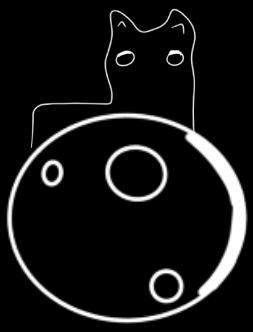


SkyPenguinLabs

Samples

How to analyze GUI components built for x64, Windows

SPL-REC6

Powered by SkyPenguinLabs

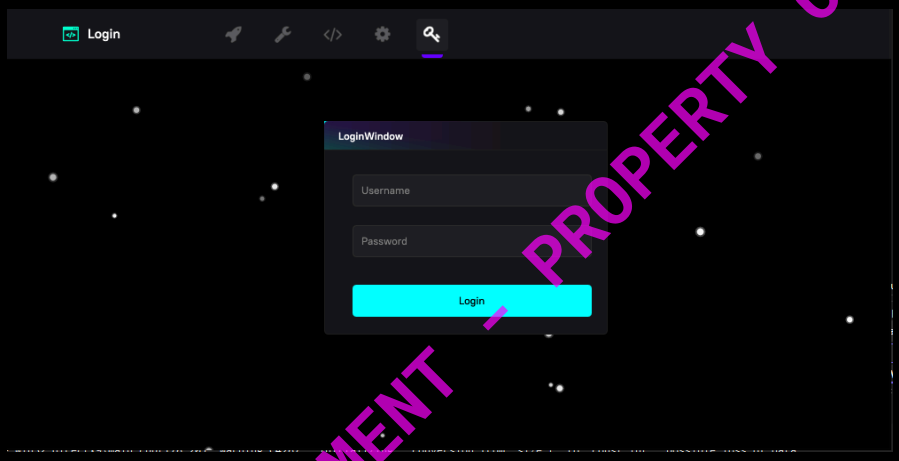
Introduction

> Mapping Technical Components During SRE Scenarios

As a reverse engineer, one of the most important things to keep in mind onto being flexible with the ability to break down components in ways that fit best for you. Not only is this a huge time-saver, but it also prevents many headaches.

Because a lot of beginners often get confused about where to go as far as identifying components in reversing, I thought it would be worthwhile to create a course on how to properly identify and break down specific components.

In this lesson, we will be going over a Graphical User Interface I created for fun and experimentation with C++, shown in the screenshot below, purely with the intention of breaking down and connecting some of the functional components we see and components we DON'T see, which other components rely on.



On top of just mapping out components, being able to break down their logic and comprehend their overall pseudostructures, we will also be applying the logic sprinkled across other lessons regarding the angles you take when attempting to understand or locate components in the app, depending on the context of the scenario.

Without much more to say, the next page discusses the course's content, its purpose, and describes its outline in depth.

Powered by SkyPenguinLabs

Table of Contents (ToC)

> Lesson Purpose and Contents in Further Detail

Purpose: The purpose behind this lesson is to give the reader a direct understanding of how to view specific components inside of an application, which enables you to break them down enough to document them in memorable ways. None of this content is forcing itself into a strict real-world case scenario, but more so aims to teach the reader different techniques for software reverse engineering strategies

- **Section 0x00 |** Prerequisites - *(What you may need to know before reading, what this course uses, what you may need before going further, etc. This exists in every course)*
- **Section 0x01** | The Fundamentals - *(To start the lesson off, this section will be discussing what you do when you need to break down a component within an application, and how you should take angles on it. Specifically, this will be discussing the methodological aspect of it, with a focus on understanding what widgets and components are within apps, and what we mean when we say 'functional we can see' and 'functional we can't see'.*
- **Section 0x02** | The Scenario At Hand - *(This section explains the scenario we are going to be working with, the application, and the component we are going to be targeting. Also, grasping how we are going to expect the component to look based on the graphical layout.*
- **Section 0x03** | Reversing It & Documenting It - *(Taking IDA and using it to reverse engineer the login component. Taking the methodologies and applying them appropriately to reverse the component and get around specific problems in the app. We also do post-analysis of the routine, document it, and graph its logic out.*
- **Section 0x04** | Conclusion - *(Finish everything within this lesson up, conclude what we learned and send you off to the next one)*

Powered by SkyPenguinLabs

Section 0x00

> Prerequisites

Before being able to work alongside this lesson or understand the material behind it, I would like to inform you that this course requires background experience in the following areas:

- **Software Reverse Engineering -** You do not need to be extremely advanced, as we are not going to be doing anything too crazy, but we will need you to have an understanding of how to use frameworks like IDA or whatever framework you choose to use while following along with the reversing portion.

