Computer Security Symposium 2019
21 - 24 October 2019

プログラミング言語に対する ホモグリフ攻撃の実現可能性評価

鈴木 宏彰[1,a)]　米谷 嘉朗[2]　森 達哉[1,3]

概要：現代のプログラミング言語は国際化が進み，データのみならず，プログラミング言語の構文要素としても ASCII (ISO/IEC 646) で定義される文字集合「以外」の文字を使える規格が増えている．それらの規格では，変数名，関数名，あるいはクラス名などの識別子としてユニコードで定義された様々な文字を使うことができる．プログラミング言語で使用できる非 ASCII 文字集合の中には，ASCII アルファベットと外見上の見分けのつかない文字が存在する．例えば ASCII 文字集合に含まれる ‘a’ に対する，キリル文字の ‘ａ’ はその一例である．このように外見が類似した文字をホモグリフと呼ぶ．プログラムの変数名や関数名を構成する ASCII 文字を対応するホモグリフに置換すれば，ソースコードの外見を変えずに異なる挙動を示すプログラムを生成することができ，新たな脆弱性となる可能性がある．本研究は，このような背景のもと，非 ASCII 文字を識別子として用いることができるプログラミング言語，エディタや開発環境におけるユニコードで構成される構文要素の表示状況，実際に非 ASCII 文字を識別子として用いているプログラムの例を調査する．また，非 ASCII 文字のホモグリフを悪用した「ホモグリフ攻撃」の実現可能性をユーザスタディにより評価する．

キーワード：ホモグリフ，プログラミング，ユニコード，ユーザスタディ

On the Homoglyph Attack against Programming Languages

Hiroaki Suzuki[1,a)]　Yoshiro Yoneya[2]　Tatsuya Mori[1,3]

Abstract: Modern programming languages are becoming increasingly internationalized. Because of that, many programming languages can define unicode variable name and function name. There are some unicode characters that are indistinguishable from alphabet. For example, for the ‘a‘ (U+0061), the Cyrillic‘a‘ (U+0430) is very similar. Characters that have similar shapes are called homoglyphs. Then, by replacing a single alphabetic character of a program variable name or function name with its corresponding homoglyph, it can be changed to a program that behaves differently without changing the appearance of the source code. In this paper, we consider attack scenarios using such homoglyph features and verify the feasibility of such attacks and, analyzed the github repository to see how much of the source code has Unicode variables. We also employ user study to evaulate the feasibility of the homoglyph attack.

1. はじめに

数あるプログラミング言語において，変数名や関数名などの識別子を表現するのに ASCII (ISO/IEC 646) で定義される文字集合，いわゆる「半角英数字」を使うことが一般的である．一方，現代における多くのプログラミング言語は国際化が進み，データのみならず，プログラミング言語の構文要素としても ASCII 文字集合「以外」の文字を使える規格が増えている．それらの規格では，変数名，関数名，あるいはクラス名などの識別子としてユニコードで定義された様々な文字を使うことができる．このような国際化は，英語以外の様々な言語によるプログラミングの要求に答えるものである．実際，Pawelka ら [14] は少なから

1　早稲田大学 (Waseda University)
2　日本レジストリサービス (Japan Registry Services Co., Ltd.)
3　情報通信研究機構 (NICT)
a)　sz0213hk@nsl.cs.waseda.ac.jp

ぬ数のオープンソースや産業ソフトウェアにおいて，コメントや識別子で非英語の利用がみられることを報告している．また，Guo [9] はコーディングスタイルが英語の識別子利用を強要する問題を指摘している．

今日代表的な符号化文字集合であるユニコードでは，Unicode 標準附属書 UAX #31 において，プログラミング言語の識別子として利用可能な文字集合を定めており [29]，Python3 などのプログラミング言語のコーディング規約で採用されている [18]．UAX #31 で識別子としての利用が推奨されている文字集合には，視覚的に見分けがつかない文字の組み合わせが存在する．例えば ASCII に相当するユニコード対応のコードブロックである基本ラテン文字に収録される ‘a’ に対し，キリル文字の ‘ａ’ (U+0430) は外見が非常に似通っている．このような外見が似ている文字のことをホモグリフ (homoglyph) と呼ぶ．ホモグリフを悪用したセキュリティ脅威としては IDN ホモグラフ攻撃が良く知られているが [1,12,24]，本研究ではプログラミング言語におけるホモグリフの悪用に着目する．プログラムの識別子を構成する ASCII 文字を対応するホモグリフに置換すれば，ソースコードの外見を変えずに異なる挙動を示すプログラムを生成することができ，新たな脆弱性となる可能性がある．

