Rustで生成されたマルウェアに対するFLIRTの検証

川越 謙宏[1,a)]　小林 良太郎[1]

概要：近年，マルウェアがRustで作成されている．Rustのバイナリは多くの関数が静的リンクされるため，静的解析時に関数の区別がつかない．Hex-Rays社のIDAが使用しているFLIRTはライブラリ関数を識別する機能である．FLIRTは高い精度で関数を識別できるが，コンパイルオプションなどの状況によって，検知できない関数が存在する．本研究ではRustで生成されたマルウェアに対して，マルウェアが使用しているコンパイラのバージョンを特定し，コンパイラが使用している静的リンクライブラリから生成したFLIRTを使用してどの程度関数を識別できるかを検証する．

キーワード：Rust, 静的解析, マルウェア, FLIRT

Verification of FLIRT for Malware Generated by Rust

Akihiro Kawagoe[1,a)]　Ryotaro Kobayashi[1]

Abstract: Recently, malware is created using Rust. Since Rust binaries often have many functions which are statically linked, it is difficult if the functions are statically linked during static analysis. FLIRT, a function identification feature used by Hex-Rays' IDA, is used to identify library functions. FLIRT can identify functions with high accuracy, but there are functions which are not identified due to compiler options and other factors. In this study, we verify how well FLIRT can identify functions for malware generated by Rust by identifying the version of the compiler used by the malware and using the FLIRT generated from the statically linked library used by the compiler.

Keywords: Static Analysis, Rust, Malware, FLIRT

1. はじめに

マルウェアの作成にはほとんどの場合，C/C++, C#が用いられる．しかし，Go言語やNimなどそれ以外の言語が使用される場合もある [1]．新しい言語はアナリストが解析に慣れていないのと，解析手法が確立されていないため，解析が困難になる．また，アンチウィルスソフトなどの自動検知ロジックの回避などの目的でこれらの言語が用いられることがある．

Rustは，FireFoxの開発元であるMozillaが開発した言語であり，C/C++と同等のパフォーマンスを持ちながら，メモリの安全性が高いことが特徴である．最近ではマルウェアにも使用されている．Rustで生成されたバイナリの多くは，ライブラリ関数を静的リンクで埋め込むため，静的解析時にライブラリ関数とそうでない関数の区別が付けづらくなる．Hex-Rays社のIDA Pro[2]には，ライブラリ関数を自動的に識別する機能としてFast Library Identification and Recognition Technology (FLIRT) がある [3]．FLIRTは，.sigファイルに含まれている関数名と関数の先頭32バイトの一部にマスクをかけた値のペアを基に，ライブラリ関数を識別する機能である．RustでFLIRTを使用する際は，マルウェアが使用しているコンパイラのバージョンとCrateを特定する必要がある．Crateとは Rustで利用可能なライブラリのことである．Crateを活用することで一部の実装を省略できる．またFLIRTでの検知はRustのコンパイルオプションによって関数が検知できないことがある．

[1] 工学院大学
Kogakuin University, Shinjuku, Tokyo 163-8677, Japan
[a)] em25009@ns.kogakuin.ac.jp

本研究では Rust を使用しているマルウェアのファミリを調査し，そのマルウェアが使用しているコンパイラのバージョンと Crate を調査した．また特定した情報を基に生成した FLIRT シグネチャを使用してどの程度関数を識別できるかを検証した．本論文の構成は，まず，第 2 章で Rust の静的解析に関する技術や研究を述べ，その後第 3 章で本研究の調査方法を述べ，その後第 4 章で検証結果を示し，最後に第 5 章で考察を述べ，第 6 章で結論を述べる．

2. 関連研究

2.1 Rust で書かれたマルウェアの研究

Meghana らは Rust でつくられたマルウェアに対する現状について，系統的レビューを行った [4]．BlackHat, DEFCON, BSides などのカンファレンスで発表された研究や Kaspersky, Microsoft, Blackberry, McAfee, IBM などが公開した研究や ACM Digital Library, IEEEXplore, ScienceDirect, SpringerLink から調査した．2023 年時点では Rust のマルウェア解析やリバースエンジニアリングに直接関連する論文が発見されなかった．

