Finding and Exploiting Access Control Vulnerabilities in Graphical User Interfaces

KiwiCon 2016

Twitter: @collinrm
Graphical User Interfaces (GUIs)

- Because 'normal' people don't like shells
GUI Security History (Shatter Attacks)

- Shatter Attacks

- Affected platform: Windows NT/2000/XP

- Remove limits of text edit fields
  - Paste input to cause memory corruption → code execution

- Target: progress with system privileges
  - Code execution → privilege escalation

- Now Windows has User Interface Privilege Isolation (UIPI)
  - Can't manipulate UI of process that have higher privileges
GUI Security History (Shatter Attacks)

- Shatter Attacks

- Affected platform: Windows NT/2000/XP

- Remote:
  - Paste input to cause memory corruption → code execution

- Target: progress with system privileges
  - Code execution → privilege escalation

- Now Windows has User Interface Privilege Isolation (UIPI)
  - Can't manipulate UI of process that have higher privileges

This talk is about Access Control issues in the UI
Graphical User Interfaces (GUIs)

- Windows, Widgets, ...
GUls → Widgets and Windows

- **Widget → base UI element**
  - Smallest element in a UI framework
  - On MS Windows: widget = window

- **Common widgets**
  - Window
  - Frame
  - Button
  - Check-box
  - Text edit field
  - Drop down box
  - Slider
Widget Attributes

- Attributes allow to change widget behavior at runtime
  - Allows user interface to be dynamic

- Common attributes

  Enabled → enable / disable widget

  Visibility → show / hide widget

  Read/Write → allow / disallow changing data stored in widget
Widget Attributes

- Attributes allow to change widget behavior at runtime
  - Allows user interface to be dynamic

- Common attributes
  - Enabled
  - Visibility
  - Read/Write

Login button disabled → indicates username required
Access Control

- Fundamental security requirement
- Common in any kind of enterprise application
  - applications that handle sensitive data
- Different privilege levels
  - Create / Add data
  - View data
  - Modify data
  - Execute privileged functionality
Access Control

- Fundamental security requirement
- Common in any kind of enterprise application
  - applications that handle sensitive data
- Different privilege levels
  - Create / Add data
  - View data
  - Modify data
  - Execute privileged functionality

- Implementing access control using the GUI is tempting
Access Control in the GUI

Button used for access control

Disabled Button

Application Specific User

“Hidden GEMs”
Access Control in the GUI

- Widgets can be manipulated
  - Feature of UI frameworks
  - No need to modify application binary

- Manipulate widget → bypass GUI-based access control
A Real World Attack DEMO
Access Control in the GUI

- Widgets can be manipulated
  - Feature of UI frameworks
  - No need to modify application binary

- Manipulate widget → bypass GUI-based access control

- Attacks using the UI are folklore

- **First to systemantically investigate GUI security**
Threat Model

- **Applications with internal user management**
  - Multiple users or user and administrator
  - Accounts are NOT backed by the OS

- **Accounts have different privileges**
  - Reading vs. writing data
  - Executing privileged functionality

- **Application domain**
  - Enterprise applications → users with different privileges
  - Applications that manage data → require access control
GUI Element Misuse (GEM)

- Misusing GUI elements to implement access control

- GEM vulnerability → access control bypass vulnerability

- GEM classes
  - Unauthorized Callback Execution
  - Unauthorized Information Disclosure
  - Unauthorized Information Manipulation
Unauthorized Callback Execution

- Activation of UI element results in callback execution
  - Click button → execute callback → perform operation

- Assumption
  - Disabled UI element cannot be interacted with

- Attack
  - Enable UI element
  - Interact with UI element
    - Execute callback → perform operation
Unauthorized Information Disclosure

- UI element is used to store sensitive information
  - UI element is shown only to privileged user

- Assumption
  - Hidden UI element cannot be made visible

- Attack
  - Set UI element visible
    - UI element is drawn by the UI framework
      - Data stored in UI element can be accessed
    - Access data stored in UI element programmatically
Unauthorized Information Disclosure DEMO