- x64 ASM - Simply just have experience reading this or at least grasping the registers that exist on this ISA.

To learn the methodologies in this lesson hardly requires any background experience. I would say having a minimal background in software RE covers the basis for all of what is required. Pretty simple lesson, which explores a large amount of depth!

With all of that being said, I would like to continue onto the next page, which begins this lesson.

Note that this content relies on a GUI application you can find here or by scanning the barcode below <presentation material for us>:

Powered by SkyPenguinLabs

Section 0x01

> The Fundamentals

When it comes to reverse engineering software, more particularly, desktop applications, we are more notably thrown into the deep, forced to unravel thousands of functions that make no sense.

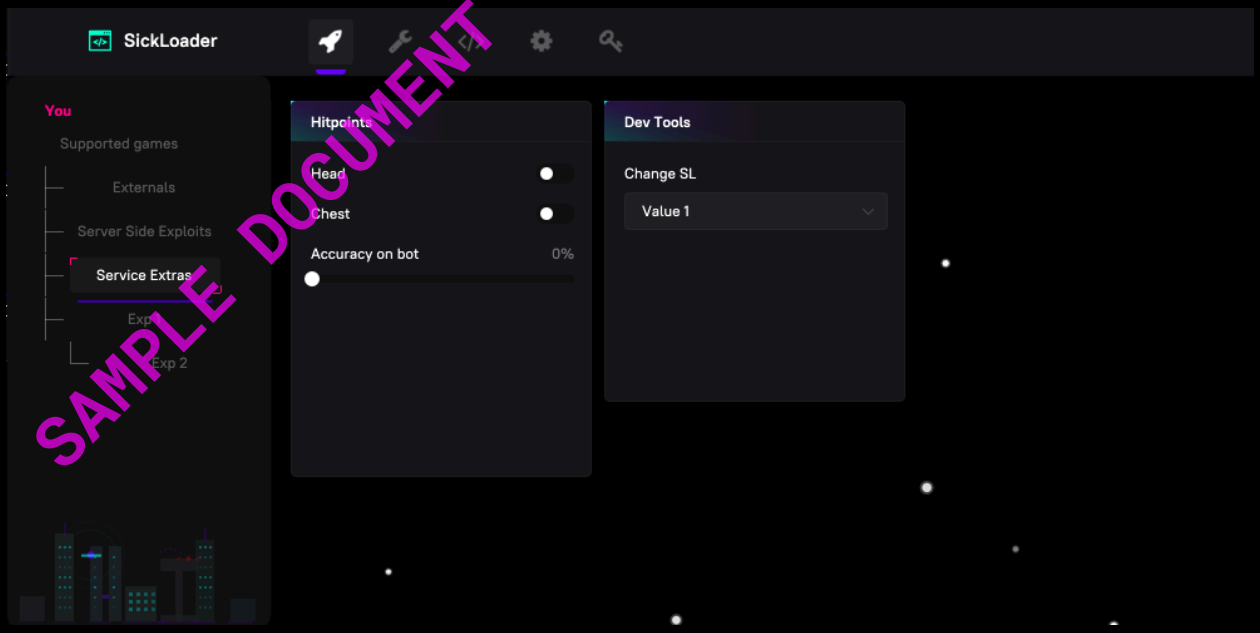
This is because, if a desktop application is a fully packed GUI, it is most likely using a library and rendering agent such as ImGUI x D3D9 (*popular among game cheaters for some awful reason*), or D3D9 x custom rendering (*which most people do*). These libraries are fucking HUGE! Analyze the backend of ImGUI here. imGUI is considered “lightweight” by some people as well.

When analyzing GUIs, its important that you understand how to track down what individual components are. In this lesson, our entire goal is to breakdown a component by hand using various techniques of understanding its context.

