[RElog] Analiza działania i modyfikacja programu bez kodu źródłowego (sheadovas/artykuly/relog/analiza-i-modyfikacja-programu-bez-kodu-zrodlowego/)

Mar 05, 2016 / RElog (sheadovas/category/artykuly/relog/)

Deasemblacja i „naprawienie” programu przy użyciu instrukcji NOP.

Witam Was serdecznie w kolejnym *RElog'u*, czyli serii luźno powiązanej z Inżynierią Wsteczną. Dzisiaj zajmiemy się naprawą błędu pozostawionego przez roztargnionego programistę.

Chciałbym też przypomnieć, że ta seria nie jest serią poradnikami, a raczej serią pokazującą pewne rzeczy i mające raczej zachęcić do samodzielnego tematu. Jeżeli ktoś poszukuje poradników to z polskich źródeł polecam produkcje w wykonaniu *Gynvaela Coldwinda*[1].

Oczywiście, jeżeli znajdziecie jakiś błąd (może się zdarzyć, ekspertem w tej dziedzinie nie jestem), albo macie jakieś własne metodyki pracy, albo po prostu chcielibyście zobaczyć artykułów tego typu więcej to zapraszam do systemu komentarzy pod tym wpisem. Miłej lektury.

Wstęp

Załóżmy, że mamy do czynienia z plikiem binarnym programu, który coś robi (przyjmijmy, że to jakiś edytor tekstu); nie mamy jego kodu źródłowego więc analiza kodu i jego zmiana odpada, musimy zadziałać na tym co posiadamy, czyli plik [exe (Windows)] / [elf (Linux) (http://www.shead.ayz.pl/wp-content/uploads/2016/03/awesome-text-editor.zip)] (program to minimalna modyfikacja tworu z mojego *kursu Qt* [source (https://github.com/sheadovas/Find-and-Replace/tree/RElog)]).

Na nasze nieszczęście autor programu przy wydawaniu programu zapomniał o usunięciu całego kodu testowego informującego o wciśniętej kombinacji klawiszy. Pozostawiona kombinacja klawiszy: *[AltGr]+[L]* aktywuje *MessageBox* z komunikatem *„Hotkey detected!”* jednocześnie uniemożliwiając napisanie litery *„ł”* (co oczywiście w edytorze tekstowym jest niepożądanym zjawiskiem).

Załóżmy, że bardzo nam brakuje wspomnianej funkcjonalności, sam edytor jest na tyle zaawansowany (przypominam, że to założenie a nie fakt!) i przyjemny, że postanawiamy zająć się tym problemem. Na początku próbowaliśmy napisać do autora z prośbą o poprawkę, lecz ten nie odpisuje, a więc zostaliśmy „zmuszeni” zając się tym sami.

Analiza

Po tym wstępie fabularnym czas zająć się konkretami. W ramach pełnej jasności: korzystam z 64 bitowego *Linuxa* (*Mint*), a do deasemblacji korzystam z programu [Hopper (http://hopperapp.com/)] (dostępnego także w wersji darmowej), jednakże w przypadku tego zadania równie dobrze sprawdzi się open-source’owe narzędzie [radare2 (http://www.radare.org/r/)], a nawet powinien wystarczyć debugger, np *gdb*.

Oprócz tego przyda nam się również *HexEditor*, na Windowsie polecam [hxd (https://mh-nexus.de/en/hxd/)], a na Linuxie [wxHexEditor (http://www.wxhexeditor.org/)]. Jeżeli jesteście na Windowsie to fajnie jest ściągnąć *mingw* i podpiąć pod PATH narzędzie *strings.exe*, w przypadku Linuxów zazwyczaj jest to już zrobione, o ile posiadacie kompilator *gcc*.

Krok 1 – analiza programu „od zewnątrz”

Pod terminem: „od zewnątrz”, rozumiem analizę programu bez wchodzenia bezpośrednio w kod maszynowy. Warto zrobić ten krok, ponieważ on zazwyczaj mówi całkiem sporo (w bardzo trywialnych przypadkach już ten krok rozwiąże problem, albo prawie rozwiązuje).

