Conclusion

In previous posts, we've explored how analysts can use radare2 for malware analysis. This powerful tool allows for a wide range of customization via aliases, macros, and functions. Along the way, we'll also explore how we can effectively implement binary and function differencing with radare2.

To get started with radare2, you'll need to create a `.radare2rc` file. This is a configuration file that stores settings and aliases. It can be helpful to automatically print the entire configuration file out as radare2 starts up.

### Passes of Arguments to radare2 Macros

From within the r2 suite, you can see all the defined aliases and macros by typing `r2 -c`. In earlier posts, we've demonstrated this.

### Reading Code and Command Line Options

As an example, let's add the following function to our shell configuration (e.g., `~/.bashrc`):

```bash
function show_diff {
    radiff2 -md -g
}
```

This function allows you to specify either three arguments (a function name, and two file paths) or four (two offsets, two file paths). This can be very useful for finding common code differences between two samples.

### Macro Management

Just as most shells have a `read command` configuration file (e.g., `~/.bashrc`), radare2 has its own configuration file, `.radare2rc`. If you later edit this configuration file, you'll need to restart radare2 for the changes to take effect.

### Customization via Aliases

Aliases are also useful for chaining oft-used commands together. If you find yourself always running the same set of commands, you could replace them with more memorable command names of your own. Creating a shell script and adding it to `.bashrc` is a good way to do this.

### Customization via Macros

Macros are written inside parentheses with each command separated by a space. For example, to check files against Apple's XProtect database signatures, you could use a macro like this:

```bash
macro my_macro {
    $XPROTECT='/Library/Apple/System/Library/CoreServices/XProtect.bundle/Contents/Resources/XProtect.yara'
    yara -w $XPROTECT
}
```

### Searches for Places Where an Executable Gathers User and Local Environment Information

As shown in the image above, the files are quite different sizes. This kind of information is extremely helpful for creating effective signatures.

### Binary and Function Differences

Search for places where an executable gathers user and local environment information. This is how to tell the r2 shell that we want a binary and function differencing with radare2.

```bash
radiff2 -md -g
```

This is how to tell the r2 shell that we want binary and function differencing with radare2.

### Conclusion

Malware analysis is an ongoing process. As analysts, we must be constantly learning and adapting to new threats. Radare2 is a powerful tool that allows us to do just that. By customizing our shells with aliases, macros, and functions, we can streamline our workflows and make our work more efficient.

Readers of our previous posts on radare2 will notice that we've continued to explore advanced topics in this area. As we do so, we hope to inspire others to dive into this fascinating world of malware analysis.