The first thing we need to chomp down on is what I mean when I say the word ‘component’.

A component often refers to an individual functional piece of a software technology.

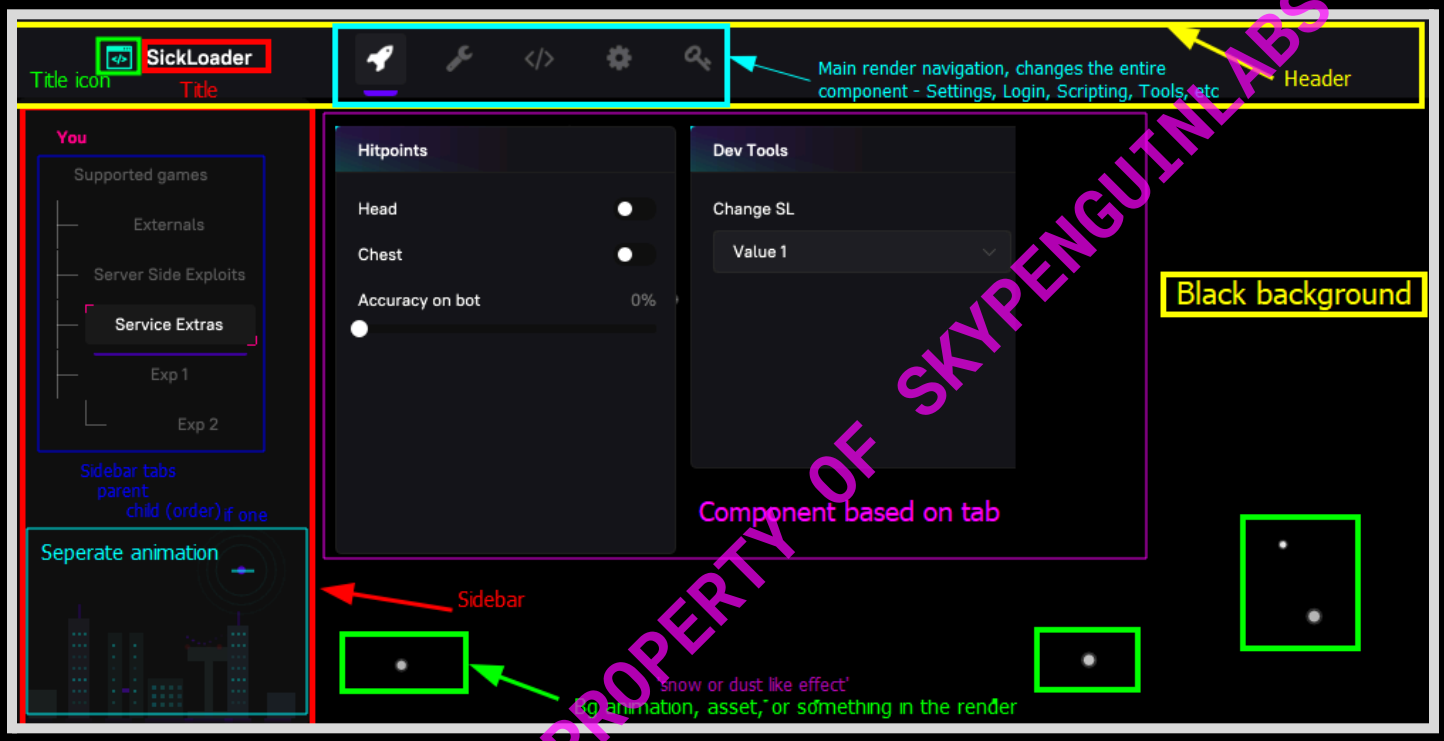
For example, our application lets us check out the main menu.



Anybody without the background might struggle on pinpointing this, so let me ask it for you.

> What's here?

Let's change that- screenshot a little bit. Take a different view on it.



Everything you see within that screenshot is a direct breakdown of the applications first render, along with its primary renders as components.

This is what we mean when we say we have a *component view* of the application. Instead of looking at it all compiled together, we can get a better view of how things are.

Now, despite us having things in this shot that show us what these components are, I have not exactly told you how they get ordered.