また Zixu は Rust の最適化とリバースエンジニアリングについて調査した [5]．この研究ではコンパイル時の opt-level などの設定で，Code Flow や構造体などがリバースエンジニアリング時にどの程度復元できるかを調査した．全体的にスコアは低く，Rust に特化したリバースエンジニアリング手法が必要であると主張している．

2.2 RIFT

Microsoft は Rust バイナリ解析ツールとして RIFT を開発した [6]．RIFT は Rust のバイナリからコンパイラのコミットハッシュと Crate の抽出を行い，抽出した情報を基に FLIRT シグネチャを作成できる IDA Pro の拡張機能である．また FLIRT の生成以外にバイナリ比較を行うこともできる．

3. 調査方法

3.1 マルウェアの選定

Rust を使用しているマルウェアは様々あるが，実験的に作られた検体など一時的にしか存在しないマルウェアもある．それを考慮して，検証に使用する検体選定には，以下の基準を設けた．

- VirusTotal 上での Indicator of Compromise (IoC) が 2024 年以降に存在する
- IoC が少なすぎない
- Portable Executable (PE) または Dynamic Link Library (DLL) ファイルである
- 話題性

まず，継続性の観点から，2024 年以降に IoC が発見されていないファミリと IoC が少なすぎるファミリを除外した．そして，本研究では Windows 向けのマルウェアを対象としたため，mac OS や Linux 向けのマルウェアを除外した．例外的に，IoC が少ないものが RIFT の検証に使用されているファミリも選定した．上記の条件の基に，以下のファミリを選定した．

- Agenda
- ALPHV (BlackCat, Noberus)
- FunkLocker
- HiveLocker
- SPICA
- RALord
- SSLoad
- Akira_v2

3.2 コンパイラの特定

まず，検体が Rust を使用しているかの判別を行う．ファミリが同じでも，検体によって Go 言語製などの場合があるため，それを除外するために検体が Rust を使用しているかの判定を行った．今回，表層解析ツールの Detect It Easy (DIE)[11] を使用し，Rust を使用しているかの判定を行った．

バイナリからコンパイラを特定する場合，まずバイナリからコンパイラのコミットハッシュを取得する．コミットハッシュとは GitHub 上で管理されている Rust のリポジトリのコミットハッシュのことである．ほとんどの場合，Rust のコンパイラでコンパイルしたバイナリには，Strings として Rust のコンパイラのコミットハッシュが含まれている．図 1 はコンパイラからコミットハッシュを確認する

```
> rustc --version --verbose
rustc 1.88.0 (6b00bc388 2025-06-23)
binary: rustc
commit-hash: 6b00bc3880198600130e1cf62b8f8a93494488cc
commit-date: 2025-06-23
host: x86_64-pc-windows-msvc
release: 1.88.0
LLVM version: 20.1.5
```

図 1 コミットハッシュの確認
Fig. 1 Check Commit Hash

コマンド例である．特定したコンパイラが本当にマルウェアが使用しているものと同一であるか確認する際に用いる．Rust のコンパイラには stable 版と nightly 版がある．rustup toolchain install 1.80.1 のようにコンパイラのバージョンを直接指定した場合は stable 版のコンパイラがインストールされる．nightly 版は毎日更新されているコンパイラであり，build-std や trim-paths など nightly 版でしか使用できないコンパイルオプションが存在する．nightly 版の場合 rustup toolchain install nightly-2024-01-01 のようにバージョンではなく日付を指定してコンパイラをイン

