A Pattern for Secure Graphical User Interface Systems

Thomas Fischer, Ahmad-Reza Sadeghi, Marcel Winandy
Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
Email: thomas.fischer@rub.de, {ahmad.sadeghi, marcel.winandy}@trust.rub.de

***Abstract*—Several aspects of secure operating systems have been analyzed and described as security patterns. However, existing patterns do not cover explicitly the secure interaction of users with the user interface of applications. Especially graphical user interfaces tend to get complex and vulnerable to spoofing and eavesdropping, e.g., due to key loggers or fake dialog windows. A secure user interface system has to provide a trusted path between the user and the application the user intends to use. The trusted path must be able to ensure integrity and confidentiality of the transmitted data, and must allow for the verification of the authenticity of the end points. We present a pattern for secure graphical user interface systems and evaluate its use in different implementations. This pattern shows how to fulfill the security requirements of a trusted path while preserving, in a policy-driven way, the flexibility that graphical user interfaces generally demand.**

***Keywords*-security pattern; secure GUI; secure windowing system;**

I. Introduction

Commodity operating systems offer a feature-rich environment for various applications, including a user-convenient graphical user interface (GUI) systems. Users can easily work with several applications simultaneously, while being able to organize and arrange the application windows on the screen. Users can click through web sites, copy and paste data from one application to another, and adapt the desktop appearance according to their needs and desires. There are even applications that can record and automate user inputs, or make snapshots of the entire screen.

However, such flexibility of functionality has an important security risk. Malware may try to mimic user interfaces and the appearance of security-critical applications, e.g., faking password input dialogs. Other types of malware may try to manipulate the data the user intends to send to a certain application, e.g., by replacing the document for a digital signature application with a maliciously modified version [1]. Although there are techniques to identify applications, such as verification of digital signatures of program binaries or secure bootstrap mechanisms [2], these identities need to be reliably shown to the user. For security-critical applications users need to be able to clearly identify the application they are interacting with, and to be sure the data they enter cannot be accessed by unauthorized applications.

A secure operating system has to provide a secure (graphical) user interface that provides a trusted path from the application to the user and vice versa. The user interface system must be able to isolate the input and output channels of different applications. Furthermore, in an environment supporting multi-level security (MLS) or domain separation the user should know about the security label of the application to which the current displayed user interface element belongs to. The user must be able to identify the trusted path via a security indicator that cannot be faked or simulated by any software running in untrusted applications.

The realization of a secure GUI system is an important, but not trivial task. However, existing secure GUI systems [3], [4], [5], [6], [7], [8] have some elements in common on a conceptual level. We present a pattern for secure graphical user interface systems and evaluate its use in the different implementations.

II. Secure GUI System Pattern

Intent: Provide a trusted path between the user and the application the user intends to use. Security-critical applications require integrity and confidentiality of user input and displayed output. All input and output devices (keyboard/mouse, display) have to be shared by all applications, but security-critical applications need exclusive access to avoid interference with other applications.

A. Example

Consider a system login dialog on a graphical window system where the user has to enter a password. The user should be sure that the dialog being displayed actually corresponds to the system login and not to a faked application. Additionally, the data transmitted between the login dialog and the user (keyboard input and displayed graphical output) should not be possible to be eavesdropped.

B. Context

You need to provide a graphical user interface for several applications with different trust and security requirements for one or more users. You need to ensure that the application the user in front of the input/output devices is currently interacting with cannot fake to be an arbitrary other (possibly trusted) application. You also need to ensure that the input of the user and the output of an application cannot be eavesdropped or manipulated by other applications.

C. Problem

Providing security for user interfaces is non-trivial because they often rely on and communicate with several components. The problem is to combine the individual goals (security and usability) in such a way that efficient interaction with the user interface remains possible.

Experience shows that commodity operating systems cannot provide sufficient security for the GUI of applications (e.g. [1]). Applications can impose other applications by adapting their look and feel. An attack on the user's privacy can be performed by applications that eavesdrop everything the user enters, e.g., keyloggers typically install themselves as device drivers to intercept all keyboard events.