本研究は，上述した問題意識のもと，以下の Work Package (**WP**) に取り組む．

WP1: 識別子として非 ASCII 文字を使えるプログラミング言語の調査．代表的なエディタや IDE における非 ASCII 文字識別子の表示方法調査

WP2: GitHub レポジトリを対象とした識別子として非 ASCII 文字を用いたプログラムの調査

WP3: ホモグリフを悪用した攻撃の実現可能性評価（ユーザスタディ）

WP1 は，各言語の規約や文献の調査，および実環境を用いた調査を行う．**WP2** は，代表的な言語の例として Python3 を対象とし，Github のレポジトリの大規模調査を行う．収集した Python コードに抽象構文木解析を適用し，識別子に非 ASCII 文字を使った例を調査する．**WP3** はホモグリフを悪用した攻撃として，コードを難読化する攻撃を想定し，開発者がそのような難読化に気がつくことができるかをユーザスタディにより評価する．

以下は **WP1**–**WP3** を通じて得られた知見のハイライトである．

- 代表的な 20 のプログラミング言語のいずれもが非 ASCII 文字による識別子をサポートすることを明らかにした．
- 代表的な 5 つの IDE/エディタのいずれもが非 ASCII 文字で構成された識別子に対して特別な修飾をせずに表示していることを明らかにした．
- Github において，176,373 の Python コードを含む 4,531 件のレポジトリを解析した結果，非 ASCII 文字を識別子として利用するコードが存在したが，その数はきわめて少ないことを明らかにした．
- 25 人の参加者によるユーザスタディの結果，プログラミング経験が浅い開発者はホモグリフの存在に気が付かない傾向が高いこと，プログラミング経験が長い場合も，必ずしも正確にホモグリフを理解しているわけではないことを明らかにした．

本研究は，プログラミング言語におけるホモグリフ攻撃の問題に取り組んだ初の研究であり，ホモグリフを用いた高度な攻撃方法の検討や，対策技術の開発など，さらなる研究の必要性と今後の指針を明らかにした．

本論文の構成は以下の通りである． 2– 4 章では，それぞれ **WP1**–**WP3** の詳細と結果を示す． 5 章では，本研究の制約事項ならびにホモグリフ攻撃に対する対策について論じる． 6 章で関連研究をまとめ， 7 章で結論を述べる．

2. WP1: プログラミング言語･エディタ/IDE の調査

本章は **WP1** の詳細と，調査結果を示す．

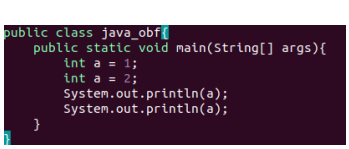
2.1 プログラミング言語の調査

文献 [28] にリストされている 20 の代表的なプログラミング言語を対象とし，識別子として非 ASCII 文字集合（ユニコード）が利用可能であるか，利用可能な場合はその文字集合の範囲について調査した．表 1 に結果を示す．表中の U+0080 などの 16 進数はユニコードのコードポイントを示している．ASCII 領域 (Basic Latin) は除外した．いくつかのプログラミング言語において，`U+0080`～`U+FFFF` の「基本多言語面」(BMP）の文字集合をサポートしていることがわかる．BMP はラテン文字から CJK 統合漢字まで，幅広い文字を収録している．unicode category は個々のユニコード文字に割り当てられているカテゴリである．Lm (Letter mofifier) 等，省略名で表示されているカテゴリ名は文献 [7] などを参照されたい．UAX #31 [29] は前述の Unicode 標準附属書 Unicode Standard Annex #31 で定めたプログラミング言語の識別子として利用可能な文字集合である．ZWNJ (zero width non-joiner) / ZWNJ (zero width joiner) はゼロ幅非接合子/ゼロ幅接合子であり，アラビア文字などで合字を扱うために用いる制御文字である．

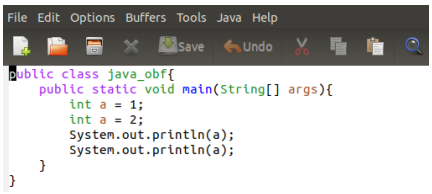
調査の結果，対象とした 20 個のプログラミング言語すべてが，識別子として非 ASCII 文字を許可していることがわかった．また，使用できる文字をユニコードにおけるカテゴリ名で細かく指定しているもの，基本多言語面あるいは拡張領域も含めた非常に広い範囲の文字の利用を許可しているもの，仕様書には陽に書かれていないが，非 ASCII 文