See, while all of these are components, some of them are child components, which rely on specific parent components being active for anything to begin with. In this case, the tab we are on in this screenshot is just a parent tab being active; the child tab and component 'Exp 1' are inactive, so it does not render.

So let's categorize these a little bit better and lessen the hurt on our brains.

Parent Component

The first thing we want to make sure we can identify or categorize are the parent component(s).

- Am I going too fast? Let's pause, this is the application mentioned in the prerequisites, it's nothing you compile yourself, instead, I give you the exe compiled for x64 targets (*Windows*). The idea of introducing you to components in an application through this app is not something you have to follow along with, yet by using the app, we use it as a visual demonstration because it's the only app that makes sense for us to show ethically within the context. The screenshot above lists all of the generalized, uncategorized components, and now we need to categorize them into parent-child components.

From a software engineering standpoint, identifying these components is easily done by thinking of them as a layered system.

Usually, the bottom layers are often the thickest; the top layers are the skeletal, navigational, and introduction layers, which give you all the necessary information to get to the primary component.

Given this is the case, we can identify the parental components as follows:

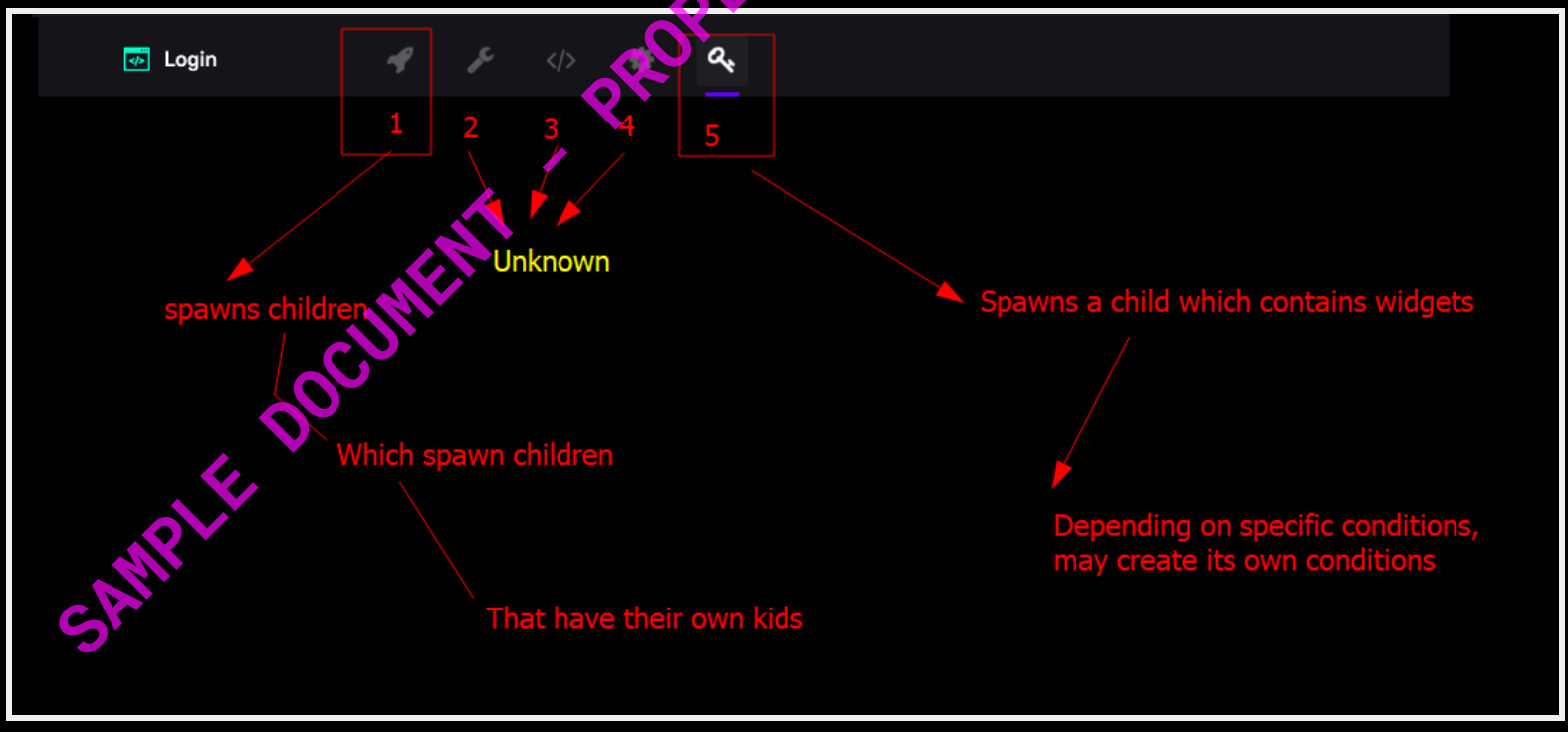
- The header: Not sure if you noticed, but when we were on different tabs in the different screenshots (*on the main tab in the second screenshot, and the login tab in the first screenshot*), the header was the only consistent navigational component. Yes, while the sidebar did exist, this is the parent UI component that is present.
    - Note: Technically speaking, it is not the first UI component, as the "first" would be the GUI's main rendering container itself, along with the backgrounds and separate components used to generate the animations. But we use this as the first for our visual reference. Interaction-wise.