ストールする．

バージョン特定の手法は 2 段階あり，まずコンパイラのバージョンを特定し，その後それが stable 版か nightly 版かを特定する．まず https://github.com/rust-lang/rust/blob/{commit_hash}/src/version からコンパイラのバージョンを特定する．その後 stable 版かを判断するために，同じバージョンのタグにアクセスする．タグ付けされているコミットハッシュと比較することで，stable 版か nightly 版かを判断する．もし nightly である場合，https://github.com/rust-lang/rust/commit/{commit_hash} からコミットされた日付を特定する．その日付を指定することでマルウェアが使用したコンパイラと同じバージョンのコンパイラを入手可能である．

3.3 Crate の特定方法

コミットハッシュと同様に，ほとんどの場合，使用している Crate も Strings として含まれている．本研究では，RIFT を用いて使用している Crate を特定した．

3.4 静的リンクされた関数の復元手法

ライブラリ関数の識別には 2 つの手法があり，1 つはシグネチャベースでの検知，もう 1 つはハッシュ値ベースでの検知である．シグネチャベースの検知には IDA Pro の FLIRT 以外に Vector35 社が開発したリバースエンジニアリングツール Binary Ninja[7] の sigkit がある [8]．ハッシュ値ベースでの検知では，NSA が開発した Ghidra[9] の Function ID や，Binary Ninja の WARP[10] がある．ハッシュ値ベースでの検知の場合 call のアドレスなど一部のオペランドにマスクをかけたのちにユニークな ID を生成する．Function ID は FNV1a ハッシュを用いており，WARP は GUID を作成している．本研究では検知率の高いシグネチャベースでの検知を採用したため，IDA Pro の FLIRT を使用した．シグネチャベースの場合，潜在的に偽陽性が発生する可能性があるが，柔軟な検知ができる．一方でハッシュ値ベースの場合，偽陽性が発生しにくいが，少しでも違うと検知できないため偽陰性が上がる特徴がある．本研究では Microsoft の RIFT を使用し FLIRT シグネチャを作成した．RIFT で抽出したコミットハッシュを RIFT のプロジェクト内にある rustc_hashes.json を使用しバージョンを特定する．コンパイラを特定後，rustup を用いてコンパイラをインストールし，コンパイラが使用している静的リンクライブラを取集する．Rust が使用している静的リンクライブラリは拡張子が.rlib であり，ar 形式のファイルである．展開すると Common Object File Format (COFF) 形式のファイルがいくつか生成される．収集した COFF ファイルを IDA が提供している FLIRT シグネチャ生成ツールの pcf.exe と sigmake.exe を用いて FLIRT シグネチャを作成する．

検証時点では RIFT は rustc_hashes.json に記載されていない nightly バージョンのコンパイラを特定できないため，バージョンを手動で設定できるようにコードを変更した．

4. 検証結果

本章では，それぞれの検体の検証結果を示す．以下の項目を結果に示す．

- VirusTotal での First Seen
- File Size
- DIE で特定した Time Date Stamp
- 特定したコンパイラのバージョン
- 使用している Crate の数
- 検体に含まれている関数の総数
- FLIRT で検知した関数の総数

VirusTotal での First Seen は UTC であるが，DIE での Time Date Stamp は JST である．コンパイラは Windows 版では MSVC 版と MinGW 版がある．末尾が gnu の場合は MinGW 版，msvc の場合は MSVC 版である．検体に含まれている関数の総数は，作成した FLIRT を適応すると関数の総数が増えることがあったため，適応後時点での関数の総数を示している．FLIRT で検知した関数の総数は，デフォルトのシグネチャで検知した関数を除外するために，FLIRT 適応前の自動検知した関数から FLIRT 適応後の自動検知した関数を減算した値とした．

4.1 Agenda

Loader と Ransomware を対象に検証を行った．

4.1.1 Loader

表 1 Agenda Loader の検証結果
Table 1 Verification Results of Agenda Loader

Item	Result
First Seen	2025-08-09
File Size	5.01 MB
Time Date Stamp	2025-08-03
Compiler	1.72.1-i686-pc-windows-gnu
Used Crates	4
Total Functions	648
Detected Functions	11