In particular, the solution to this problem has to resolve the following forces:

- You need to provide a graphical user interface system that can be used by several applications simultaneously.
- You need the flexibility that applications can draw any content without constraints (application flexibility).
- Users need to know with which application they are currently interacting with (application authentication).
- Users need to be able to invoke interaction with a certain trusted service at any time (trusted path).
- You need protected input/output of GUIs, i.e., unauthorized applications should not be able to eavesdrop or manipulate user input directed to or display output coming from other applications.
- You (optionally) need controlled communication between user interfaces of different applications – for example, in a multi-level security system copy&paste has to adhere to the information flow policy.

D. Solution

The central idea is to mediate all user input/output through a Secure User Interface (SUI) system, and to separate the content drawn by applications from what is actually displayed on the screen. The SUI controls solely the graphics rendering hardware and the input events from the user input devices (typically, keyboard and mouse). The SUI receives the user input and sends it to the currently active application. Only one application can be active at a certain time, i.e., it has the "input focus". On the other hand, the SUI receives the graphical output of all applications and multiplexes it to the graphics hardware, composed with a visible labeling of applications according to a policy. Hence, only the SUI decides what is going to be displayed on the screen and how.

Besides the input routing and the visible labeling, the SUI has to detect trusted path invocation, i.e., when the user requests to interact with a trusted service, and implement certain trusted path features, e.g., when the user wants to switch the input focus to another application.

1) Structure: The SUI system interacts with Applications on the one hand, and Users via Graphics Hardware (screen, graphics processing unit (GPU)) and Input Devices (typically, keyboard and mouse) on the other hand. Applications are defined as interactive programs that want to display a GUI on the screen and receive user input. Depending on the operating system, Applications can be single processes, groups of processes (e.g., belonging to the same security level in an MLS system), or entire virtual machines.

The main elements of the SUI system are the Input Manager, Display Manager, Copy&Paste Manager, a Policy, and the SUI Control (see Figure 1).

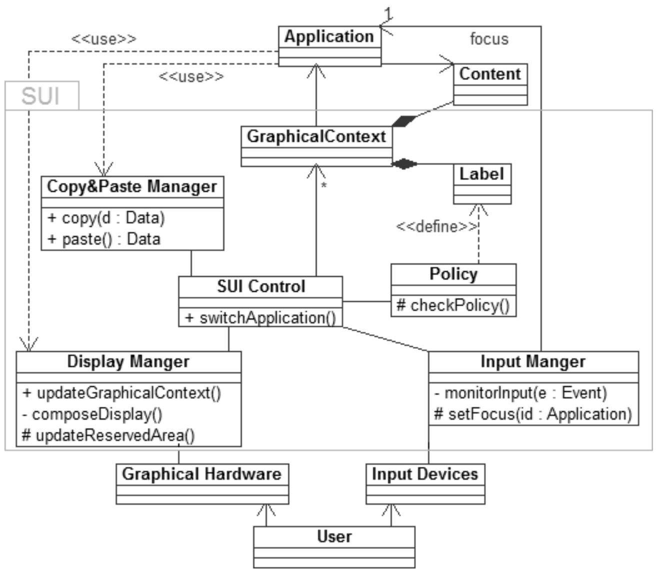

Figure 1. Participating elements of the Secure GUI System pattern.

The *Input Manager* is responsible for the input routing and detection of trusted path invocation. It monitors all input events and forwards them to the Application which currently has the focus. Certain input events are not forwarded, they are used as *Secure Attention Key*, e.g., a special key sequence, to invoke trusted path features of the SUI Control.

The *Display Manager* multiplexes the GUI output and is responsible for drawing visible labeling. Visible labels allow the User to reliably identify Applications, especially the one having the input focus. It is important to note that the visible labeling must be drawn in a way that arbitrary Applications cannot fake it. For example, the Display Manager could only display the content of one application at a time on the screen and keep a reserved area, e.g., on the top of the screen, in which no application can draw. This area is used to display the identity of the current application. The screen could also be divided into fixed regions belonging to different groups of applications, e.g., of the same security level. To allow most flexibility and to display all application windows simultaneously, the Display Manager could apply a window labeling policy [9], i.e., all windows are decorated with colored border where each color corresponds to a security level. In addition to colors, the name of the application (or its security level) can be shown in the window border, and a reserved area on the screen shows the current input focus.

The *SUI Control* implements trusted path features, in particular, to switch the input focus to another Application.