- gemtools_unhide.exe
  - Make all widgets of an application visible
  - Take screenshots of app windows
  - Tool available:
    - http://www.mulliner.org/security/guisec/feed/
Unauthorized Data Modification

- UI element is used to display and edit data
  - Privileged user can edit data
  - Unprivileged user can view data

- Assumptions
  - Read-Only UI element does prevent data modification
  - Data modified only if element was writable → save data

- Attack
  - Set UI element Read-Write
    - Set/Change data
    - Click “save”
Unauthorized Data Modification **DEMO**

- WinSpy++ gemcolors edition!
  - Identify R/W settings of widgets
Widget Configuration

User1 (Low Privileges)       User2 (High Privileges)

“Hidden GEMs”
Technical Requirements 1/2

- Applications must be executed by the same OS user
  - Interaction between applications via IPC

- Attack steps:
  - Discover UI elements (widgets)
  - Obtain window HANDLE for widget
  - Manipulate widget
Technical Requirements 2/2

- All this is done through very basic Win32 APIs
  - SendMessage() family of functions
  - EnableWindow()
  - SendInput()
  - EnumChildWindows() → get all windows
  - SetWindowPos() → visible/hide window
  - GetWindowLong()
  - IsWindowEnabled()
  - IsWindowVisible()
  - GetClassName()

- This stuff is very well documented
UI Frameworks

- On MS Windows a window is the basic UI element
  - Everything is a window

- Win32 API provides basic functionality
  - 'actual' window
  - Button
  - Text field

- Other UI frameworks are build on top of the Win32 UI API
  - Provide their own widget types
  - Implement drawing and receiving user input
Win32 vs. .NET

- .NET
  - Win32 windows + custom widgets
  - Implement drawing and receiving user input
  - Win32 API can see widget but not always manipulate it

- Attacker
  - Can use Win32 API to interact .NET widgets
    - Enough for most attacks
  - Using .NET API provides access to actual .NET widgets
    - e.g., see individual buttons inside a 'button bar'

.NET 'button bar' for Win32 this is one button, for .NET it is 19
Two Corner Stones of GEM Vulnerabilities

- **False assumptions by developers**
  - GUI cannot be changed externally
    - Widget attributes are protected

- **Non sophisticated attacker**
  - *Only point-and-click*
  - Black box attack → change value in field or click button
    - No reverse engineering or program understanding
    - Don't need to manually tamper with files or database
    - No network protocol knowledge
The GEM Miner Analysis

- Systematically test applications for GEM vulnerabilities
  - Automated analysis
  - Complex applications cannot be tested manually

- Black box analysis
  - We do **NOT** require: source code, reverse engineering, etc.
The GEM Miner System

- Explore application UI and record widgets and attributes
- Identify GEM candidate widgets
- Check the GEM candidates
Application Seeding

- Create application specific users
  - Users + administrator

- Create data
  - e.g., items of an inventory management system

- Configure access control (restrict privileges of one account)
UI Exploration

- Explore the application's UI
  - Interact with widgets
    - click button, set check box, select drop down, ...

- Record
  - Widgets and attributes
  - Interactions
UI Exploration – for all privilege levels

- UI Exploration is executed once for each distinct privilege level
- Result: UI State for each privilege level
- UI State
  - Windows, contained widgets, and their attributes
GEM Candidate Identification

- Compare UI States of different privilege levels
  - Widget with different attributes → GEM candidate

<table>
<thead>
<tr>
<th>Level</th>
<th>Attributes</th>
<th>UI Element</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Visible Disabled</td>
<td>TbitBtn</td>
<td>“New Article”</td>
</tr>
<tr>
<td>High</td>
<td>Visible Enabled</td>
<td>TbitBtn</td>
<td>“New Article”</td>
</tr>
<tr>
<td>Low</td>
<td>Visible Enabled</td>
<td>TbitBtn</td>
<td>“Help”</td>
</tr>
<tr>
<td>High</td>
<td>Visible Enabled</td>
<td>TbitBtn</td>
<td>“Help”</td>
</tr>
<tr>
<td>Low</td>
<td>Visible Enabled</td>
<td>Read EDIT</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>Visible Enabled</td>
<td>Write EDIT</td>
<td></td>
</tr>
</tbody>
</table>
**GEM Candidates**