表 2 Agenda Loader が使用していた Crate
Table 2 Crate Used by Agenda Loader

Crate	Version
addr2line	0.21.0
gimli	0.28.0
object	0.32.0
rustc-demangle	0.1.23

表 1, 2 が Agenda Loader の検証結果である．バージョン 1.72.1 コンパイラのリリース日を調べたところ 2023-09-19 であった．使用している Crate については, rustc-demangle などのコンパイラが使用している Crate であった．

4.1.2 Ransomware

表 3 DLL 版 Agenda Ransomware の検証結果
Table 3 Verification Results of DLL Version Agenda Ransomware

Item	Result
First Seen	2025-08-01
File Size	4.68 MB
Time Date Stamp	2024-11-06
Compiler	1.72.1-i686-pc-windows-gnu
Used Crates	42
Total Functions	6581
Detected Functions	112

表 4 PE 版 Agenda Ransomware の検証結果
Table 4 Verification Results of PE Version Agenda Ransomware

Item	Result
First Seen	2024-06-29
File Size	1.58 MB
Time Date Stamp	2022-09-13
Compiler	nightly-2022-09-12-i686-pc-windows-gnu
Used Crates	22
Total Functions	2431
Detected Functions	112

DLL と PE に対して検証を行った．表 3 が DLL 版 Agenda Ransomware の検証結果であり，表 4 が PE 版 Agenda Ransomware の検証結果である．DLL は PE で使用していた Crate をすべて使用していることが判明した．Crate のバージョンは固定されていなかった．PE のコードの一部を使っている可能性が示唆される．

PE は Time Date Stamp から検体が少し古いものだと推測できる．また，特定したコンパイラ情報からこの頃の検体は日付指定をしない nightly コンパイラを使用していたと考えられる．表 1，3 から，最近の検体については，1.72.1 で固定されていると考えられる．

4.2 ALPHV

表 5 が ALPHV の検証結果である．Time Date Stamp から検体が少し古いものだと推測できる．nightly コンパイラを使用しており，作成した FLIRT シグネチャではほとんど関数を検知できなかった．使用している Crate が多く，whoami や walkdir など Crate を使用していた．このことからほとんどの処理を独自に実装せず，Crate に依存している可能性が高い．コンパイラが MinGW 版なので，クロスコンパイルすることを意識していると考えられる．

表 5 ALPHV の検証結果
Table 5 Verification Results of ALPHV

Item	Result
First Seen	2025-06-26
File Size	2.93 MB
Time Date Stamp	2021-12-11
Compiler	nightly-2021-12-04-i686-pc-windows-gnu
Used Crates	62
Total Functions	2253
Detected Functions	35

4.3 FunkLocker

表 6 FunkLocker の検証結果
Table 6 Verification Results of FunkLocker

Item	Result
First Seen	2025-01-05 22:08:08(UTC)
File Size	2.93 MB
Time Date Stamp	2025-01-06 07:01:03(JST)
Compiler	1.83.0-x86_64-pc-windows-msvc
Used Crates	58
Total Functions	14846
Detected Functions	10723

表 6 が FunkLocker の検証結果である．First Seen より Time Date Stamp が新しいが，これは First Seen が UTC なのに対して DIE が JST で表示しているからである．従って，JST を UTC に変換すると 2025-01-05 22:01:03 であり，First Seen と非常に近いことからアクター自身が VirusTotal に検体をアップロードした可能性が考えられる．コンパイラが msvc 版のものを使っているが，表層解析したところ文字列として pdb のファイルパスが含まれていた．MSVC 版の場合，pdb のパスがデフォルトで入るようになっており，消すためのコンパイルオプションを設定する必要がある．またファイルサイズに対して関数の数が多いことから，あまりコンパイルオプションを設定していなかったものと考えられ，検知率が高かったと考えられる．