GraphicalContexts represent the GUI windows or screens of Applications. They are composed of two parts: a *Label*, which represents the identity of the Application (e.g., its name or security level), and the actual graphical *Content*. The Content is the only region to which the Application can draw its GUI elements. The Label is under control of the SUI. This prevents that an Application can fake the visible identity of another Application since it has no access to the Label, and the Label is finally visually "attached" on the screen by the Display Manager.

The *Copy&Paste Manager* acts as trusted intermediate component to enable Applications to exchange data in compliance to the information flow policy of the system.

Finally, the *Policy* of the SUI defines the labeling strategy of the Display Manager, constraints to allow to switch the input focus, constraints to allow copy&paste, and the secure attention key events and corresponding reactions.

Note that the Display Manager and the Input Manager are the only components connected to the hardware input/output devices. This has to be enforced by the underlying operating system (OS), e.g., by allowing access to the corresponding device drivers exclusively to these components. Moreover the SUI system relies on proper authentication of Applications and Users, which has to be done by the OS as well.

2) Dynamics: When an Application wants to update its GUI, it draws into its Content object and requests to update the corresponding GraphicalContext. Figure 2 shows the corresponding sequence diagram. The important step in this process is the composition of the labeling information with the basic display content provided by the Application. The Application cannot alter the Label itself, as it is defined by the Policy based on the application identity and composed within the GraphicalContext by the Display Manager.

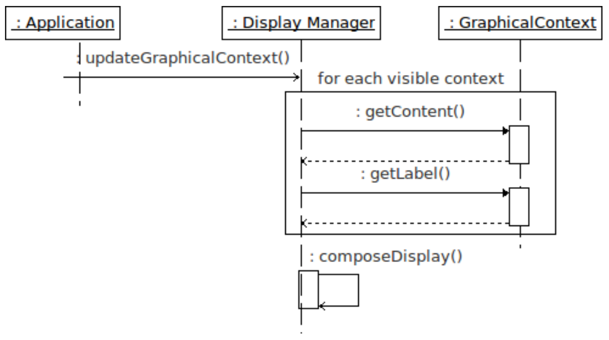

Figure 2. Sequence diagram for updating GraphicalContext

Users can request to switch the input focus by invoking the trusted path to the SUI Control (e.g., via a hot key). Applications may also request to switch the focus to themselves or another Application, e.g., to request a password input or a confirmation from the user when they do not have the focus. In this case, the SUI Control asks the Policy and determines whether to allow or deny the request (see Figure 3). Alternatively, Policy can contain rules that effectively result in asking the user for a password in order to authorize the switch. Once authorized, SUI Control changes the input focus and updates the reserved area if required.

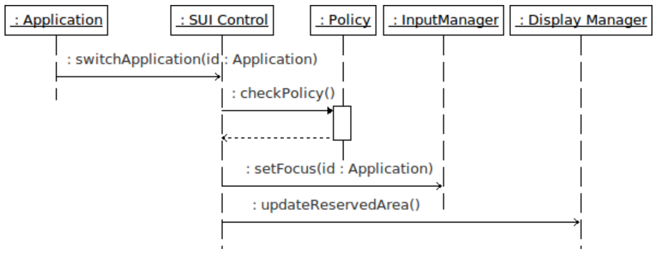

Figure 3. Sequence diagram for switching application

To enable copy&paste, the Policy must define access rules to the Copy&Paste Manager (CPM) which are in compliance to the information flow policy of the operating system. For example, in an MLS system it is not allowed to write data from high to low security levels. Therefore, all data transfer via GUI elements must go through the CPM. An Application can copy (write) data to the CPM, and other Applications may request to paste (read) data from the CPM. The CPM has to enforce the policy on such requests accordingly.

E. *Example Resolved*

By applying the Secure GUI System pattern, a Policy can be defined to keep a reserved area on the screen to display the active application's name. The Display Manager controls solely the graphics hardware and multiplexes the output to the display. Hence, no application can modify the reserved area in order to change the displayed name. This allows the user to identify the login application and to distinguish it from faked ones. Moreover, since the Input Manager routes the user input only to the one application that has the focus, the input cannot be eavesdropped by any other application.

F. *Known Uses*

The Secure GUI System pattern can be found in several existing implementations:

- Trusted X [4], [10] is a multi-level secure X window system. It encapsulates the untrusted functionality of an ordinary X server and polyinstantiates it for each security level. Trusted X multiplexes the windows of all levels and attaches a colored label on each window to indicate its security level. A reserved area on the screen always shows the level having the input focus. A dedicated trusted application implements the secure copy&paste function and mediates the data transfer according to the information flow policy.