表 1　プログラミング言語と unicode 識別子の対応

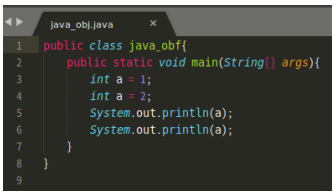
プログラミング言語	識別子のユニコード対応	識別子として利用可能な非 ASCII 文字の範囲 (ユニコード)
JavaScript [4]	✓	UAX #31 [29], ZWNJ , ZWJ
Python3 [18]	✓	UAX #31 [29]
Java [10]	✓	U+0080~U+FFFF (基本多言語面)
C++ [2]	✓	U+00A8~U+EFFFD (拡張領域含む．範囲多数)
Swift [25]	✓	U+0080~U+FFFF (基本多言語面)
TypeScript [27]	✓	範囲不明 (仕様には載ってないが非 ASCII 文字が使える)
Go [8]	✓	unicode category (Letter, Nd) [7]
SQL [13]	✓	U+0080~U+FFFF (基本多言語面)
Ruby [20]	✓	範囲不明 (仕様には載ってないが日本語や絵文字などが使える)
R [19]	✓	範囲不明 (仕様には載ってないが非 ASCII 文字が使える)
PHP [16]	✓	U+0080~U+00FF (ラテン 1 補助)
Perl [15]	✓	unicode category (Letter, Nd) [7]
Kotlin [17]	✓	範囲不明 (仕様には載ってないが非 ASCII 文字が使える)
C# [3]	✓	unicode category (Lu, Ll, Lt, Lm, Lo, Nl) [7]
Rust [21]	✓	範囲不明 (rust-lang RFC2457 に記載あり [22])
Scheme [11]	✓	unicode category(Lu, Ll, Lt, Lm, Lo, Mn,... 他多数), ZWNJ, ZWJ
Erlang [6]	✓	U+0080~U+00FF (ラテン 1 補助)
Scala [23]	✓	U+0080~U+FFFF (基本多言語面)
Elixir [5]	✓	UAX #31 [29], unicode category (Lu, Ll, Lt, Lm, Lo, Nl, Mn, Mc, Nd, Pc) [7]
Haskell [26]	✓	unicode category (Lu, Ll, Lt) [7]



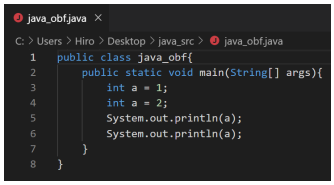
Vim



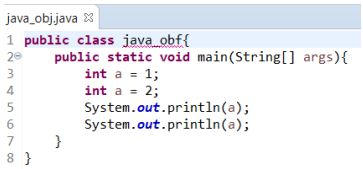
Emacs



Sublieme Text



Visual Studio Code



Eclipse

図 1　非 ASCII 文字を使用した時の IDE やエディタのスクリーンショット．最初の `int a=1;` の `a` は非 ASCII 文字のホモグリフであり，次の `int a=2;` の `a` が ASCII 文字である．

字を使える言語などが存在した (「範囲不明」). ユニコードコンソーシアムが策定した UAX #31 を使用している言語は必ずしも多数派ではなく，独自に定めた範囲の文字集合を許可している言語が多く見られた．識別子として使える非 ASCII 文字の範囲が最も狭いのは PHP と Erlang の「ラテン 1 補助」であり， ISO/IEC 8859-1 で定められる `0x80` から `0xFF` までの文字を含む．ラテン 1 補助は制御文字の他，西ヨーロッパで使用されるアクセント付きアルファベットを含む．アクセント付きアルファベットは例えば ‘ö’（ウムラウト）などであり，これは ‘o’ のホモグリフとして知られている．

2.2 エディタ・IDE の調査

代表的なエディタ・IDE として，Vim，Emacs，Sublime Text，Visual Studio Code，Eclipse を対象とし，非 ASCII 文字で構成される識別子がどのように表示されるかを調べた．その結果，調査対象としたすべてのエディタ・IDE において，非 ASCII 文字は ASCII 文字と同じ表示であり，両者を区別することはできなかった．この結果は，識別子として非 ASCII 文字を悪用する攻撃を解析する際に，これらのツールではそのことに気が付きにくいことを示唆する．各エディタ・IDE の表示のスクリーンショットを**図 1** に示す．Sublime Text や Visual Studio Code や Eclipse においては変数をクリックなどで選択すると同じ変数がハイライトされた．すなわち，ホモグリフが存在すると，その識別子はハイライトされない．これらのツールでは，ホモグラフ変数に気が付きやすい可能性がある．