4.4 HiveLocker

表 7 HiveLocker の検証結果
Table 7 Verification Results of HiveLocker

Item	Result
First Seen	2023-10-31
File Size	620 KB
Time Date Stamp	2023-10-26
Compiler	1.??.?-x86_64-pc-windows-gnu
Used Crates	Unknown
Total Functions	1477
Detected Functions	45

Hive Ransomware は 2023 年 1 月に FBI による妨害を受け，その後，Water Ouroboros (Hunters International) が後継であるとされている [12]．表 7 が HiveLocker の検証結果である．調査した検体には Strings として.rs のファイルパスが含まれていたがコミットハッシュや使用している Crate に関する情報が一切なかった．これらの情報は nightly で使えるオプションで消すことができる．そのため Time Date Stamp の情報を基に，使用しているコンパイラ nightly-2023-10-26-x86_64-pc-windows-gnu と仮定して，標準ライブラリのみの FLIRT シグネチャを作成し検証を行った．検知した関数に関しては，コンパイラが暗黙的に使用している compiler_builtins 関連のみであった．

4.5 SPICA

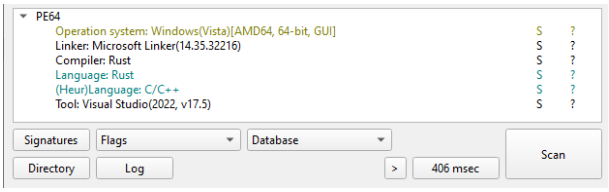

図 2 SPICA の DIE 解析結果
Fig. 2 DIE Analysis Results of SPICA

表 8 SPICA の検証結果
Table 8 Verification Results of SPICA

Item	Result
First Seen	2024-01-18
File Size	6.66 MB
Time Date Stamp	2023-09-19
Compiler	1.71.1-x86_64-pc-windows-msvc
Used Crates	69
Total Functions	16649
Detected Functions	1611

図 2 が SPICA の DIE での表層解析結果，表 8 が SPICA の検証結果である．DIE ではコンパイラが Rust と C/C++ の 2 種類検知された．メインの言語は Rust であったので一部の関数を C/C++で実装し，Rust の Foreign Function Interface (FFI) で呼び出している可能性が考えられる．コンパイラは MinGW ではなく MSVC を使用しているが pdb のパスは発見されなかった．また 1.71.1 のリリース日を調査したところ 2023-08-03 であり，その時点では比較的新しいバージョンのコンパイラを使用していた．

4.6 RALord

表 9 RALord の検証結果
Table 9 Verification Results of RALord

Item	Result
First Seen	2025-03-25 19:00:30 (UTC)
File Size	376 KB
Time Date Stamp	2025-03-25 01:57:12 (JST)
Compiler	1.84.1-x86_64-pc-windows-msvc
Used Crates	9
Total Functions	1199
Detected Functions	946

表 10 RALord が使用していた Crate
Table 10 Crate Used by RALord

Crate	Version
rand	0.8.5
rand_chacha	0.3.1
crossbeam-epoch	0.9.18
orion	0.17.9
rand_core	0.6.4
rayon-core	1.12.1
hashbrown	0.15.0
crossbeam-deque	0.8.6
rustc-demangle	0.1.24

RALord は 2025 年 3 月に初めて発見された．しかしそれ以降は発見されていないため IoC が 1 つしかない．表 9, 10 が RALord の検証結果である．First Seen と Time Date Stamp が非常に近いことから，マルウェア開発者自身が VirusTotal に検体をアップロードした可能性が考えられる．コンパイラバージョン 1.84.1 のリリース日を調査したところ，2025-01-30 であった．FLIRT で検知できた関数も多く，Strings として pdb のパスが含まれていたことから，実験的に作ったものの可能性も考えられる．