- Solaris TX [7] supports multi-level desktop sessions in a similar way to Trusted X, but uses a single trusted desktop system instead of polyinstantiated X servers.
- EWS [5] is a mandatory access control capable window system for the EROS operating system. It consists of a small display server that renders the output of client applications and supports window labeling to indicate the trusted path. It does not have an X server, instead all drawing logic is in the applications, and shared memory is used to define Content. Secure copy&paste is realized via special invisible windows which are dedicated trusted applications.
- Nitpicker [6] is a small, framebuffer-based secure GUI server on top of the L4 microkernel. It controls the physical display and aggregates the virtual screens of clients, which can be native processes or virtual machines. The server also supports visible labels.
- Green Hills' INTEGRITY [8] is a microkernel-based operating systems that supports multi-level security. It displays the maximum security level and the current input security level at the top and, respectively, bottom of the screen as a colored bar. Applications in the system are virtual machines (VMs) that run isolated from each other. The VMs can only access virtual devices and cannot draw on the reserved screen areas.
- The SDH architecture [3] divides the screen in separate regions according to security levels. For each region, an untrusted processor draws the Content, and a hardware-implemented Display Manager multiplexes the output on the screen. A software-implemented Input Manager functions also as SUI Control, i.e., it switches current security level and input focus simply by moving the mouse pointer from one region on the screen to another.

While the examples above are mainly mandatory access control systems or research prototypes, commodity operating systems would also benefit from applying the Secure GUI System pattern. However, this would require to change the access to graphics and input hardware, i.e., not allow every application to arbitrarily use these devices (e.g., preventing installation of "filter" drivers that can intercept keyboard input). Moreover, the GUI system of a commodity operating system needs to separate the graphical content that applications can draw from what is finally presented on the screen (i.e., providing window labeling and/or a reserved area).

G. Consequences

Applying this pattern is expected to result in the following benefits:

- You will get a trusted path for the user input and the application output. This channel is secure against eavesdropping and manipulation by unauthorized applications.
- Security-critical applications need not to implement their own protection mechanisms against faked dialogs because they can rely on the SUI to show the user the identity of applications via labels.
- The resulting SUI system is very flexible since labeling and access decisions can be delegated to the policy.

However, this pattern may also have the following liabilities as consequences:

- The SUI might become a single point of failure. If the SUI does not work correctly, the whole system might become unusable for users, or the security of user intentions is compromised.
- There might be usability issues due to security constraints of the GUI system. For example, when applying a window labeling policy (based on colors), users might need extra training and education to understand the meaning of the colors.
- You need to trust the SUI. For a high assurance system, the components within the SUI (cf. Figure 1) are critical and require high assurance for their development and integrity protection during runtime.
- The increasing usage of graphics accelerator functions in commodity operating systems and 3D game applications may pose an implementation problem since any direct access to the GPU could circumvent the SUI.

Today's GPUs lack of proper virtualization functionality. However, there are approaches to virtualize the 3D OpenGL graphics language [11], [12].

The actual implementation of the labeling in the SUI can be fitted to many scenarios. For example, one can implement different labeling methods for color blind people, and another can use a special braille external interface with separated "secure label zone" for blind people. It is also possible to decouple the label from the screen at all. For example, an additional small display attached to the keyboard could display the security level of the application that has currently the input focus.

Another area of extension is to control audio input/output in the same way as it is done with keyboard inputs. However, details are beyond the scope of this pattern.

H. Related Patterns

On an architectural view, the Secure GUI System pattern is a **Single Access Point** [13], [14] for users to the graphical interfaces of the system's applications, but also for applications to the user input/output devices, especially keyboard/mouse and the graphical processing unit.

Reference Monitor [15] intermediates access to resources defined by policy. This is similar to the Secure GUI System pattern since the SUI system controls the usage of the graphics output and the user input, i.e., which application receives the input. But the Secure GUI System has additionally to ensure authenticity of the end points. Actually the Secure

GUI System can use the **Reference Monitor** pattern to implement a policy-driven secure GUI.

The **Authenticator** [16] pattern can be applied to provide proofs of identity of users and applications, which are requested by the Secure GUI System. For example, digital certificates of program binaries can be used to reliably identify applications, which the operating system has to verify when a process is started.