Najbardziej podstawową rzeczą jaką możemy teraz zrobić, to użyć program *file* wchodzący w standardowy pakiet systemów opartych o Debiana (istnieje też port na Windowsa[2]).

file <nazwa_pliku> Shell

```
$ file awesome_text_editor
awesome-text-editor: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=924a4caf43c4af01b172e55e275ebf9ac2c0ba1c, not stripped
```

Z tego polecenia możemy się dowiedzieć, że jest to plik wykonywalny skompilowany pod 64 bitowe urządzenia, pod procesory x86-64, korzysta z dynamicznie linkowanych bibliotek, pod jądro Linuxa 2.6.24. W tym przypadku te informacje są średnio przydatne, ale zawsze wiemy nieco więcej.

Kolejnym bardzo przydatnym narzędziem jest *strings*. Dzięki niemu jesteśmy w stanie wyciągnąć stringi z pliku wykonywalnego w nieco przyjaźniejszej formie niż analiza za pomocą hex edytora.

Shell

```
$ strings awesome-text-editor > dump.txt
```

Należy zwrócić uwagę, że wciąż znajduje się w tym pliku wiele śmieci, które należy pominąć. Zachęcam do przeanalizowania zawartości rezultatu samodzielnie, a dopiero później do dalszej lektury tego artykułu.

dump.txt

```
/lib64/ld-linux-x86-64.so.2
W~g.;
1J?CI1
libQt5Widgets.so.5
_ZN11QFileDevice5closeEv
_ITM_deregisterTMCloneTable
_ZN11QTextStreamlsERK7QString
_ZNK7QString7indexOfERKS_iN2Qt15CaseSensitivityE
_ZN11QMetaObject10ConnectionD1Ev
_ZN7QString15fromUtf8_helperEPKci
_ZNK7QObject10objectNameEv
_ZN7QObject10childEventEP11QChildEvent
_ZN5QFile4openE6QFlagsIN9QIODevice12OpenModeFlagEE
_ZN7QObject10timerEventEP11QTimerEvent
_ZN7QObject11customEventEP6QEvent
__gmon_start__
_ZN7QString6numberEii
_ZN5QFileD1Ev
_ZN5QFileC1ERK7QString
_ZN12QKeySequenceD1Ev
_ZN10QArrayData11shared_nullE
_Jv_RegisterClasses
_ZN11QTextStreamD1Ev
_ZN7QObject7connectEPKS_PKcS1_S3_N2Qt14ConnectionTypeE
_ZNK11QMetaObject2trEPKcS1_i
_ZN7QObject16disconnectNotifyERK11QMetaMethod
_ZN12QKeySequenceC1Eiiii
_ZN7QObject11eventFilterEPS_P6QEvent
_ZN16QCoreApplication9translateEPKcS1_S1_i
_ZN7QObject13connectNotifyERK11QMetaMethod
_ZN10QArrayData10deallocateEPS_mm
_ZN7QString6appendERKS_
_ITM_registerTMCloneTable
```

Linia 4 dostarcza nam już potencjalnie ważnej informacji, mianowicie mówi nam, że program korzysta z biblioteki *QtWidgets*, po poszukaniu informacji w Internecie dowiadujemy się że jest to zestaw bibliotek C++ do tworzenia cross-platformowych aplikacji okienkowych [3]. Jednak to, że znaleźliśmy taki wpis to nie oznacza jeszcze, że ten program jest w całości napisany w tej bibliotece, może po prostu używać niektórych funkcji/modułów.

```
libQt5Widgets.so.5
```

Jednak wyrażenie Qt pojawiające się kilkakrotnie (z nazwami różnymi modułów) i te poniżej upewniają nas w pewności, że przynajmniej okienka zostały napisane w Qt. Świetnie, teraz wiemy z jakiej dokumentacji będziemy musieli korzystać.

```
/home/shead/Applications/Qt/5.5/gcc_64:/home/shead/Applications/Qt/5.5/gcc_64/lib
GLIBC_2.2.5
GCC_3.0
CXXABI_1.3
GLIBCXX_3.4
```

Dalej widzimy sekcję z tekstem widocznym w samej aplikacji oraz nazwy, które wyglądają jak nazwy funkcji/metod mających zareagować na odpowiednie zdarzenia, przy okazji widzimy okropny nawyk nazywania obiektów po polsku (wybaczcie, dawno pisałem tamten kurs i miałem okropne nawyki :/).

```C++
```