- **GEM Candidate**
  - Widget that likely can be used to bypass access control

- **Candidate information**
  - Widget type and ID
  - Path to candidate widget
  - “successor” (e.g. if widget creates a new window)
GEM Checking

- Execute AUT
- Drive application to GEM candidate
- Test GEM candidate
  - Manipulate and activate widget
  - Inspect result
GEM Candidate Testing

- Different strategy for each widget and GEM type
  - Callback execution: active widget → callback executed?
  - Information disclosure: can widget contain data?
  - Information modification: modified data accepted by app?

- Black box testing
  - Manipulate the UI for testing
  - Check results by only inspecting the UI

- Tests are independent from the application
  - No application specific knowledge needed
Testing Data Modification GEMs 1/4

- Drive application to window containing GEM candidate

![Product window]

Candidate
Testing Data Modification GEMs 2/4

- Set text edit field writable
- Change/Set test value
- Close window
Testing Data Modification GEMs 3/4

- Drive application to window containing GEM candidate
- Check if test value present
Testing Data Modification GEMs 4/4

- Drive application to window containing GEM candidate
- Check if test value present

GEM Candidate confirmed!
Result → GEMs no longer hidden!

**Diagram:**

- **AUT**: Application Under Test
- **Test Engineer**: Initiates the process
- **UI Exploration**: Identifies potential GEMs
- **GEM Candidate Identification**: Refines the candidate GEMs
- **GEM Checker**: Validates the GEMs
- **GEMs**: Final output

**Flowchart Steps:**

1. **UI Exploration** → **UI States** → **GEM Candidate Identification** → **GEM Checker** → **GEMs**

**Annotations:**

- **Widget + Type**
- **Window**
- **Path to Widget**

**Message:**

“Hidden GEMs”
Analyzing Real World Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>GEM Candidates</th>
<th>Automatically Confirmed</th>
<th>Manually Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Disclosure</td>
<td>Modification</td>
<td>Callbacks</td>
</tr>
<tr>
<td>App1</td>
<td>44</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>App2</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Proffix</td>
<td>-</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>24</td>
<td>20</td>
</tr>
</tbody>
</table>

- **App1**: inventory management
  - Multiple users + admin mode

- **App2**: employee and project management
  - Multiple users + admin

- **Proffix**: customer relationship management
  - Multiple users + admin, fine-grained access control

“Hidden GEMs”
Analyzing Real World Applications

- **App1**: inventory management
  - Win32
  - Multiple users + admin mode

- **App2**: employee and project management
  - Win32
  - Multiple users + admin

- **Proffix**: customer relationship management
  - .NET
  - Multiple users + admin, fine-grained access control

<table>
<thead>
<tr>
<th>Application</th>
<th>GEM Candidates</th>
<th>Automatically Confirmed</th>
<th>Manually Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Disclosure</td>
<td>Modification</td>
<td>Callbacks</td>
</tr>
<tr>
<td>App1</td>
<td>44</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>App2</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Proffix</td>
<td>-</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>24</td>
<td>20</td>
</tr>
</tbody>
</table>
Summary

- GEM Vulnerabilities
  - Exist in commercial software
  - Can be exploited by non sophisticated attackers

- GEM Miner Analysis
  - Systematic method to find GEM vulnerabilities
  - Independent of UI framework and application

- The GEM Miner System
  - Can automatically find and verify GEM bugs
  - Implemented for Windows but can be ported to other OSes
Conclusions

- We introduced GUI Element Misuse (GEMs)
  - New class of security vulnerabilities
  - Misuse of the UI to implement access control

- We defined three classes of GEMs
  - Information Disclosure and Modification, Callback Execution

- We build GEM Miner to analyze Windows applications for GEMs
  - We discovered a number of previously-unknown bugs

- First step towards including the UI in security testing
  - We specifically address access control vulnerabilities
Thank you!

Any Questions?

http://mulliner.org/security/guisec/