3. WP2: Github レポジトリの調査

本章は **WP2** の詳細と，調査結果を示す．はじめにデータの収集・分析方法を示す．次に，分析結果を示す．

3.1 データ収集・分析方法

3.1.1 Python コード/レポジトリの収集

Github レポジトリで公開されている Python コードを収集するために，Github が提供している API を使用する．API に対して日付やプログラミング言語などを指定したクエリを発行すると，マッチするレポジトリの情報を JSON 形式で得ることができる．本調査ではユーザ数が多いプログラミング言語の代表例として Python を調査対象とする．他言語の調査は今後の課題である．API の制約事項として，1 つの検索条件につき得られる検索結果が最大 1,000 件という条件がある．本調査では日付の検索条件を 2018 年の 1 月 1 日から 2019 年の 7 月 1 日までの毎月 1 日以降と指定し，合計で 19 回の検索を行った．その結果，合計で 4,531 件の python のレポジトリの情報を収集した．収集した JSON ファイルにはレポジトリのファイルをローカルの環境にダウンロード（クローン）するための URL が存在する．収集した URL と git コマンドを用い，収集した全レポジトリのクローンを行った．クローンには合計で約 14 時間の時間を要した．

表 2 収集したレポジトリの統計

	数
レポジトリ数	4,401
すべてのファイル数	1,205,909
Python ファイル数	176,373
抽象構文木生成エラー数	12,402

表 3 収集したファイルの拡張子 (Top 5)

順位	拡張子	ファイル数
1	.jpg	263,187
2	none	202,237
3	.py	176,373
4	.png	115,286
5	.txt	90,108

3.1.2 識別子の取得・判定

コードに含まれる識別子を取得する方法として，本研究は抽象構文木 (abstract syntax tree; AST) と呼ばれる木構造のデータ構造を用いる．抽象構文木は，プログラミング言語の動作に関係しない情報を除外し，意味がある情報のみを取り出して抽象化したデータ構造であり，変数名，関数名，クラス名などの識別子に関する情報が含まれる．本研究は Python の標準ライブラリの一つである ast モジュールを用いて抽象構文木を抽出する．取得した識別子が非 ASCII 文字であるかの判定は，識別子を構成するすべての文字を先頭から順番に探査し，ASCII 文字であるかをチェックすることによって実現した．1 文字でも ASCII ではない文字を含む場合，その識別子を非 ASCII 識別子として抽出する．

3.2 分析結果

3.2.1 収集した github レポジトリの統計

表 2 に取得したレポジトリの統計を示す．git コマンドによるクローンでエラーが発生したレポジトリも存在した．またクローンにより作成されたディレクトリを全て調査し，ファイルの拡張子が「.py」になっているファイルを Python スクリプトであると判定した．表中のエラーはソースコードを解析して抽象構文木を取得できなかったケースである．その原因は，ast モジュールによる解析を Python3 を用いて行ったため，Python2 で書かれたソースコードを正しく解析できなかったケース，あるいは解析対象のファイルそのものにエラーが含まれていたため取得できなかったケースである．**表 3** に取得したレポジトリの拡張子種別ファイル数を示す．本研究では，拡張子 .py を持つファイルを Python ファイルとして抽出した．

3.2.2 変数名、関数名の分析結果

表 4 に収集したコードの内，非 ASCII 文字を識別子として使用しているファイルの統計を示す．合計で 17 個の

表 4　非 ASCII 文字の識別子を使っていた Python コードの統計

タイプ	レポジトリ数	ファイル数
変数名	10	17
関数名	2	2
クラス名	1	1

表 5　非 ASCII 文字で構成される識別子の例

タイプ	名前	使用用途	個数
変数名	left_Σ_T, left_μ_Q, left_σ_Q, right_Σ_T, right_μ_Q, right_σ_Q, vähim, vähima_paari_nimi, Σ_T, Σ_T_future, δ, ε, μ, μ_Q, μ_Q_future, π, σ, σ_Q, σ_Q_future, 一首诗, 参数, 变量, 变量 1, 变量 2, 平均分, 总和	数学	26
	X, ä, 蟒, Ⅴ	その他	4
関数名	乘, 减, 加, 除, 念诗函数	数学	5
クラス名	加 100 类, 幸运, 念诗类, 成绩单, 类, 类 A	数学	6

Python スクリプトが変数名に非 ASCII 文字を使っており，そのうちの 2 つが関数名にも非 ASCII 文字を使っていた．また，クラス名に非 ASCII 文字を使っているレポジトリが 1 つ存在した．抽出した識別子で使用されていた非 ASCII 文字を**表 5** に示す．合計で 41 個の非 ASCII 文字で構成される識別子が見つかった．発見した Python スクリプトの内容を見ると，ユニコードを使った識別子は数学記号として使われるケースが多かった．　数学以外の目的で使っていたファイルは，変数名としてユニコードを使った場合の動作確認をしていたものなどが中心的であった．その中にはラテン文字である x のホモグリフ文字の 1 つである，Ballot X (U+2717) が存在し，テスト目的で使われていた．非 ASCII 文字は中国語のケースが大多数であり，他にエストニア語，数学記号などが含まれていた．

以上の結果より，Python3 ではユニコードを識別子に利用することができるものの，それを実際に活用している事例は少ないことがわかる．このことは，開発者がそのような方法を積極的に利用しないことを望んでいる，あるいはそもそもユニコードを識別子として使えることを知らないことを反映していると考えられる．

4.　WP3: ホモグリフ攻撃の実現可能性評価

本章ではソースコードの場合のホモグリフ攻撃の概要と実施した実験の分析結果を示す．

4.1　ホモグリフ攻撃の概要

ホモグリフ攻撃の簡単な例として，ユーザからの入力を使用する web アプリケーションのソースコードに対する攻撃を示す．一般にユーザからの入力を使用する Web アプリケーションにでは，XSS や SQL injection に対する対策として，入力に対してエスケープ処理が行われる．

```
1 data = input("input :");
2 data = escape_func(data);
3 do_something(data);
```

攻撃者が上記のコードを含むレポジトリに対し，ホモグリフを用いた改変を行う攻撃を考える．攻撃者は対象レポジトリをクローン後，上記コードの 2 行目の escape_func 関数からの戻り値を代入する変数，data の変数名を構成する文字の一部をホモグリフに置き換える．この場合，3 行目の do_something 関数の引数にはエスケープ処理がされていないデータが入力されるため，XSS や SQL injection 対策を無効化することが可能となる．またこの時ホモグリフの置き換えだけでなく，他の箇所で新たな機能の追加なども行っておく．その後攻撃者は改変後のソースコードを対象レポジトリにプッシュし，レポジトリの所有者が改変に気づかずマスターにマージを行うことで攻撃が完了する．Github にはソースコードの差分を色をつけて表示する機能などもあるため，実際にこのような攻撃がどの程度成功するかは未知数である．またこのような攻撃の対象となる可能性があるレポジトリの調査は今後の課題である．

```
def fizzbuzz(a):
    for x in range(1,a+1):
        if x % 15 == 0:
            print("fizzbuzz")
        elif x % 3 == 0:
            print("fizz")
        elif x % 5 == 0:
            print("buzz")
        else:
            print(x)

if __name__ == "__main__":

    x = 100
    fizzbuzz(x)
```

図 2　オリジナルの FizzBuzz 問題

```
def fizzbuzz(a):
    for x in range(1,a+1):
        if x % 5 == 0:
            print("fizzbuzz")
        elif x % 15 == 0:
            print("fizz")
        elif x % 3 == 0:
            print("buzz")
        else:
            print(x)

if __name__ == "__main__":

    x = 100
    fizzbuzz(x)
```

はU+0445

図 3　識別子にユニコードを隠蔽した FizzBuzz 問題 (エラーあり)

4.2　実験方法

上述した通常の変数名とホモグリフを入れ替える攻撃に対し，開発者や解析者が気がつくことができるかという観点で実験を行う．題材として，FizzBuzz として知られるプログラミングの問題を用いる．FizzBuzz とは，1 から始まって加算されていく数字に対し，その数を 3 で割り切れたら “fizz”，5 で割り切れたら “buzz”，そして 15 で割り切れたら “fizzbuzz” と画面に出力する，基礎的なプログラミ

表 6　参加者の理解度

レベル	理解度
レベル 0	問題に不正解した
レベル 1	変数名の問題のみ指摘
レベル 2	文字コードの問題であることを指摘
レベル 3	ホモグリフを明示的に指摘

ングの問題である．この問題を Python で実装したコードが**図 2** である．オリジナルの FizzBuzz のコードを**図 3** で示すコードのように変更したコードを用意し，正しく動作する FizzBuzz コードに修正する問題を参加者に出題する．

コードの変更箇所は 2 点であり，1 つは，条件分岐の順番と内容である．これはある程度のプログラミング経験があれば，容易に修正が可能なものである．もう一つはホモグリフの挿入である．図 3 中の赤枠で囲った変数名に対し，ASCII 文字の x を キリル文字の x (U+0445) に変更した．このような変更により，上記の条件分岐を修正しても正しい出力にはならない．プログラムを正しく修正するためには，変数名に問題があることに気がつく必要がある．また，問題の所在を正しくつきとめ，根源的な対策を講じるためには文字コードに問題があることを理解する必要がある．以上の考察を含め，**表 6** に参加者の理解度を示す指標を示した．以下で示す質問項目に対する回答に応じて，参加者の理解度をコーディングする．コーディングは 2 名の研究者が実施した．

参加者に対する質問項目を以下に示す．参加者はプログラムを修正するタスクに取り組んだ後，質問に回答する．タスクに取り組むことができる時間の目安は最大 10 分間と設定した．

- 年齢を教えてください。
- プログラミング全般の経験を教えて下さい．
- Python3 の経験を教えて下さい．
- プログラムが間違っていた原因はなにですか？
- どのように解きましたか？
- 解くのにどの程度の時間がかかりましたか？

ユーザ実験は，メッセンジャーや SNS を通じて著者らの知人に参加依頼をした．**表 7** に参加者のデモグラフィック統計を示す．20 代を若者を中心に，比較的幅広い年齢層，およびプログラミング経験年数を持つ参加者 25 名に，ボランティアで実験に協力頂いた．

4.3　実験結果

表 8 にユーザ実験の結果をまとめた．参加者 25 名中，19 名が条件分岐の問題だけでなく，変数名の問題に気がついた．その内，文字コードに問題があることに気がついたのはレベル 2 およびレベル 3 の 13 名であった．レベル 0 は 1~3 分で解答したユーザが多く，原因の記述は条件分岐の問題のみを指摘したものが多かった．これらのケースでは，プログラムを実際に動作させておらず，またホモグリフによる問題点にまったく気がついていない可能性が高い．FizzBuzz 問題の場合プログラム実行結果が明白であるため，実行によって問題の所在に気がつくことが容易であるが，一般のプログラムの場合，ホモグリフを見逃す割合は更に高まると考えられる．

次にプログラミングの経験年数と理解度の関係を調べた結果を**表 9** に示す．プログラミングの経験が浅い (0–5 年) 開発者はレベル 0 が過半数を占めており，問題に気がつきにくい傾向にあった．反対に，経験年数が高い (10 年以上) の開発者は，レベル 2，3 のケースが多く，問題を良く理解していた．しかしながら，経験年数が高くてもホモグリフの問題であることを正確に回答できた開発者は少なかった (全体で 2 名のみ)．以上の結果をまとめると，ホモグリフ攻撃は特にプログラミング経験の浅い開発者に対して有効であること，また経験値が高い開発者であっても，ホモグリフの問題を正確に理解しているケースは少ないことを示唆している．なお，利用したエディタの種別は理解度に対して影響を与えていなかった．紙面の都合上，結果は割愛

表 7　実験参加者のデモグラフィック統計

項目	属性	人数
年齢	20 代	10
	30 代	3
	40 代	7
	50 代	4
	60 代以上	1
プログラミングの経験年数	0-5 年	5
	5-10 年	5
	10-15 年	6
	15 年以上	9
Python3 の経験年数	0-3 年	20
	3-5 年	5
	5-10 年	0
	10 年以上	0

∗ 複数のエディタを回答した参加者がいたため，合計人数は 25 にならない．

表 8　ユーザ実験の結果

解答時間	レベル 0	レベル 1	レベル 2	レベル 3
1 分	2	1	3	0
2 分	2	0	1	0
3 分	1	1	2	0
4 分	0	0	0	0
5 分	0	1	1	1
6 分	0	0	0	1
7 分	0	1	1	0
8 分	0	0	1	0
9 分	1	1	0	0
10 分	0	1	2	0
合計	6	6	11	2

表 9　プログラミング経験年数ごとの理解度

経験年数	レベル 0	レベル 1	レベル 2	レベル 3
0–5 年	3	0	2	0
5–10 年	1	4	0	0
10–15 年	1	1	2	2
15 年以上	1	1	7	0

する．

5. 議論

本章では本研究の制約事項，攻撃への対策，および研究倫理について論じる．

5.1 制約事項

本研究の **WP2** では Python3 を対象としたコード収集，解析を行ったが，他の言語に関する検証はできていない．調査対象の拡大，およびホモグリフ攻撃が実在するかの調査は今後の課題である．本研究の **WP3** では，ランダムな開発者に対する実験ではなく，著者らの知人に対して依頼したため，参加者に偏りが生じた可能性がある．また，実験でとりあげた題材である FizzBuzz はプログラムの入出力が明快かつ単純であり，最終的にホモグリフに気が付きやすい状況にあった．より多様な参加者を含む大きな母集団を対象としたユーザ実験の実施，および実践的なコードを題材としたユーザ実験の設計は今後の課題としたい．

5.2 対策

表 1 が示すように，各々のプログラミング言語が識別子として使うことを許可する文字集合の範囲は多様であり，必ずしも一貫性がない．UAX #31 は，一般にユニコード文字に基づく構文解析での扱いを標準化したものであるが，同標準が定める文字集合が，セキュリティ脅威—とりわけホモグリフ攻撃—の観点で妥当なものになっているかは十分な検証がなされていない状況にある．今後プログラミング言語側で必要となる根源的な対策は，プログラミング言語の仕様や規約において，識別子として利用可能な文字集合として何を選択し，何を排除するのがセキュリティ上適切であるか，十分に検討・議論を重ねること，そして得られた知見を標準化もしくはガイドラインとして定めることである．一方，開発側では，まず IDE やエディタにおける対策が挙げられる．図 1 に示したように，今日代表的な IDE・エディタにおいて，ユニコードを用いた識別子に対する特別な文字修飾や警告はない状況にある．ホモグリフ攻撃のリスクを回避するためには，ユニコードで構成された修飾子に対して，あるいは特に ASCII 文字に対するホモグリフとなっている文字が使われたときに，開発者に対してそれを知らせるメカニズムを確立することが必要である．そのような目的として，文献 [24] で提案されたホモグリフデータベースである SimChar を使うことができる．

5.3 研究倫理

本研究におけるユーザスタディでは，早稲田大学が設置する研究倫理オフィスが定める「人を対象とする研究に関する倫理規程」および同オフィスが提供するフローチャートに則り，実験参加者に一切の不利益が生じることがないよう，慎重に実験を設計した．具体的には，実験参加は強制ではなく任意であること，参加者に対する負荷をかけないように時間制限を設けたこと，そして実験は匿名で行い，個人情報を一切収集しないことを遵守した．

また，本研究で考察した新たな攻撃ベクトルは特定のアルゴリズムやシステムの脆弱性が対象となるものではなく，ホモグリフの悪用に起因する一般的な攻撃である．また，現時点において，プログラミング言語に対するホモグリフ攻撃は，脅威が顕在化した攻撃ではない．プログラミング言語を対象としたホモグリフ攻撃のリスクや対策を広く公開することにより，脅威が顕在化する前に対策を確立する利点が見込まれる．以上の洞察に基づき，本研究の発表は公益性にかなうものであると考える．

6. 関連研究

著者らの知る限り，プログラミング言語を対象としたホモグリフ攻撃に関する研究は本論文が初の試みであり，これまでの既存研究は存在しない．ホモグリフを用いたセキュリティ脅威としては，国際化ドメイン名 (IDN) の例がよく知られている．プログラミング言語と IDN ではターゲットが異なるが，攻撃が成功する根源となるアイディアは，ホモグリフの悪用による，人間にとって一見しただけでは見分けがつかない異なる「識別子」がもたらす脅威であり，技術の本質や，有効な対策技術に関して相互に通底するものがある．以下では，IDN ホモグラフ攻撃に関連する研究を示す．

Liu ら [12] は Alexa ランキングの上位 1,000 件と，別途収集した 140 万の IDN を画像処理によって比較することで，ホモグラフドメイン名を検出し，結果として 1,516 個の IDN ホモグラフドメイン名を発見した．Chiba ら [1] は大規模な IDN ホモグラフドメイン名の分析を行い，非英語のブランド名ターゲットとした IDN ホモグラフドメイン名が多数あること，およびいくつかの domain squatting を組み合わせた悪性ドメイン名が数多く存在することを明らかにした．Suzuki ら [24] は IDN として利用可能なユニコードの文字集合を探索し，ホモグリフとなる文字ペアを自動的に抽出する技術を開発した．またその技術を用いて，大規模なドメイン名分析を行った結果，未知の悪性 IDN ホモグラフドメイン名を多数発見した．

7. まとめ

本研究は，今日の多くのプログラミング言語において，変数名，関数名，クラス名といった識別子に非 ASCII 文字が使える事実を背景に，プログラミング言語に対するホモグリフ攻撃の実現可能性を評価した．調査の結果，特に人気が高い Top-20 のプログラミング言語のすべてが，識別子として非 ASCII 文字を使えること，代表的なエディタ・IDE では非 ASCII 文字で構成される識別子に特別な修

飾を適用しないことがわかった．これらの事実は，環境としてホモグリフ攻撃を実行する土壌が整っていることを示唆している．その一方で，GitHub レポジトリに存在する Python コードの分析では，識別子として非 ASCII 文字を用いたプログラムが存在するものの，その数はきわめて少ないことを明らかにした．さらにホモグリフによる難読化を施したコードを利用したユーザスタディを行った結果，25 人中，6 人は変数名に問題があることに気がつくことができなかった．変数名に問題があることに気がついた 19 名の内，問題の所在がホモグリフであることを正しく理解できたのは 2 人であった．また，プログラミング歴が浅い参加者ほど，ホモグリフの存在に気がつくことができない割合が高いことも判明した．

プログラミング言語に対するホモグリフ攻撃は，現時点では実世界で顕在化していない PoC 攻撃である．攻撃の脅威が顕在化する前に実現すべき課題として，UAX #31 などのフレームワークのセキュリティ評価を行うこと，および IDE やエディタなどの開発環境において，非 ASCII 文字で構成される識別子，あるいは検出されたホモグリフ文字に対して何らかの修飾を加える実装を進めることが挙げられる．