4.7 SSLoad

SSLoad は DLL と PE の 2 種類に対して検証を行った．

4.7.1 DLL

表 11 DLL 版 SSLoad の検証結果
Table 11 Verification Results of DLL Version SSLoad

Item	Result
First Seen	2024-05-03
File Size	446 KB
Time Date Stamp	2024-04-16
Compiler	1.77.2-i686-pc-windows-msvc
Used Crates	3
Total Functions	1323
Detected Functions	502

表 12 SSLoad が使用していた Crate
Table 12 Crate Used by SSLoad

Crate	Version
rustc-demangle	0.1.23
base64	0.21.7
serde_json	1.0.115

表 11 が SSLoad の検証結果，表 12 が SSLoad が使用していた Crate である．コンパイラバージョン 1.77.2 のリリース日は，2024-04-09 であった．使用している Crate が少ないが，winapi などの Crate を使わずに Windows API を呼び出し，アンチデバッグや C2 サーバに接続していた．

4.7.2 PE

表 13 PE 版 SSLoad の検証結果
Table 13 Verification Results of PE Version SSLoad

Item	Result
First Seen	2024-04-19
File Size	357 KB
Time Date Stamp	2024-04-03
Compiler	nightly-2024-02-16-i686-pc-windows-msvc
Used Crates	1
Total Functions	1101
Detected Functions	70

表 13 が PE 版 SSLoad の検証結果である．使用していた Crate は rustc-demangle のみであり，DLL 版と同様に使用している Crate が少なかった．また winapi などの Crate を使わずに Windows API を呼び出し，C2 サーバに接続していた．

4.8 Akira_v2

表 14 Akira_v2 の検証結果
Table 14 Verification Results of Akira_v2

Item	Result
First Seen	2024-06-24
File Size	544 KB
Time Date Stamp	2023-11-07
Compiler	nightly-2023-05-08-x86_64-pc-windows-msvc
Used Crates	9
Total Functions	1540
Detected Functions	307

表 15 Akira_v2 が使用していた Crate
Table 15 Crate Used by Akira_v2

Crate	Version
rand_core	0.5.1
threadpool	1.8.1
rustc-demangle	0.1.21
winapi-util	0.1.5
seahorse	2.1.0
time	0.3.28
curve25519-dalek	3.2.1
simplelog	0.12.1
termcolor	1.1.3

表 14, 15 が Akira_v2 の検証結果である．MSVC 版のコンパイラであるが pdb のパスは発見できなかった．このことから，耐解析用にコンパイルオプションを設定していると考えられるが，作成した FLIRT で検知できた関数が多かった．

4.9 まとめ

表 16 検証結果
Table 16 Verification Results

Family Name	Type	Detection Rate
Agenda	Loader	1.7%
Agenda	Ransomware (DLL)	1.7%
Agenda	Ransomware (PE)	4.6%
ALPHV	Ransomware	1.6%
FunkLocker	Ransomware	72%
HiveLocker	Ransomware	3.0%
SPICA	Trojan	9.7%
RALord	Ransomware	78%
SSLoad	Downloader (DLL)	38%
SSLoad	Downloader (PE)	6.4%
Akira_v2	Trojan	20%

表 17　使用率の高い Crate Top 10
Table 17　Top 10 Crate Used by Malware

Crate	Rate
rustc-demangle	100%
rand_core	67%
hashbrown	56%
rand	56%
once_cell	44%
smallvec	44%
serde_json	44%
base64	44%
aes	33%
time	33%
object	33%
cipher	33%
rand_chacha	33%
gimli	33%
miniz_oxide	33%
addr2line	33%

表 16 は検証結果をまとめたものである．検知率の低いファミリは他の検体でも検知率が低い傾向があった．表 17 が使用率の高い Crate Top 10 である．ランサムウェアは rand 関連の Crate を使用している傾向が高かった．また処理を高速化するために用いられていると考えられる hashbrown, smallvec などの使用率が高かった．他には非同期ランタイムの tokio や並行処理の crossbeam, parking_lot などの Crate が使用されていた．Windows API を扱える Crate は windows, winapi-util, crossterm_winapi の 3 種類を観測した．これらの Crate を使用しているマルウェアの割合は 44%であった．

