Building Trust in the User I/O in Computer Systems

Yeongjin Jang
Georgia Institute of Technology
Problem Statement

• User I/O is important
  • Input controls system / output contains sensitive data
  • User input/output makes security decision

• Many attack points on the systems
  • Neither isolated nor protected
  • Attackers modify / inject / eavesdrop I/O data
Thesis Scope

• Building trusted user I/O path in computer systems
  • Do not let attacker intervene in the I/O path

• Approaches
  • Analyze systems’ user I/O paths and threats on them
  • Build security mechanisms to block attack pathways by guaranteeing:
    • Integrity
    • Confidentiality
    • Authenticity
Thesis Topics

• Integrity of User Input
  • Gyrus – authorizing network traffic output using user input information

• Confidentiality of User I/O
  • M-aegis – providing end-to-end encryption of user I/O in messaging apps

• Authenticity of User I/O
  • A11y attacks – presenting attacks caused by missing authenticity check

• Assurance of User I/O
  • SGX-USB – establishing a secure USB I/O channel in Intel SGX
Gyrus: Protecting the Integrity of User Input

Send $1.00 to Yeongjin

Click!

Processing...

Send $1.00 to Yeongjin
Motivation

- User controls system by supplying input

- Think a computer as a function:
  - On user’s input, there will be an output

- Can we make a system that only generates a “correct” output that correspond to the input?

\[ F(x) = 2x \]
Matching Network Output to User Input

- User types "send $1.00 to Yeongjin"
  - Expected network output: send $1.00 to Yeongjin
- A malware in the system can alter the value of network output
  - Malicious network output: send $1000.00 to Bob
- But, *user input* has the correct values  
  Send $1.00 to Yeongjin

Can we use utilize *user inputs* as basis for a correct behavior to block attacks?  
E.g., only allow the *traffic that matches with user input*
Related Work

• Use Timing Information for User Intent Detection
  • BINDER [Cui et al., ACSAC ‘05]
  • Not-A-Bot [Gummadi et al., NSDI ‘09]

• Method
  • Monitor physical keystroke/mouse clicks
  • A network packet sent within a short-time after user action is user intended
    • $T_{\text{network}} - T_{\text{input}} < T_{\text{threshold}}$
    • E.g., packet sent 500ms after user’s action
Related Work

• Use UI Widget for User Intent Detection
  • User-driven Access Control [Roesner et al., Oakland ’12]
• Bind permissions with UI widget
  • Only grant permission to resource if the user clicks a widget
Use On-screen Text as User’s Intention

• New security policy: What You See is What You Send (WYSIWYS)
  • Assume on-screen text is user-intended input
  • Only allow outgoing traffic that matches to on-screen text

Send Yeongjin, $1.00
Send Bob, $1000.00
Examples

• We can infer expected **network output** from **on-screen text**
Cases of WYSIWYS

• Internet Messenger  
  • Messaging, e-mail, etc.

• Submitting text-forms  
  • Online banking, online social network (facebook), etc.

• Not WYSIWYS  
  • Uploading files  
  • Encrypted traffic -> Man-in-the-middle proxy can handle standard encryption (TLS)
Threat Model

• Not trusted
  • OS and all lower privileged programs

• Trusted
  • Virtual machine monitor and programs in dom0
  • Input devices
  • Display device

• No physical access to the machine
Design

• Capturing User-intended Input
  • What you see

• Monitoring Network Traffic
  • What you send

• Protecting Security Monitor
  • A secure way of matching what you see and what you send
Read On-screen Text from UI Elements

• UIAutomation
Per-application Signature

```json
{
"TAG" : "LIVEMAILCOMPOSE",
"EVENT" : "LCLICK",
"WINDOW" : "ATH_NOTE",
"COND" : {
  "0" : {
    "CONT" : "BUTTON",
    "NAME" : "Send this message now"
  },
  "+2" : {
    "CONT" : "EDIT",
    "NAME" : "To:
  },
  "+3" : {
    "CONT" : "EDIT",
    "NAME" : "Subject:
  },
  "P-1CCCCCCCCC" : {
    "CONT" : "PANE"
  },
  "CAPTURE" : {
    "A" : "+2.value",
    "B" : "+3.value",
    "C" : "P-1CCCCCCCCC.value"
  },
  "TYPE" : "SMTP",
  "BIND" : {
    "METHOD" : "SEND",
    "PARAMS" : {
      "to" : "A",
      "subject" : "B",
      "body" : "C"
    }
  }
}
```
Monitoring Network Traffic

• Deep packet inspection
  • Redirect network traffic into a proxy, then inspect the content

• Encrypted traffic
  • Use man-in-the-middle proxy to decrypt, inspect, then encrypt again

• Text transformation
  • We can apply same transformation to the user input
  • E.g., ”Hi there” -> Hi%20there
Monitoring Network Traffic

- Use database to store user interaction data

Send $1.00 to Yeongjin

Click!

Processing...

Send $1.00 to Yeongjin

Store (Paypal, $1.00, Yeongjin)

Query (Paypal, $1.00, Yeongjin)
Protecting Security Monitor

• Modern malwares have the highest privilege in OS (i.e., root)
  • To monitor the system, we need more higher privilege

• Virtual machine isolation
  • Hypervisor isolates security monitor
  • User VM cannot attack security monitor
  • Security monitor can inspect User VM through Hypervisor
Data from User VM is not Trustful

- UIAutomation data from User VM
- Attackers can alter the data
  - On screen: send Yeongjin $1.00
  - On UI Element: send Bob $100.00
  - The system will catch (Bob, $100) instead of (Yeongjin, $1.00)
Security Overlay

Combined Screen
What You See == What Monitor Captured

- Receive UI data from untrusted VM

- Re-draws all editboxes in secure domain
  - Draw at the exactly same location, size, etc.
  - Update text on each change (Autocomplete)

- User will only see the data at the overlay
  - Make sure our security monitor correctly capture the intended text
On clicking Continue, Stores - mail and amount

“ACTION” : “Paypal Send”,
“Recipient” : yeongjinjanggrad@gmail.com,
“Amount” : “1.00”

Authorization Vector

Security Monitor VM

Guest VM

Network Application (e.g., Paypal)

UI Monitor

Hypervisor

Network Device

Input Device
Application Examples

This is the content of e-mail.

It supports Rich-text, colors, text-size.

Thanks,
Evaluations

• No traffic can go out without having proper user interaction
  • All transactions sent from malware was blocked
  • Attacker can still launch denial of service attack
    • **Fail-safe**: no attack traffic can go out

• Negligible overhead on interposing user input and network monitoring
  • Adding 34ms of delay on click
  • ~50ms of delay on webpage load, 3% overhead on bandwidth
M-Aegis: Protecting the Confidentiality of User Input/Output
# The Status Quo: Encryption for Messengers

<table>
<thead>
<tr>
<th></th>
<th>Encrypted in transit?</th>
<th>Encrypted so the provider can’t read it?</th>
<th>Can you verify contacts’ identities?</th>
<th>Are past comms secure if your keys are stolen?</th>
<th>Is the code open to independent review?</th>
<th>Is security design properly documented?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook chat</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Google Hangouts/Chat &quot;off the record&quot;</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Skype</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SnapChat</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Standalone Solutions

• Protect data confidentiality
• Good isolation from untrusted entities

• Examples:
  • PGP, Pidgin, TextSecure, SafeSlinger, FlyByNight, etc.

• Problem:
  • Requires open protocol
  • Do not preserve user experience
Browser Plugins/Extensions

• Provides transparent integration with applications of interest

• Examples:
  • Ghost for chat, TrustSplit, NOYB, SafeButton, etc.

• Problem: Only applicable to web applications.
  • How about native apps and mobile devices?
M-Aegis: Design Goals

• Targets native applications and mobile devices
• Offer good security
  • End-to-end encryption, and strong isolation from untrusted entities
• Preserve user experience
  • Transparent interaction with existing apps
• Does not require protocol reverse engineering
  • A sufficiently general-purpose approach

• Out-of-scope: Key exchange
  • We assume key exchange can be done by other means (out-of-band).
Threat Model

• Untrusted parties:
  • Service providers
  • Client-side apps
  • Middle boxes between a service provider and the client-side app

• Trusted components:
  • Hardware, OS
  • Soft keyboard
  • Components of M-Aegis
M-Aegis Architecture

• Layer 7.5
M-Aegis Architecture

• UI Automation Manager (UIAM)
  • Gives M-Aegis the context of the screen
  • Provides information to correctly render mimic GUIs on L-7.5
  • Relays user touch to the underlying app
M-Aegis Architecture

• **Per-Target Client App (TCA) Logic**
  • Processes UI tree to determine a TCA’s current UI state
  • Draw overlay for en/decryption
    • Editbox for message, etc.
    • ‘Send’ button
    • Display decrypted string for an encrypted message
M-Aegis Architecture

• Searchable Encryption Scheme
  • Easily-Deployable Efficiently-Searchable Symmetric Encryption Scheme (EDESE)
  • Main idea – tag the encrypted text
  • Utilize bloom filter (BF) to “collect” keywords.
    • Problem: email providers don’t support BF tests.
    • Solution: cleverly encode BF in such a way that it is searchable by simple string matching.
M-Aegis WhatsApp Workflow

1. Do key exchange with a friend
2. On entering the messaging UI, TCA detects recipient information
   - Retrieve the correct key to use
3. TCA will detect encrypted messages, and decrypt them then display on the overlay
4. TCA will overlay editbox and send button and send encrypted message on clicking the button
M-Aegis Gmail Preview

11:41 AM
me
maxing out the screen this time - six

11:41 AM
me
test the robustness of system - going five

11:40 AM
me
the subject gets longer - four

11:40 AM
me
three words subject - mimes

11:40 AM
me
longer subject - content doesn't

11:03 AM
me
great - encryption t

10:59 AM
me
Hello – Clear email
Performance Evaluations

• Experimental Setup:
  • Stock Android phone (LG Nexus 4)
    • Android 4.4.2 (Kit Kat, API Level 19)

• Preview Encrypted Email:
  • 76 ms to render plaintext on L-7.5
  • Well within expected response time (50 – 150 ms)

• Composing and Sending Encrypted Email:
  • Used Enron Email Dataset
  • With longest email:
    • 953 words, of which 362 are unique
    • 205 ms to encrypt, build the search index, and encode
Limitations

- Social engineering attacks (phishing)
- Only handles text-based apps
- TCA-logic update is required if the app updated the UI
A11y Attacks: On the Importance of checking the Authenticity of User I/O
Traditional User I/O Paths in OS
New User I/O Devices in OS – Accessibility (a11y)

• Voice commander
  • Receives user input from microphone

• Screen reader
  • Send UI output to a11y system as well as output display

• On-screen keyboard
  • Generates key clicks by software

• etc.
A11y Added New I/O Paths to OS

Click Continue!
A Malware Can Attack A11y

Click Continue!

Diagram:
- Assistive Technology
  - Process Input
  - Process Output
- OS
- A11y Library
- App
  - App Output
  - Input Handler
- Screen Output
- Regular Input Devices

Original I/O path
Alt. input through a11y

Play audio: click continue

You'll need to provide administrator permission to copy to this folder
Systray
Date created: 6/13/2013 8:28 PM
More details

Play audio:

Malware can attack A11y.
A Malware Can Directly Send Command

Click Continue!

Assistive Technology

Process Input
Process Output

OS

A11y Library

App

App Output
Input Handler

Screen Output

Regular Input Devices

A11y Input (Voice)

Original I/O path

Alt. input through a11y

Alt.
input
through
a11y

Listening

Destination Folder Access Denied

You'll need to provide administrator permission to copy to this folder

Sysprep
Date created: 6/13/2013 8:28 PM

Click Continue!
Security Implications of A11y

- Creates new I/O Paths
  - A11y allows a program to send an input event to the application
  - A11y allows a program to read an output of the other applications

A (malicious) program can pretend as a user if systems miss security checks on a11y inputs
Security Analysis for A11y

- **Objective**
  - Check OSes if they are secure under a11y attacks through new I/O path

- **Method**
  - Test security checks from the component
  - At assistive technology level (e.g., voice commander)
  - At OS level
  - At App level
At Assistive Technology (AT) Level

Authenticate the input

Is the voice from **real human**?  
If not, **machine** can access it!

Is the voice matched with **registered user**?  
If not, **any other human user** can access it!
Attacks on Voice Commander

**Siri’s Lockscreen Bypass A Growing Privacy Issue For iOS Users**

In less than 30 seconds, anyone with access to an Apple iPhone or iPad can extract a lot of personal data using Siri, Trend Micro says.

Security vendor Trend Micro has sounded the alarm once again on a continuing issue with Apple’s Siri digital assistant that lets anyone with physical access to an iOS device to interact with it and easily extract data even if the device is locked.

In a blog post today, security researchers from the company said it takes just 30 seconds for someone to extract names, phone numbers, and calendar entries -- or even post to a connected social media account -- from a locked iOS device using simple voice commands.

“A locked device should not disclose the owner’s identity and contact information, as well those of the owner’s friends, family, and contacts,” the researchers wrote. “Siri bypasses this and provides detailed information and other functions on a locked mobile device.”
Access control is required

**Assistive technology** allowed to access a11y?
If not, any program (possibly malware) can access it!
At Application Level

Distinguish User input from A11y Input
Do not allow to perform security sensitive UI!
Permission Views in iOS

A malicious app can click this!
A11y Output Have a Conflict with System Features

• Visual feedback as accessibility
  • **No tactile feedback** in touch-screen devices
  • To reduce *typo*, OS provides visual feedback
  • Assumes **only user** can see it
  • Existing feature breaks its security
    • Screenshot!
    • iOS6: Private API allows screenshot
    • Windows: no restriction at all
    • Android: screen recording permission
Attacks on Missed Checkpoints

• We found 12 new vulnerable points
  • Windows (3)
    • 2 Privilege escalation, 1 password leak
  • Linux (2)
    • Bypassing process boundary, password leak
  • iOS (4)
    • Bypassing sandbox and authentication
    • Privilege escalation, Password leak
  • Android (3)
    • Bypassing sandbox and authentication
    • password leak
No Authentication for Alternative Input

• Any user, or a program can send voice to Siri

• Simple authentication is not enough
  • Liveness check
  • Challenge-response

• Vendors cannot ignore practical issues
  • Computational power
  • Power consumption
  • etc.
Weak Access Control on A11y Libraries

- Windows: None
- Ubuntu: None
- OS X: None
- iOS 6: None
- Android: **User settings**
Compatibility Makes the Confusion

Event Layer

User Input

A11y Input

UI Internals

performClick()

Application
Recommendations for A11y

• Apply access control on a11y library
• Provide mechanism to distinguish a11y I/O from the real I/O requests
• For the security sensitive UIs, get input with proper authentication.
Bringing Assurance of User I/O in Intel SGX
Motivation & Problem

• Intel SGX
  • Provides a trusted execution environment only with hardware TCB
  • Security of an enclave is guaranteed even under the untrusted OS

• Challenges: SGX does not have any secure user I/O path
  • All I/O event must be handled by the untrusted OS
  • Most of existing works protect I/O through encryption
    • Haven, VC3, Ryoan, SGX-TOR, etc.
  • SGX-IO [CODASPY ‘17]
    • Requires a trusted hypervisor
SGX-USB Overview

• **Goal**
  - Establish a trusted channel between a USB port and an enclave program to securely support the USB device I/O in an enclave

• **Place a trusted hardware at the USB port (USB Proxy Device)**
  - Authenticates with an enclave (remote attestation)
  - Delivers USB packet through the secure communication channel

• **Enclave program**
  - Interprets USB packets (driver)
  - Processes I/O at the user-level
Threat Model

• Trust
  • The processor
  • The application that runs in an enclave
  • The remote attestation infrastructure
  • Building blocks for USB proxy device
  • Two public keys – from Intel and from Service Provider

• Do not trust
  • Do not trust OS and other applications

• Attackers do not have physical access to the devices
Architecture

- App Enclave
  - Device Driver
  - Encryption Engine
  - Attestation Engine
- Secure Application
- OCALL Layer
- Ethernet
- Remote Server
  - Enclave
    - App
    - Driver
  - Attest.
  - Encryption
- Remote Attestation Service Provider
- Intel Attestation Service

- USB Proxy Device
  - Encryption Engine
  - Attestation Engine
- USB Controller

- Trusted component
- Untrusted component
- Protected by TLS
- Trusted Path in SGX-USB
Remote Attestation Process

1. User initiates SGX-USB
2. Enclave sends msg0, msg1, and g_a
3. SP sends msg2, sigRL, and g_b
4. Enclave sends msg3 (Quote)
5. SP sends signed quote and verifiers (encrypted with g_a_b)
6. Enclave sends signed quote (g_a)
7. Send g_c and verifier (g_a_c)
8. Enclave sends verifier (encrypted with g_a_c)

RA END
A User’s Workflow

- If user wants to communicate with AuthMgr, then
  - Initiates RA process
  - The enclave authenticates with the RASP
  - USB Proxy Device authenticates with the enclave (shared g_A_C)
  - Display authentication information

After user allows opening the channel, then
- Enclave asks user a credential (e.g., password, OTP Token, etc.)
- Both user and enclave have authenticated
Use Cases

• Password manager & 2-factor authentication
  • User types passwords, and it only delivered to an enclave program
  • The program in enclave gets cookie and delivers the cookie to the application

• Secure video chat
  • Camera, microphone, display, speaker, etc.
Implementation

• Prototype of SGX-USB
  • AuthMgr: get password input from a keyboard
  • Raspberry Pi 3 for USB Proxy Device
  • Implemented USB HID driver for keyboard in enclave
  • The RASP server

• ~ 4700 lines of C++ code
Evaluations

- Throughput
- Latency
Discussions

• General I/O support
  • All USB devices. Possibly support devices through RDMA

• Feasibility of hardware implementation
  • Required logics: USB Host, Networking, and Crypto (AES & ECHDE)
  • Small firmware can drive the logics

• Availability
  • Inherent limitation of Intel SGX
Conclusion

• Building Trust in the User I/O in Computer Systems
• Integrity
  • Gyrus: authorizing outgoing network traffic using user input data
• Confidentiality
  • M-aegis: provide end-to-end encryption of user input/output by implementing encryption layer on top of the UI layer
• Authenticity
  • A11y attack: pointed out security problems caused by missing checks for the source of user input and the destination of system output
• Assurance
  • SGX-USB: establishing a secure channel between a USB device and an enclave to provide integrity, confidentiality, and authenticity of user input