Recall that OS-level User impersonation attack (OS-UI) is a type of attack where an OS-level attacker impersonates the user and send user-unintended requests to the service provider. For instance, Mallory may impersonate Alice and send a request to her bank and transfers $100 out of Alice’s account. Defenses [1, 2, 3, 5, 6, 8] try to deliver user intention to service providers in the format of raw user inputs. For instance, if Alice never typed in number $100 on her bank’s transfer page, then any request attempting to transfer money out is not what Alice intends. One assumption of all this work is that user input reflects user intention. We propose a type of user interface (UI) attack, called Context Forgery (CF), that tricks the user, Alice, into carrying out unintended action by herself, by showing misleading and luring contextual information. The idea of CF is similar to clickjacking [4, 7], but the differences are 1) CF shares the same threat model as OS-UI: an OS-level attacker 2) CF shares the same goal as OS-UI: to sent out user-unintended request to the service provider causing service interruption. We point out all existing defenses for OS-UI fall prey to CF.

I made several modifications to my original proposed defense. Our current design, called AINT, requires service providers to 1) tabularized their web pages so that UI elements fall into cells and 2) pre-calculate a hash value for every cell based on its expected appearance using an image hash. On the client-side, when a tabularized web page is rendered, AINT captures and validates by validating the rendering of each cell using a combination of image hash and optical character recognition (OCR). Image hash hashes similar images to similar hashes, and thus, by setting an appropriate threshold in the comparison of client-side calculation and ground truth hashes from providers, AINT can tolerate the client-side rendering variations. OCR ensures texts displayed is identical to the expected text in raw HTML data. OCR is sensitive to subtle changes such as single-character tampering. A validated display is said to be free of UI modification, thus preventing any UI attack, including CF. The validation is performed continuously. At the same time, AINT extracts user focus and user inputs form the display using a combination of structural knowledge and OCR. AINT relies on an important observation that a careful user always corrects the display to follow her intention. For instance, a non-malicious user, after typing $100 on the keyboard, would verify this number on the display in case she mistypes. Relying on this observation, user inputs extracted off screen are treated as her intention and is used to generate service-specific requests, similar to JavaScript in modern web programming.

I have implemented AINT. The server-side modification is mostly simulated. We show an example of tabularized web page in Appendix A. The client-side AINT is split into four modules:

- **IntData**: This module stealthily captures authentic user IO data and user display (context).
- **IntUI**: This module validates the UI is rendered properly according to the ground truth information from service providers.
- **IntInput**: This module tracks user focus and extracts user inputs and semantics. Since the display is validated and the user enters and validates her input non-maliciously. We assume the captured inputs reflect her intention.
- **IntRequest**: This modules takes inputs from IntInput and generates a service request in a secure environment and send it to the service provider.

The client-side implementation is done in Xen’s dom0. IntData is simulated by pre-recording user interaction. IntUI is implemented using OpenCV, Tesseract and Wavelet image hash. OpenCV provides color manipulation and contour detection. Tesseract is an OCR engine from Google. IntInput is implemented similar to IntUI, it extracts user focus based on the focus box, a blue gradient that occurs outside of the textbox when a user focuses, it also extracts user inputs using Tesseract. IntRequest is mostly simulated using a standalone process in dom0. We plan to move IntRequest into a trusted execution environment (TEE) such as Intel SGX.
References


A An tabularized web page example

![Tabularized web page example](image)

Figure 1: TD: an example of tabularized web page, user's view
Figure 2: TD: AINT’s interpretation