謝辞 プログラミング言語の国際化と歴史的経緯に関して有益な知見をご教示頂いた，筧捷彦名誉教授に感謝致します．また，ソフトウェア工学の観点から非英語話者のプログラミングに対する要求を調査した文献を多数ご教示頂いた，鷲崎弘宜教授に感謝致します．最後に，本研究の実験に快くご協力頂いた匿名の参加者の皆様に感謝致します．

参考文献

[1] Chiba, D., Hasegawa, A. A., Koide, T., Sawabe, Y., Goto, S. and Akiyama, M.: DomainScouter: Understanding the Risks of Deceptive IDNs, *Proc. the 22nd Int. Symp. on Research in Attacks, Intrusions and Defenses (RAID)* (2019).

[2] cppreference.com: Identifiers, `https://en.cppreference.com/w/cpp/language/identifiers`.

[3] Ecma International: C# Language Specification, `https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf`.

[4] Ecma International: ECMAScript 2019 Language Specification, `http://www.ecma-international.org/ecma-262/10.0/index.html#sec-identifier-names`.

[5] Elixil: Unicode Syntax, `https://hexdocs.pm/elixir/master/unicode-syntax.html`.

[6] Erlang: Introduction, `http://erlang.org/documentation/doc-5.9/doc/reference_manual/introduction.html#id73076`.

[7] FileFormat.info: Unicode Character Categories, `https://www.fileformat.info/info/unicode/category/index.htm`.

[8] Go: The Go Programming Language Specification, `https://golang.org/ref/spec#Identifiers`.

[9] Guo, P. J.: Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities, *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI)*, pp. 396:1–396:14 (2018).

[10] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith: The Java Language Specification Java SE 12 Edition, `https://docs.oracle.com/javase/specs/jls/se12/jls12.pdf`.

[11] John Cowan: r7rs, `https://bitbucket.org/cowan/r7rs/src/draft-10/rnrs/r7rs.pdf`.

[12] Liu, B., Lu, C., Li, Z., Liu, Y., Duan, H., Hao, S. and Zhang, Z.: A Reexamination of Internationalized Domain Names: The Good, the Bad and the Ugly, *Proc. IEEE/IFIP DSN 2018*, pp. 654–665 (2018).

[13] MySQL: 9.2 Schema Object Names, `https://dev.mysql.com/doc/refman/8.0/en/identifiers.html`.

[14] Pawelka, T. and Juergens, E.: Is This Code Written in English? A Study of the Natural Language of Comments and Identifiers in Practice, *Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME'15)* (2015).

[15] Perl 6 Documentation: identifier identifiers, `https://docs.perl6.org/syntax/identifiers`.

[16] PHP: Basics, `https://www.php.net/manual/en/language.variables.basics.php`.

[17] progmize: Kotlin Keywords and Identifiers, `https://www.programiz.com/kotlin-programming/keywords-identifiers`.

[18] Python Software Foundation: PEP 3131 – Supporting Non-ASCII Identifiers, `https://www.python.org/dev/peps/pep-3131/`.

[19] R: R Language Definition, `https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Identifiers`.

[20] Ruby: 字句構造，`https://docs.ruby-lang.org/ja/latest/doc/spec=2flexical.html`.

[21] Rust: The Rust Reference, `https://doc.rust-lang.org/reference/identifiers.html#identifiers`.

[22] rust-lang: Allow non-ASCII identifiers RFC2457, `https://github.com/rust-lang/rfcs/pull/2457`.

[23] Scala: Unicode Syntax, `https://hexdocs.pm/elixir/master/unicode-syntax.html`.

[24] Suzuki, H., Chiba, D., Yoneya, Y., Mori, T. and Goto, S.: ShamFinder: An Automated Framework for Detecting IDN Homographs, *Proc. of the 19th ACM Internet Measurement Conference (IMC)* (2019).

[25] swift 5.1: Lexical Structure, `https://en.cppreference.com/w/cpp/language/identifiers`.

[26] The Haskell 98 Report: Lexical Structure, `http://www2.informatik.uni-freiburg.de/~thiemann/haskell/haskell98-report-html/lexemes.html`.

[27] typescript: Documentation, `https://www.typescriptlang.org/docs/home.html`.

[28] UbuntuPIT: Top 20 Most Popular Programming Languages To Learn For Your Open-source Project, `https://www.ubuntupit.com/top-20-most-popular-programming-languages-to-learn-for-your-open-source-project/`.

[29] Unicode Consortium: Unicode Identifier and Pattern Syntax, `https://unicode.org/reports/tr31/`.