1 Screenshots

The fleeting nature of a message sent via snapchat is what makes the service so attractive. In theory, a message sent is visible only for the duration the sender intends. This behavior can be thwarted on a mobile device. By taking a screenshot of the current contents of the screen, the user would be able to capture a picture of the message (including window "chrome").

The security policy around snapchat is tailored to the data which snapchat has the ability to control: the contents of the message. Snapchat only has the ability to control the messages delivered to it, and it has the ability to delete messages. As such, the security policy that Snapchat implements can be expressed as deleting the message from the device.

Obviously, this screenshot-taking action does not break any sort of security policy set forth by Snapchat itself. In fact, Snapchat does not handle screenshots directly. On a mobile device, screenshots are handled by the device's display system, which is married to the operating system. This is to say that the services provided by the device's display system are indistinguishable from the operating system itself, from the app's point of view. To Snapchat's point of view, the message was deleted.

From a meta-policy standpoint, however, the ability to take a screenshot is a fatal flaw in the idea of a message that lasts mere seconds. If the meta-policy is defined as "a user should only be able to see a message for a limited amount of time", and the user may take a screenshot of the message, preserving their ability to access the message beyond the time intended, then the meta-policy is being broken by the system being allowed to take screenshots of the content.

Protecting Against Screenshots on the Mac

On desktop operating systems (and in our case, the mac in particular), the operating system grants the application more latitude in how it is displayed. The display system is less tightly bound to the operating system, and the display system can be tweaked to suit the application's specific needs.

In researching this project, we came across an interesting property of the Mac OS X built-in DVD player application. Taking a screenshot of the application does not work. Instead of the still-frame you would expect to be in the screenshot, a grey and white checkerboard is used in place. The windowing system seems to be unable to expose the contents of the video to whatever part of the system handles screenshots.

Further research into why the DVD player application does not expose screenshots revealed a property set by the interface that disallows the window system to allow access to the screen buffer of the player-window. By using this window property, we should be able to block access to the contents of our message from other processes, to include the process that takes a screen grab.

What needs to be done is test corner cases: in the event that a screenshot were taken as root, would the window property discussed above still protect the message, or have we found a hole in the window manager's window sharing policies? This is a subject that still needs testing.

Thoughts on Protecting Against Screenshots in Mobile

A few methods could be used to attempt to thwart the use of screen-grabbing a message meant to be temporary. The obvious solution would be for platform-makers (Apple, AOSP, others) to allow for an application to request that the system not be allowed to take screenshots of the active application. Although this solution would be the most globally effective, it would require changes to the platform to work. Alternatively, the screen could be strobed to provide a limited timing window for the attacker to take any screenshot of what would be on the screen – most of the time, a strobed non-message image.

What About a Camera?

Screenshots are only part of the problem. In both cases, desktop and mobile, not only can the user take a screenshot of the message, but they can also take a picture of the physical device (with a message on-screen) with a camera. If taking a screenshot of the screen violated metapolicy, then taking a picture of the screen certainly does too.

There are a number of solutions that could be used to solve this. The first would be to make the contents of the screen obscure such that a camera would have a hard time capturing the image. This could be accomplished by dimming the image or strobing a black overlay on top of the image. These methods rely on the quality of the camera, and the ability of the adversary to time pictures being taken. Both methods are also inherently passive.

Alternately, a more active approach could be taken, where the app on a mobile device or an application on a proper computer could use cameras available to watch for cameras. This method would require computer vision techniques to create an "imaging device identifier". In theory, if the application were forced to run on screens with a camera attached (usually the main screen), then the application could watch for imaging devices pointed at the screen, and blank the screen if such an imaging device were suspected.