Now that we have identified the header as the parent component, changing the way the child components respond, we can start identifying all of the child components and the application widgets that come into play with them.

- Mind Refresher - An application widget is a type of feature embedded into an application's UI render, which allows users to interact with specific functionalities. For example, a button widget produces a button, a slider widget is commonly used for '%' sliders and amount sliders such as volume (*even though, according to audio standards, using sliders for volume is often seen as an incorrect practice, due to the way audio percentages get mapped to curves*), and more. Widgets are a huge portion of both parent and child components, but are most commonly used in child components.

Child Component(s)

Now, the child components of this application change depending on what type of tab we are on. Since we are using this app as a user right now, let's picture the navigation routines like this.



Because we visited both tabs 1 and 5, we understand that 1 spawns a bunch of tabs on the far left, which are their parents of child components, which spawn their children along with widgets, which can also have their children.

We then understand that 5 is simpler, it spawns a child which can have its own widgets and as a result of actions from those widgets, spawn more children. But it stops there. Not too interlaced.

Now the fun part is being able to reference what is what. Because, truly, we are only expecting this to be played out as such. However, the controversial part with this is that rendering on a GUI happens SO fast now (speed aside as well) that even if it were possible, trying to see the order of a GUI render would be impossible.

So this is why we make assumptions about the components and use these assumptions to build out our reverse engineering goals when trying to pick apart an application. Now, let's stop being a user, try to view this all from a different angle to help ourselves.

Viewing It From a Software Internal Viewpoint

Before attempting to reverse-engineer the application, it's important to try and visualize its internal structure from an internal perspective. Even if you do not have the experience. This simply means focusing on the internal hierarchy of component calls.

Essentially, we can only infer this based on what's visible.

For example, consider the login component: it likely calls functions to render two input fields, suggesting there's a shared function used to render text inputs. There may also be a separate function for rendering a button. When the button is pressed, it likely sets a variable or triggers a function that is related to the input widgets that were rendered. If certain conditions are met, such as the variable having a specific value, or configuration values being set/detected as changed and confirmed, the program then proceeds to execute the login process, which handles authentication.

From here, we can start to see how we might navigate to any component we want, such as the tab rendered by the GUI in the application's main menu (*accessed by selecting the first icon in the header*). This can be done by tracing

string-based identifiers or program symbols, which can provide insight into the underlying functionality or context of the surrounding code.

Now, since the GUI has a frontend we can see, it also has a backend that we can not see, which gets triggered by the frontend. I labeled these components as

- ***Components you can see -*** often, frontend user widgets, buttons, input forms, etc, which influence backend functionalities and trigger functions on the backend.

- ***Components you can't see -*** Anything internal to the application, such as routine managers, or the specific functions called by the frontend functionality, which are hidden or barely visible.

Most of the time, we are focusing on getting to the backend functionalities and backend components we can't see, because these tell us the most about the internals. However, we can not get there before internally pinpointing the components we can see, which influence and reference the components we can't see.

With that being said, the internals of the application you are paying attention to are important to think about because they prepare you even more for locating the components.