The SUI has to be executed in an **Execution Domain** [15] which restricts the usage of the user I/O devices solely to the SUI. Other processes must not be allowed to directly access the GPU, otherwise they could easily circumvent the SUI.

To protect the input sent to an application and the output sent to the graphics hardware, it must be possible to isolate process memory. Otherwise, one process could directly read or modify the memory of another process and, hence, intercept or manipulate the data entered by or presented to the user. Process isolation can be achieved with the **Controlled Virtual Address Space** [15] pattern.

To protect the SUI process from other processes and to allow some applications additional privileges, the **Secure Process** [17] pattern is used to isolate processes and assign authorization rights. Some applications have special dedicated meaning and hence have additional privileges. For example, a screen snapshot application would need to capture the graphical output of other applications. Such processes need corresponding authorization rights, which are defined in the SUI policy and enforced by the SUI.

III. Conclusion

Secure operating systems need secure user interfaces to provide users a trusted path to applications. We have presented a security pattern for graphical user interface systems. It shows how to control the access to user input/output devices such as keyboard, mouse, and display, and how to provide an indication of authenticity of applications to the user. The pattern can be found in both research prototypes and commercial products. We expect this pattern to be an important amendment to other operating system security patterns.

References

[1] A. Spalka, A. B. Cremers, and H. Langweg, "The fairy tale of 'what you see is what you sign' - trojan horse attacks on software for digital signatures," in *Security & Control of IT in Society - II (SCITS-II). Proceedings of the IFIP WG 9.6/11.7 Working Conference*, 2001, pp. 75–86.

[2] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*. Oakland, CA: IEEE Computer Society, May 1997, pp. 65–71.

[3] R. Sherman, G. Dinolt, and F. Hubbard, "Multilevel secure workstation," U.S. Patent 5,075,884, 1991, issued December 24.

[4] J. Epstein, J. McHugh, H. Orman, R. Pascale, A. Marmor-Squires, B. Danner, C. R. Martin, M. Branstad, G. Benson, and D. Rothnie, "A high assurance window system prototype," *Journal of Computer Security*, vol. 2, no. 2, pp. 159–190, 1993.

[5] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia, "Design of the EROS Trusted Window System." in *USENIX Security Symposium*, 2004, pp. 165–178.

[6] N. Feske and C. Helmuth, "A nitpicker's guide to a minimal-complexity secure gui," in *21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[7] G. Faden, "Solaris trusted extensions," Sun Microsystems Whitepaper, April 2006, http://opensolaris.org/os/community/security/projects/tx/TrustedExtensionsArch.pdf.

[8] Green Hills Software Inc., "INTEGRITY PC Technology," http://www.ghs.com/products/rtos/integritypc.html, Nov. 2008.

[9] J. Epstein, "A Prototype for Trusted X Labeling Policies," in *Sixth Annual Computer Security Applications Conference (ACSAC)*, 1990.

[10] ——, "Fifteen years after TX: A look back at high assurance multi-level secure windowing," in *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference*. IEEE Computer Society, 2006, pp. 301–320.

[11] Andres, N. Tolia, M. Satyanarayanan, and E. de Lara, "VMM-independent graphics acceleration," in *VEE '07: Proceedings of the 3rd International Conference on Virtual Execution Environments*. ACM Press, 2007, pp. 33–43.

[12] C. Smowton, "Secure 3D graphics for virtual machines," in *EuroSEC '09: Proceedings of the Second European Workshop on System Security*. ACM, 2009, pp. 36–43.

[13] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2006.

[14] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," in *PLoP'97: Conference on Pattern Languages of Programs*, 1997.

[15] E. B. Fernandez, "Patterns for operating systems access control," in *Proceedings of the 9th Conference on Pattern Language of Programs (PLoP)*, 2002, http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html.

[16] E. B. Fernandez and J. C. Sinibaldi, "More patterns for operating system access control," in *Proceedings of EuroPLoP 2003*, pp. 381–398, http://hillside.net/europlop/europlop2003/.

[17] E. B. Fernandez, T. Sorgente, and M. M. Larrondo-Petrie, "Even more patterns for secure operating systems," in *PLoP '06: Proceedings of the 2006 Conference on Pattern Languages of Programs*. New York, NY, USA: ACM, 2006, pp. 1–9.