2-
DOB: 8/30/1951

Billing Address:
70. W. Madison Street
Suite 1050
Chicago, IL 60602
312-673-7281

<table>
<thead>
<tr>
<th>Date</th>
<th>Charge</th>
<th>Credit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/12/2009</td>
<td>$125.00</td>
<td></td>
<td>ER Visit - Hand sanitizer over-exposure</td>
</tr>
<tr>
<td>1/18/2009</td>
<td>$78.50</td>
<td></td>
<td>Very embarrassing lab tests</td>
</tr>
<tr>
<td>1/24/2009</td>
<td>$125.00</td>
<td></td>
<td>ER Visit - Hand sanitizer over-exposure</td>
</tr>
<tr>
<td>2/03/2009</td>
<td>$125.00</td>
<td></td>
<td>ER Visit - Hand sanitizer ingestion</td>
</tr>
</tbody>
</table>
Salami Slicing Variant

- Traditional Salami Slicing has been well known since at least the 1970’s
- Office Space, Superman III...
- Stealing small amounts of money repeatedly can add up
- From June 2007 to May 2008, Michael Largent obtained at least $60,000 from E-trade, Schwab.com, Google
- Brokerages will commonly deposit a few cents to confirm new bank accounts
- Largent programmatically opened thousands of accounts
- The transfers were legal, the phony checking accounts were not
- 11,385 Schwab accounts were opened as "Speed Apex" from only five AT&T IP addresses
Salami Slicing Variant

Root causes:
- Poor application design: Insufficient steps to detect automated account creation

History:
- Apparently, a lack of account confirmation functionality

Prevention
- CAPTCHAs probably aren’t enough
- Where human identity is important, more sophisticated data correlation is required
Logic Flaw Poster Child: SocGen

- Société Générale is a major European bank: over $1 trillion in managed assets, and 160,000 employees

- A leading industry analyst said they were "considered one of the best risk managers in the world." ...until January 2008

- In one year, Jerome Kerviel made $73 billion in unauthorized trades, losing $7 billion

- A junior trader; used to work in the bank’s compliance department.
Logic Flaw Poster Child: SocGen

- Without using any "advanced" hacking skills, he evaded all of the bank’s approval systems

- The CEO described Jerome’s knowledge of the bank’s controls as "intimate and perverse".

- Internal audit findings:
  - Many controls were batch run, and could be evaded within a limited window
  - Some controls were based on the net value of a group of holdings and could be evaded by creating a fictitious opposite entry
  - Some management approvals were email-based and were easily spoofed
Unsolvable: Poker Collusion

- Some logic flaws are impossible to solve
- It can be made difficult:
  - Analyze player win patterns
  - Correlate table-mate frequency
  - Attempt to validate human identity
  - Ask the software client for computer description
Preparing to Test for Logic Flaws

• Obtain or create thorough documentation of:
  • Business rules
  • Business processes
  • Domain data

• Identify hypothetical violations of business rules
  • Where are the rules enforced
  • How can the relevant data be accessed and changed

• Understand the technology used to exchange data between the client & server
Verifying Logic Enforcement

- **Stand-alone transactions:**
  - What business rules apply to this transaction?
  - What is the mechanism of enforcement?
  - What is the purpose of each piece of data sent to the server from the client?
  - Are any data fields in the transaction relevant to business rules?
  - What business domain information is returned by the server?
Verifying Logic Enforcement

- **Multi-step**
  - How is each step defined? (Different URL, query parameter, server-side state, etc)
  - Can a future step be requested before prerequisites are satisfied?
  - Can data from past steps be modified after the initial business logic has been applied?
Verifying Logic Enforcement

- **Combining Processes**
  - Logic flaws can span applications
  - All applications accessed by a user should be considered
  - Publicly-available information should also be a factor
Summary

• Poor design & poor planning lead to logic flaws

• Logic flaws are one-off, custom creations

• Logic flaws are generally driven by underlying programming weakness
  • Unique instances of vulnerabilities
  • Combination of vulnerabilities to create a flaw
  • Requires manual testing to find

• Adherence to secure coding techniques will go far to remove logic flaws but code generally cannot fix design issues.
Trustwave SpiderLabs

- **SpiderLabs Website & Wiki – Papers, Tools, Service Information**
  - [http://www.trustwave.com/spiderlabs](http://www.trustwave.com/spiderlabs)
  - [https://wiki.trustwave.com/display/sl/SpiderLabs+Team+Site](https://wiki.trustwave.com/display/sl/SpiderLabs+Team+Site)

- **Twitter – Security News, Event Information, etc.**
  - [http://www.twitter.com/spiderlabs](http://www.twitter.com/spiderlabs)

- **LinkedIn – Security News, Event Information, etc.**
  - [http://www.linkedin.com/groups?home=&gid=90640](http://www.linkedin.com/groups?home=&gid=90640)
Questions?

Presented by: