モバイル端末における電力消費攻撃の試み

程 斌[1]　菊田 翼[1]　利光 能直[1]　齋藤 孝道[2]

概要：
モバイル技術の進化により，スマートフォンやタブレットには GPS やカメラなどの高度な機能が追加されてきた．また，これらの機能の利用は電力を大量に消費する．悪質なモバイルアプリがこれらの機能を無駄に使い，意図的に電力を消費することで端末へ悪影響を与え得る．本論文では，Android に搭載されている機能の内，特に電力の消費が多い機能を複数用いて，意図的に電力を消費する手法を調査し評価した．その結果，モバイル端末の消費電力を故意に増やし，使用可能時間を通常より短縮させる攻撃が可能であることがわかった．

キーワード：モバイル, Android, 電力消費攻撃

Attempt to Battery-Draining Attack on Mobile Devices

Bin Cheng[1]　Tsubasa Kikuta[1]　Yoshinao Toshimitsu[1]　Takamichi Saito[2]

Abstract: With the evolution of mobile technology, advanced functions such as GPS and cameras have been added to smartphones and tablets. Also, the use of these functions consumes a large amount of electric power. A malicious mobile application can use these functions in vain, and intentionally consume electric power to adversely affect the device. In this paper, we investigated and evaluated a method of intentionally draining a battery by using multiple functions that consume a lot of electric power in Android. As a result, we found that it is possible to execute an attack that intentionally increases the power consumption of the mobile devices and shortens the battery life than usual.

Keywords: Mobile, Android, Battery-draining Attack

1. はじめに

モバイル技術の普及により，誰でも気軽にスマートフォンを利用できるようになった．その普及率は 2019 年において，83.4%(総務省『令和 2 年版情報通信白書』[13] より) であり，生活必需品の 1 つになりつつある．

しかし，モバイル技術の発達に伴い，GPS やカメラなどの電力を大量に消費する高度な機能が追加されてきた．これらの機能を長時間利用すると，モバイル端末の使用可能時間が短縮してしまう．また，悪質なモバイルアプリがこれらの機能を無駄に長時間使い，意図的に電力を消費することで端末へ悪影響を与える攻撃（以降，電力消費攻撃と呼ぶ）の可能性がある．

本論文では，Android 端末に標準搭載されている機能の内，CPU，ディスプレイ，無線通信機能，メディア関連機能などの，特に消費電力が多い機能を複数用いて，意図的にエネルギーを消費するためのプログラムを作成し，電力消費攻撃を試みた．その結果，電力消費攻撃が可能であることが確認できた．また，この攻撃は特別な権限を必要とする機能を使わないため，対策が困難であることがわかった．

2. 関連研究

2.1 ハードウェアの消費電力に関する研究

坂本ら [10] は，Android 端末における，出力 RGB 値と消費電力および出力 RGB 値と照度の関係を調査した．そ

1 明治大学大学院
Graduate School of Meiji University
2 明治大学
Meiji Univevrsity

の結果に基づいて，RGB 値を制御することにより消費電力を削減する手法を提案した．調査結果から，実験端末において，明るさの調整値と出力の照度や消費電力はほぼ比例していることがわかった．

小川ら [11] は，Android 端末の性能と，処理に要する総消費電力の関係を調査した．その結果，JIT や VFP などの高速化手法と CPU クロック周波数を下げることが総消費電力の低下に有効であることがわかった．

2.2 ソフトウェアによる電力消費に関する研究

早川ら [12] は，Android 端末においてアプリが無操作時の振る舞いが電池消費量に及ぼす影響を調査した．そこで，多くの場合ユーザやアプリが意図せず発行しているブロードキャストインテントに着目し解析を行った．ブロードキャストインテントとは，Android OS がスリープ中に特定の作業を実行するために発行される，システムとアプリの間を橋渡しするものである．その結果，モバイル端末を 24 時間放置した状態において調査した結果，電池消費量と相関の高いブロードキャストインテントを明らかにした．さらに，WiFi 通信を許可した場合，端末において 29%のバッテリ消費量がみられ，電力消費の要因の一つとわかった．

中村ら [14] は，無操作状態の Android 端末における電力消費に着目し，WakeLock の実行回数に基づいた無操作時電力消費の原因となるアプリの特定手法を調査した．その結果，WakeLock の実行回数が多いアプリを削除することが，消費電力の低減に有効であることがわかった．

3. 関連知識

3.1 電力消費が多いハードウェアや機能

CPU

モバイルデバイスにおいて主な計算は CPU が行う．大量な高負荷計算を行うことにより，電力の消費量が増加する．また，CPU の利用率を見ることによって，その使用状況を確認することができる．通常，CPU の利用率が高いほど，電力の消費量も多い．

ディスプレイ

モバイル端末には主に，液晶ディスプレイ (LCD) と有機 EL が利用されている．LCD の液晶分子は，有機 EL のように画素ごとの発光ができないので，スクリーン全体を照らすバックライトが必要である．また，LCD は液晶分子が電圧印加の作用で回転し，光源からの光を部分的に遮ったり透過させることで表示を行うので，画面表示を変更する際にも電力が消費される．そして，ディスプレイは端末と利用者の最も重要な情報伝達手段で長時間使われるので，電力消費も多い．

無線通信機能

モバイル通信だけでなく，Wi-Fi, Bluetooth, GPS といった無線通信も大量な電力消費の原因である．そして，これらは通信の頻度や通信時の電波の強度によって電力消費量が変わる．

メディア関連機能

モバイル端末には多様なファンクションを実現するために，フラッシュライト，カメラ，スピーカーなどのメディア関連のハードウェアが搭載されている場合がある．これらの機能は常に継続動作する機会は少ないが，長時間使用されると，バッテリー減少の原因にもなる．

3.2 Android 端末の省電力モード

Android OS には端末の消費電力を抑えるために，いくつかの省電力モードが搭載されている．これらにより，端末上実行するアプリの状態を制御し，電力の無駄使いを防止できる．本論文では，故意に電力を消費するために，以下の省電力モードへの遷移を全て WakeLock（3.3 で説明）の使用により回避した．

アプリスタンバイモード

Android 6.0 から追加された省電力機能であるアプリスタンバイモード [2] は，一時的に使用されていない非アクティブな状態のアプリをターゲットとして検知する．そして，そのアプリにおけるバックグラウンドでのネットワークアクテビティを制限し，電力消費を抑える．また，使用中のアプリはこの影響を受けない．

スリープモード（CPU アイドル）

Linux カーネルレベルの省電力モードであり，一般的には，スリープモードと呼ばれている．プロセススケジューリングに基づき，CPU の仕事がなくなる場合，システムは低消費電力状態に入り，CPU を停止させる，もしくはその動作周波数を下げる．また，CPU と共に，CPU に依存する一部の機能やサービスもオフまたは制限する．つまり，CPU をアイドル状態にして，電力を節約する．

Doze モード（ディープスリープモード）

Android 6.0 から，スリープモードよりも電力を節約できる Doze モード [2] が追加された．Doze モードの動作を図 1 に示す．

このモードでは，Android 端末が長時間使用されていない時，CPU やネットワークなどの高負荷の機能とサービスを制限し，ディープスリープすることにより電力消費を抑える．Doze モードに入る条件は，モバイル端末が電源に接続せず，画面がオフのまま静止状態で，一定時間経過することである．また，システムは定期的に Doze モードを一時停止し，「メンテナンスの時間枠」と呼ばれる期間で，アプリで保留されたアクティビティを続行する．

Doze モード中に制限される機能は以下の通りである．

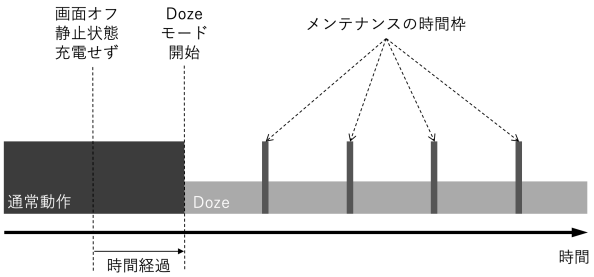

図 1 Doze の概念図

- ネットワークアクセスを切断する
- WakeLock を無視する
- 標準 AlarmManager アラームを次のメンテナンスの時間枠に保留する
- Wi-Fi スキャンを停止する
- 同期アダプターの実行を禁止する
- JobScheduler の実行を禁止する

3.3 WakeLock

WakeLock とは，Android OS が提供する，システムが省電力モードに入らないようにするための機能である．アプリが WakeLock を利用している限り，3.2 で紹介した Android 端末の省電力モードへの遷移を回避できる．なお，WakeLock は標準搭載の機能であるため，ルート権限の解除などの特殊手段を使わなくても利用可能である．WakeLock は，タイムアウト式ロックと通常式ロックの 2 種類に使い分けられている．タイムアウト式ロックは事前に設定したタイムアウト時間を経過すると自動的にロックが解除される．それに対し，通常式ロックは手動で解除する必要がある．WakeLock の実際の使用方法は 5.1 で後述する．

4. 提案手法

3.1 で説明した多くの電力を消費する主な機能の内，複数選択した．また，それらを用いて意図的に電力を消費する手法を考案した．さらに，考案した手法を用いて電力を消費するためのプログラムを作成した．本節では，それらの手法について解説する．

4.1 WakeLock による省電力モード回避

電力を意図的に消費させるには，デバイスをスリープモードなどの省電力モードに遷移させないことが有効である．そこで，WakeLock を用いて省電力モードに遷移するのを妨げ，端末を常時稼働させる．

4.2 CPU による電力消費

電力消費を増加させるために，CPU に大量な複雑計算をさせ，CPU 利用率を向上させる．

また，計算アルゴリズムとしてハッシュ値計算を利用した．ハッシュ値計算は大量に複雑な計算を行い，多数の CPU 性能評価ツールにおいて負荷を発生させる手段として使われている．

4.3 ディスプレイによる電力消費

モバイル端末のディスプレイにおいて，LCD のバックライトの明るさは電力消費と関係がある．そこで，Android 端末の画面の明るさを最大化することで，大量に電力を消費させることが可能だと考えた．

また，素早く画面表示を切り替え，全ての画素に対して，液晶分子を大幅かつ素早く回転させることが，電力消費に有効だと考えられる．

4.4 無線通信による電力消費

Wi-Fi や Bluetooth などの無線通信を利用する際，大量に電力が消費される．Android OS では省電力モードなどを通して，アプリからネットワークへのアクセス回数を減少させることで，電力消費を抑えている．そこで，あえて無線通信の回数を増やすことで，電力消費量を増加させる．

4.5 メディア関連機能による電力消費

以下のメディア関連機能を継続的に動作させることで，大量の電力消費が可能だと考えられる．

- フラッシュライトを点滅，または常に点灯させることによる電力消費
- スピーカーを常に最大音量にして，音声を流すことによる電力消費
- 振動モーターを利用することによる電力消費

他にも幾つかのハードウェアや機能による方法を考案したが，事前に試したところ，電力消費への効果はあまり著しくなかった．そこで，本節で紹介した以上の機能だけを用いて，意図的に電力を消費するプログラムを作成した．

5. 電力消費アプリ

本節では，提案手法に基づいて実装した，意図的に電力を消費する Android アプリ（以下，電力消費アプリと呼ぶ）について解説する．

電力消費アプリは以下の各部分によって構成される．また，各構成部分は個別にスレッドを立ち上げ，お互いに影響なく，オン・オフすることができる．

5.1 WakeLock の実装

WakeLock[3] を利用するには，android.os.PowerManager.newWakeLock(int flags, String tag) を呼び出すことにより，新たな PowerManager.WakeLock クラスのインスタンス（新しい WakeLock）

が生成される．その際には，第一引数の”int flags”として渡されるフラグを選び，CPU，ディスプレイ，キーボードのバックライトの動作パターンを指定する必要がある．指定可能なフラグおよび対応している WakeLock の動作パターンを表 1 に示す．

表 1 WakeLock のタイプを指定するフラグ

フラグ (int flags)	CPU	ディスプレイ	キーボード
PARTIAL_WAKE_LOCK	オン	オフ	オフ
SCREEN_DIM_WAKE_LOCK	オン	暗くする	オフ
SCREEN_BRIGHT_WAKE_LOCK	オン	明るくする	オフ
FULL_WAKE_LOCK（API レベル 17 より廃止）	オン	明るくする	明るくする

WakeLock を使用する際，PowerManager.WakeLock.acquire() を呼び出すことでリクエストを行う．acquire() の引数を表 2 のように指定することで，WakeLock をタイムアウト式ロックか通常式ロックかを使い分けることができる．

表 2 acquire() のタイプ

引数タイプ	WakeLock 種類	解除方式
acquire()	通常式ロック	release() の呼び出しによる手動解除
acquire(long timeout)	タイムアウト式ロック	timeout(msec) 時間経過後に自動解除

本論文では，ディスプレイを継続表示させる SCREEN_BRIGHT_WAKE_LOCK，および CPU の動作だけを保持する PARTIAL_WAKE_LOCK を選んで 2 種類の通常式 WakeLock を実装した．

5.2 電力消費機能の実装

CPU

CPU を大量計算をさせ，負荷を発生させる．
本論文では，Github 上で公開されている CPU 負荷テストツール ElephantStress[1] を利用した．このツールは，同時に複数のスレッドを立ち上げ，MD5 値の計算を平行で行う．これにより，CPU 利用率を高く保つことで電力消費を実現する．

ディスプレイ

本論文では，ディスプレイの明るさを最大に設定して，LCD のバックライトまたは有機 EL の画素自体による消費電力の最大化を実現する．また，フルスクリーンで表示する画面の色を 10ms 毎に白黒反転させて，LCD の液晶分子を最大限かつ高速に回転させることで電力消費を実現する．

Wi-Fi

周囲の Wi-Fi ネットワークの検出を行う API[4] である WifiManager.startScan() を呼び出すことで，Wi-Fi スキャンを実行することができる．本論文では，5 秒間隔のループによって実行を繰り返して，Wi-Fi スキャンの頻度を必要以上に上げることで電力消費を実現する．

Bluetooth

周囲の Bluetooth の検出を行う API[5] である BluetoothAdapter.startDiscovery() を呼び出すことで，周囲の Bluetooth デバイスをスキャンし始める．本論文では，Wi-Fi と同様に，5 秒間隔のループによって実行を繰り返して，Bluetooth スキャンの頻度を必要以上に上げることで電力消費を実現する．

LED フラッシュライト

Android OS のカメラ関連機能の API[6] である CameraManager.setTorchMode(String cameraId, boolean enabled) を呼び出すことで，LED フラッシュライトを利用することができる．具体的には第二引数の boolean enabled に True または False を指定して LED フラッシュライトをオンまたはオフに切り替えできる．本論文では，10ms 間隔のループによる LED フラッシュライトを点滅させることで電力消費を実現する．

スピーカー

Android OS のメディア関連機能の API[7] である ToneGenerator.startTone (ToneGenerator.TONE_PROP_BEEP) を使用して，ビープ音を生成して再生できる．引数の ToneGenerator.TONE_PROP_BEEP には Android OS が提供しているビープ音の種類 [8] の中から指定する必要がある．本論文では，スピーカーで継続的な音を流すことで電力消費を実現する．

振動機能

Android OS の振動機能の API[9] である VibrationEffect.createWaveform(long[] timings, int[] amplitudes, int repeat) を使用して継続的な振動を生成できる．本論文では，振動モーターを継続的に回転させることで電力消費を実現する．

これらの機能は全て Android OS に標準搭載される API を利用して作成したため，当手法は特殊な権限を必要としない．

6. 実験

本節では，意図的に Android 端末の電力消費を行う実験について説明する．

6.1 実験環境

実験に利用する 2 台の Android 端末については以下（表 3，表 4）の通りである．

表 3 実験で使用する端末 1

CPU	Qualcomm Snapdragon 660 (2.2 GHz)
RAM	6GB
ディスプレイ	2280x1080 (有機 EL)
OS	Android 9
バッテリー	3,350 mAh (4.2V)

表 4 実験で使用する端末 2

CPU	MediaTek Helio P70 (2.1GHz)
RAM	6GB
ディスプレイ	2160x1080 (LCD)
OS	Android 9
バッテリー	4,220 mAh (4.2V)

Android 端末が消費する電力量を，端末に USB 回路計を接続し直接測定する．この測定方法は，Android の API を利用してバッテリー残量から平均消費量を割り出す方法に比べ，より正確な数値を観測することが可能である．また，バッテリーへの物理的アクセスが困難なため，今回の実験では USB 回路計を使用した．今回の実験で使う USB 回路計では，端末での瞬間的な電流 (A)，電圧 (V) 以外に，電力 (W) も表示され，その数値を記録する．

6.2 実験手順

本論文では 2 台の Android 端末に対し，以下の様に実験を行う．

(1) **電力消費アプリのインストール**
5 節で説明した電力消費アプリを ADB を通して，それぞれの端末にインストールする．

(2) **端末の充電を満タンにする**
正確な測定を行うために，各実験を行う前に充電をし，バッテリーが満タンな状態にする．

(3) **USB 回路計の接続**
USB 回路計と外部電源を実験用端末に接続する．USB 回路計端末は外部電源からの電力消費量を計測する仕組みなので，外部電源に繋ぐ必要がある．そうすると，端末は自動的に充電し始め，充電による電力消費も計測値に含まれてしまう．その部分を排除して，正確な電力消費量を測るためには充電が停止するまで待つ必要がある．そのため，必ず端末をフル充電し，充電が停止していることを確認した上で次のステップに進む．

(4) **電力消費アプリの実行**
全てのアプリが実行されていないことを確認した上で，事前にインストールした電力消費アプリを実行する．

(5) **ディスプレイ以外の各機能ごとの電力消費量の測定**
電力消費アプリの各電力消費機能を実行して電力消費量を測る．まずは，ディスプレイ以外の各機能ごとに対して測定を行う．具体的には，以下の表 5 に示している項目を参照し，各機能の起動停止をコントロールするボタンを押して電力消費アプリの動作パターンを切り替え，USB 回路計で表示された値を記録する．測定は，各測定項目に対し，開始直後，30 秒後，1 分後の計 3 回行い，その平均値を記録する．

表 5 ディスプレイ以外の測定項目

測定項目	説明
待機状態	ディスプレイをオフにし，全てのアプリが実行されていない状態
CPU 負荷テスト	CPU に負荷を掛けている状態
Wi-Fi スキャン	周囲の Wi-Fi ネットワークを探している状態
Bluetooth スキャン	周囲の Bluetooth デバイスを探している状態
フラッシュライト点灯	フラッシュライトが継続点灯している状態
フラッシュライト点滅	フラッシュライトが高速点滅している状態
振動機能	振動モーターが継続回転している状態
スピーカー	スピーカーが最大音量で音を流している状態

測定時はその他の機能からの影響を受けないため，測定項目以外の機能を停止させ，ディスプレイをオフにする．また，CPU，メモリ，センサーなどの停止できないハードウェアに対して，出来る限り他のアプリやサービスで実行されないようにする．

(6) **ディスプレイの電力消費量の測定**
(5) と同様な測定方法で，ディスプレイの電力消費量の測定を行う．ただし，ディスプレイの消費電力を測定する際，明るさ調整及び白黒反転それぞれの効果を評価したいため，表 6 に表した 3 つのディスプレイの表示パターンそれぞれに対して測定を行う．

表 6 ディスプレイの測定項目

測定項目	説明
画面明るさ最大	ディスプレイの明るさが最大で，白の静止画面を表示している状態
画面点滅 (明るさ最小)	ディスプレイの明るさが最小で，全画面を高速白黒反転表示している状態
画面点滅 (明るさ最大)	ディスプレイの明るさが最大で，全画面を高速白黒反転表示している状態

また，ディスプレイ以外の機能を全て停止させ，出来

る限り他のアプリやサービスを実行しないようにする．

(7) **全機能稼働中の電力消費の測定**

2 台の端末それぞれに対し，電力消費アプリの全機能を起動させ，その際の合計電力消費量を測定する．ただし，ディスプレイの項目については，明るさ最大で全画面白黒反転する状態を選択した．

(8) **全機能稼働中のバッテリー持ち時間の測定**

2 台の端末それぞれに対し，フル充電であることを確認する．そして，電力消費アプリの全機能を稼働させた状態で外部電源を抜き取り，バッテリーが切れるまでの持ち時間を測定する．

7. 実験結果

2 台の端末において，電力消費アプリでの各項目ごとの電力消費量を測定した結果を表 7 に示す．

表 7 各機能による消費電力

機種名	端末 1	端末 2
待機状態	0.5W	0.4W
CPU 負荷テスト	5.1W	4.3W
Wi-Fi スキャン	1.5W	1.2W
Bluetooth スキャン	0.9W	1.1W
フラッシュ点灯	0.8W	1.2W
フラッシュ点滅	1.0W	1.5W
振動機能	0.8W	1.1W
スピーカー	1.8W	2.5W
画面明るさ最大	1.9W	1.7W
画面点滅 (明るさ最小)	0.9W	1.0W
画面点滅 (明るさ最大)	2.0W	2.3W
全機能稼働	5.8W	6.5W

調査の結果，最も電力を消費する方法は CPU に負荷をかけることだと確認できた．また，ディスプレイの明るさを最大にすると，明るさ最小の状態に比べると倍以上の電力を消費していることがわかった．

2 台の端末において，電力消費アプリの全機能を稼働させた際のバッテリー持ち時間を表 8 に示す．

表 8 全ての機能が使われる時バッテリーの持ち時間

機種名	機種 1	機種 2
持ち時間	3.1h	3.2h
バッテリー容量	3,350 mAh (4.2V)	4,220 mAh (4.2V)

2 台の実験端末いずれも，日常的な使用場面における経験上の持ち時間（使用方法にはよるが，約 8〜12 時間）と比べ，より大幅に短縮していたとがわかった．

8. 考察

8.1 全機能稼働中のバッテリー持ち時間について

それぞれの端末の，バッテリー電力量 $W_p(Wh)$ および推定持ち時間 $T(h)$ について，

$$W_p = Q \times U$$

$$T = W_p \div P$$

という式が成り立つ．ここで，$Q(Ah)$ はバッテリー容量，$U(V)$ はバッテリー電圧，$P(W)$ は全機能稼働中の消費電力を表す．

この式を用いて，それぞれの端末における推定バッテリー持ち時間を計算する．

端末 1 の場合

$$W_{p1} = 3,350mAh \times 4.2V \div 1000 = 14.07Wh$$

$$T_1 = 14.07Wh \div 5.8W = 2.4h$$

端末 2 の場合

$$W_{p2} = 4,220mAh \times 4.2V \div 1000 = 17.72Wh$$

$$T_2 = 17.72Wh \div 6.5W = 2.7h$$

それぞれを表 8 の結果と比べると，実際の測定したバッテリーの持ち時間は推定値より長いことが明らかである．

その差について，8.2 節で考察する．

8.2 CPU による電力消費について

各端末における推定持ち時間と持ち時間の差の理由を明らかにするため，CPU のみに負荷を掛ける実験を追加で行った．その結果，CPU 利用率と周波数は計測開始直後高く上昇したが，途中から大幅な上下の変動を繰り返してることがわかった．また，実験中に端末の温度が大幅に上昇したため，システムの熱制限により CPU の性能が低下し，測定した実際の持ち時間と推定持ち時間が異なっていたのだと考えられる．

8.3 ディスプレイによる電力消費について

表 7 から，有機 EL（端末 1）と LCD（端末 2）での電力消費の違いがわかる．有機 EL は画素自体の発光を部分的に制御することで，省電力を実現している．明るさ最大での画面点滅の実験では，約半分の時間で黒い画面が表示された．よって，有機 EL はその間全画素が発光しないため，LCD よりその間の消費電力を抑えることができる．しかし，白い画面を表示している間は，有機 EL の方が LCD より多く電力を消費していた．さらに，有機 EL が白い画面を表示している間に消費した電力は，LCD と比べ約半分の時間にも関わらず，LCD の消費電力と大差はなかった．

また，明るさ最小の際，LCD のバックライトは低電力であるため，有機 EL ディスプレイとの差は見られなかった．

以上により，本論文におけるディスプレイによる電力を消費する手法は，有機 EL と LCD 両方に有効だと確認できた．

8.4 攻撃の可能性

本論文の実験で，通常 8〜12 時間ほどのバッテリー持ち時間を，端末 1 では 3.1 時間に，端末 2 では 3.2 時間に，短縮できた．使用可能時間を半減できたので，考案した攻撃手法は効果的だと言える．

これらの手法を利用し，正常なアプリに組み込むことで，利用者の端末に対して，故意に必要以上な電力を消費させることができる．また，これらの提案手法による攻撃は，ルート権限を必要とせず，特定の脆弱性に基づいてないため，実行が容易である．しかし，今回の実験はモバイル端末の各機能による電力消費を確認することが主要目的のため，ディスプレイの点滅や，LED フラッシュライトの点滅など，利用者に容易に気付かれる機能も利用した．攻撃という面で考えると，CPU に負荷を掛けたり，人間に聞こえない周波数の音声を流し続けるなど，気付かれずに攻撃を行うことも十分可能であると考えられる．

8.5 今後の課題

実験の結果，考案手法は効果的だと確認できたが，改善できるポイントはいくつかある．

今回の実験では正確な瞬間電力を測定することで，端末の各機能による電力消費量を測定した．しかし，CPU の熱制限による性能低下により，測定した全機能稼働中でのバッテリーの推定持ち時間と，実際の持ち時間との間に差が生じた．今後は，この問題を解決するための方法を再考し，精度を向上させることが一つ目の課題である．

また，実際に攻撃が行われると想定した際に，利用者に気付かれないよう，機能や実装方法を変えることが二つ目の課題である．

9. 研究倫理

本論文では，意図的に電力を消費させるプログラムを作成し，それを用いて，モバイル端末に対し一連の実験を行った．これらの実験の目的は，電力消費攻撃の可能性を評価することである．作成した電力消費アプリおよびそのソースコードは，研究以外の目的には使用せず，研究室にて厳重に管理している．

10. おわりに

本論文では Android に搭載されている機能の内．特に電力の消費が多い機能を複数用いて，意図的に電力を消費する手法を調査し評価した．その結果，実験用の 2 台の Android 端末におけるバッテリーの利用可能時間を，それぞれ 3.1 時間，3.2 時間に短縮させることができた．また，モバイル端末を大きく電力消費させる最も有効な手段は，CPU に負荷をかけることだと明らかになった．以上より，電力消費攻撃が可能であることがわかった．

謝辞 本研究の一部は JSPS 科研費 JP18K11305 の助成を受けたものです．

参考文献

[1] https://github.com/SyncHack/ElephantStress.
[2] Google. Documentation for app developers. https://developer.android.com/training/monitoring-device-state/doze-standby?hl=ja.
[3] Google. Documentation for app developers. https://developer.android.com/training/scheduling/wakelock?hl=ja.
[4] Google. Documentation for app developers. https://developer.android.com/guide/topics/connectivity/wifi-scan.
[5] Google. Documentation for app developers. https://developer.android.com/guide/topics/connectivity/bluetooth.
[6] Google. Documentation for app developers. https://developer.android.com/reference/android/hardware/camera2/CameraManager#setTorchMode(java.lang.String,\%20boolean.
[7] Google. Documentation for app developers. https://developer.android.com/reference/android/media/ToneGenerator#ToneGenerator(int,\%20int).
[8] Google. Documentation for app developers. https://developer.android.com/reference/android/media/ToneGenerator#constants_1.
[9] Google. Documentation for app developers. https://developer.android.com/reference/android/os/VibrationEffect.
[10] 寛和 坂本, 優太 中村, 駿 野村, 真太郎 濱中, 実靖 山口, and 亜樹 小林. Android 端末における照度と消費電力の関係を考慮した読みやすさの低減を抑えたディスプレイ消費電力の低減. Technical Report 10, 工学院大学大学院工学研究科電気・電子工学専攻, 工学院大学大学院工学研究科電気・電子工学専攻, 工学院大学大学院工学研究科電気・電子工学専攻, 工学院大学工学部情報通信工学科, 工学院大学工学部情報通信工学科, 工学院大学工学部情報通信工学科, jan 2015.
[11] 小川寿人 and 山口実靖. Android 端末における、演算性能の評価と消費電力の測定. **第 74 回全国大会講演論文集**, 2012(1):145–146, mar 2012.
[12] 早川愛, 磯村美友, 竹森敬祐, 山口実靖, and 小口正人. Android 端末におけるアプリケーションごとの消費電力の解析に関する一考察. In **第 76 回全国大会講演論文集**, volume 2014, pages 61–62, mar 2014.
[13] 総務省. 令和 2 年版情報通信白書, 2020. https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r02/pdf/index.html.
[14] 優太 中村, 愛 早川, 文乃 小柳, 明大 半井, 敬祐 竹森, 正人 小口, and 実靖 山口. Android os におけるアプリケーションの起動による電力消費に関する一考察. In **第 77 回全国大会講演論文集**, volume 2015, pages 255–256, mar 2015.