Updating Internal State

Let's see how a user could choose to dictate how internal state is updated.

WE'LL COVER THE FOLLOWING

There are many different ways the user could choose to dictate how internal state is updated.

For example, the hacker could be more explicit about making their own state changes only when the action type refers to a toggle expand.

Here's what I mean:

```
// user's app
...
function appReducer (currentInternalState, action) {
    if (
      hasViewedSecret.current &&
      // look here 👇
      action.type === useExpanded.types.toggleExpand
    ) {
      return {
        ...action.internalChanges,
        // 👇 override internal update
        expanded: false
      }
    }
    return action.internalChanges
  }
...
```

By doing this we may extend our custom hook's functionality to expose an `override` callback like this:

```
// useExpanded.js
...
```

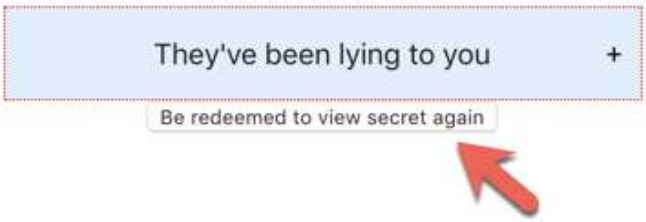
```
  // exposed callback 👇
  const override = useCallback(
    () => setExpanded({ type: useExpanded.types.override }),
    []
  )

...
useExpanded.types = {
  toggleExpand: 'EXPAND',
  reset: 'RESET',
  override: 'OVERRIDE' // 👈 new "OVERRIDE" type
}
```

The `override` type is then handled via our internal reducer as follows:

```
...
case useExpanded.types.override:
      return {
        ...state,
        expanded: !state.expanded // update state
      }
...
```

What this allows for is a way to override changes. For example, the hacker may decide to show a button a user may click in order to view the secret again.




```
// override being retrieved for our hooks call 👇
const { expanded, toggle, override, reset, resetDep = 0 } = useExpanded(
    false,
    appReducer
  )

...
// button is only shown if reader has viewed secret
{ hasViewedSecret.current && (
        <button onClick={override}>
                        Be redeemed to view secret again
       </button>
)}
```

Note how the button's callback function is `override`.

This allows for the expanded container to be open — and letting the user view the secret one more time.

What We Have Now

Here's how it works so far.

```
.Expandable-panel {
    margin: 0;
    padding: 1em 1.5em;
    border: 1px solid hsl(216, 94%, 94%);;
    min-height: 150px;
  }
```

Quick Quiz!

Let's take a quiz.

Q What does the `override` functionality allow us to do?

COMPLETED 0%

1 of 1

We're almost done with the course! Let's have a look at some quick cleanups in the next lesson and then we're done!