5. 考察

検証結果から検知率が低い検体はコンパイル時に Link Time Optimization (LTO) を使用していると考えられる．LTO を有効にすると静的リンクで埋め込む関数が最適化されるので，その影響で検知率が大きく下がったものと考えられる．LTO が使用されていると考えられるファミリは Agenda, ALPHV, HiveLocker である．コンパイラのバージョンは同じファミリでも違うバージョンのものが使用されることもあるが，MSVC か MinGW かは同じであった．使用されている Crate に規則性がなかったが，Crate には Windows API を扱える windows, winapi などのように，同様のことを行える Crate が存在するので，それを考慮することで規則性が見える可能性がある．しかし HiveLocker や SSLoad のように Strings に使用している Crate の情報がない場合もある．Rust のマルウェア解析はバイナリ内の Strings の依存度が高いので今後，Strings に依存しない検知手法が必要になる可能性が考えられる．

6. 結論

本研究では Rust で生成されたマルウェアに対して，ライブラリ関数識別機能である FLIRT を使用し，どの程度関数を識別できるかを検証した．検証結果から Agenda, ALPHV, HiveLocker がコンパイル時に LTO を使用していると考えられ，検知率が低くなった．使用している Crate の傾向はあまりなかったが，類似する Crate をグループ分けすることで，マルウェア検知の指標に使える可能性がある．今後の課題として，Strings に依存しない Rust のバージョン検知，Crate の特定手法の検討，LTO が有効なバイナリに対するライブラリ検知手法の検討が挙げられる．

謝辞　本研究の一部は，JSPS 科研費 23H03396 の支援により行いました．また，検体の調査および収集に際し，株式会社サイバーディフェンス研究所にご協力いただきました．ここに深甚なる謝意を表します．

参考文献

[1] BlackBerry Ltd., “BlackBerry 2022 Threat Report,” https://www.blackberry.com/ (Accessed 2024-08-20).
[2] Hex-Rays, “IDA Pro,” https://hex-rays.com/ida-pro (Accessed 2025-08-18).
[3] Hex-Rays, “FLIRT,” https://docs.hex-rays.com/user-guide/signatures/flirt (Accessed 2025-08-18).
[4] M. Praveen and W. Almobaideen, “The Current State of Research on Malware Written in the Rust Programming Language,” Proceeding of the 11th International Conference on Information Technology (ICIT), pp. 266-270, 2023.
[5] Z. Zhou, “Decompiling Rust: An Empirical Study of Compiler Optimizations and Reverse Engineering Challenges,” https://arxiv.org/abs/2507.18792 (Accessed 2025-08-15).
[6] Microsoft, “RIFT,” https://www.microsoft.com/en-us/security/blog/2025/06/27/unveiling-rift-enhancing-rust-malware-analysis-through-pattern-matching/ (Accessed 2025-08-15).
[7] Vector35 Inc., “Binary Ninja,” https://binary.ninja/ (Accessed 2025-08-15).
[8] Vector35 Inc., “sigkit,” https://github.com/Vector35/sigkit (Accessed 2025-08-15).
[9] National Security Agency, “Ghidra,” https://github.com/NationalSecurityAgency/ghidra (Accessed 2025-08-15).
[10] Vector35 Inc., “WARP,” https://github.com/Vector35/warp (Accessed 2025-08-15).
[11] horsicq, “Detect It Easy,” https://github.com/horsicq/Detect-It-Easy (Accessed 2025-08-15).
[12] Trend Micro Inc., “ランサムウェア・スポットライト - Water Ouroboros（別名: Hunters International）,” https://www.trendmicro.com/ja_jp/research/25/c/ransomware-spotlight-water-ouroboros.html (Accessed 2025-08-15).