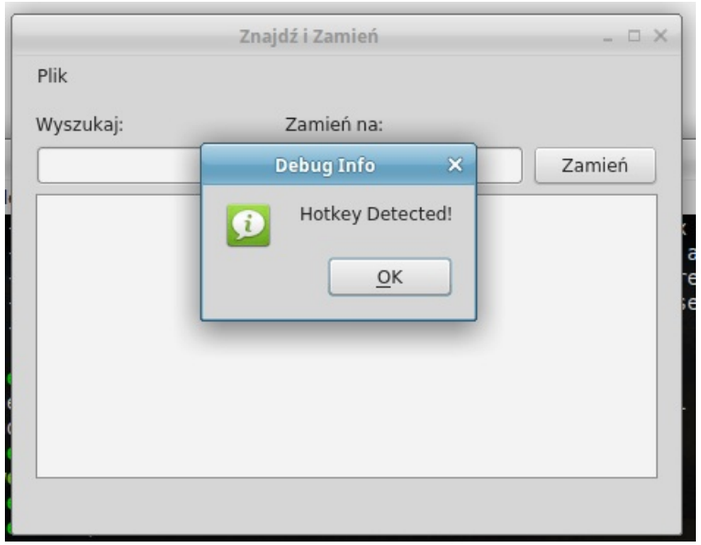
```
MainWindow
actionOtw_rz
actionZamknij
actionZapisz_jako
actionZapisz
centralWidget
gridLayout
label
wyszukaj
label_2
zamien_na
przycisk
poleTekstowe
statusBar
menuBar
menuPlik
Znajd
 i Zamie
Ctrl+O
Zapisz jako...
Ctrl+S
Wyszukaj:
Zamie
 na:
1onHotkeyDetected()
2activated()
Hotkey Detected!
Debug Info
Znaleziono
 wyraz
Raport
Pliki txt (*.txt)
/home/
```

Najbardziej interesującą linią w ramce powyżej jest ta ostatnia: *„onHotkeyDetected”*, wiemy jak nazywa się metoda wywołująca ten okropny MessageBox!

Dalej widzimy jeszcze jakiś tekst, ale jest on raczej mało istotny, przejdźmy do kolejnego kroku.

Krok 2 – analiza poprzez uruchomienie

Jeżeli wiemy, że program jest nieszkodliwy (a taki jest), to możemy go przeanalizować po prostu uruchamiając go i sprawdzając jak działa. My jednak to wiemy (bo to jest nasz ulubiony edytor tekstowy), to co jest warte uwagi to fakt, że stringi zauważone przez nas powyżej, powtarzają się i w samym programie, a po wciśnięciu skrótu AltGr + L wyświetla nam się MessageBox.



(https://i2.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/scrn-messagebox.png)

Zauważamy również, że nie jesteśmy w stanie pisać po wyświetleniu MessageBoxa, a także nie możemy zmienić focusu ponownie na pole tekstu o ile nie wyłączymy powiadomienia. Warto tutaj nieco poszukać informacji o samym Qt, abyśmy wiedzieli jakiego wywołania funkcji szukamy. Po chwili szukania znajdujemy dokumentację klasy [QMessageBox (https://doc.qt.io/qt-5/qmessagebox.html)].

Dla nas interesującą informacją jest to, że do wywołania funkcji potrzebne są minimum 3 argumenty: 2 stringi, które znamy (nagłówek: „Debug Info”, wiadomość: „HotkeyDetected”) oraz jakaś liczba, która jest wskaźnikiem na rodzica. Mając tyle informacji możemy spróbować zmierzyć się z problemem.

Krok 3 – analiza kodu maszynowego

W tym kroku chcemy znaleźć miejsce odpowiedzialne za uruchomienie samego MessageBox’a; sprawa (mimo że się taka nie wydaje) to jest dość trywialna, ponieważ miejscem stworzenia komunikatu jest też miejscem wrzucenia na stos znanych nam już stringów (w celu przekazania jako argumenty do funkcji tworzącej msgboxa).

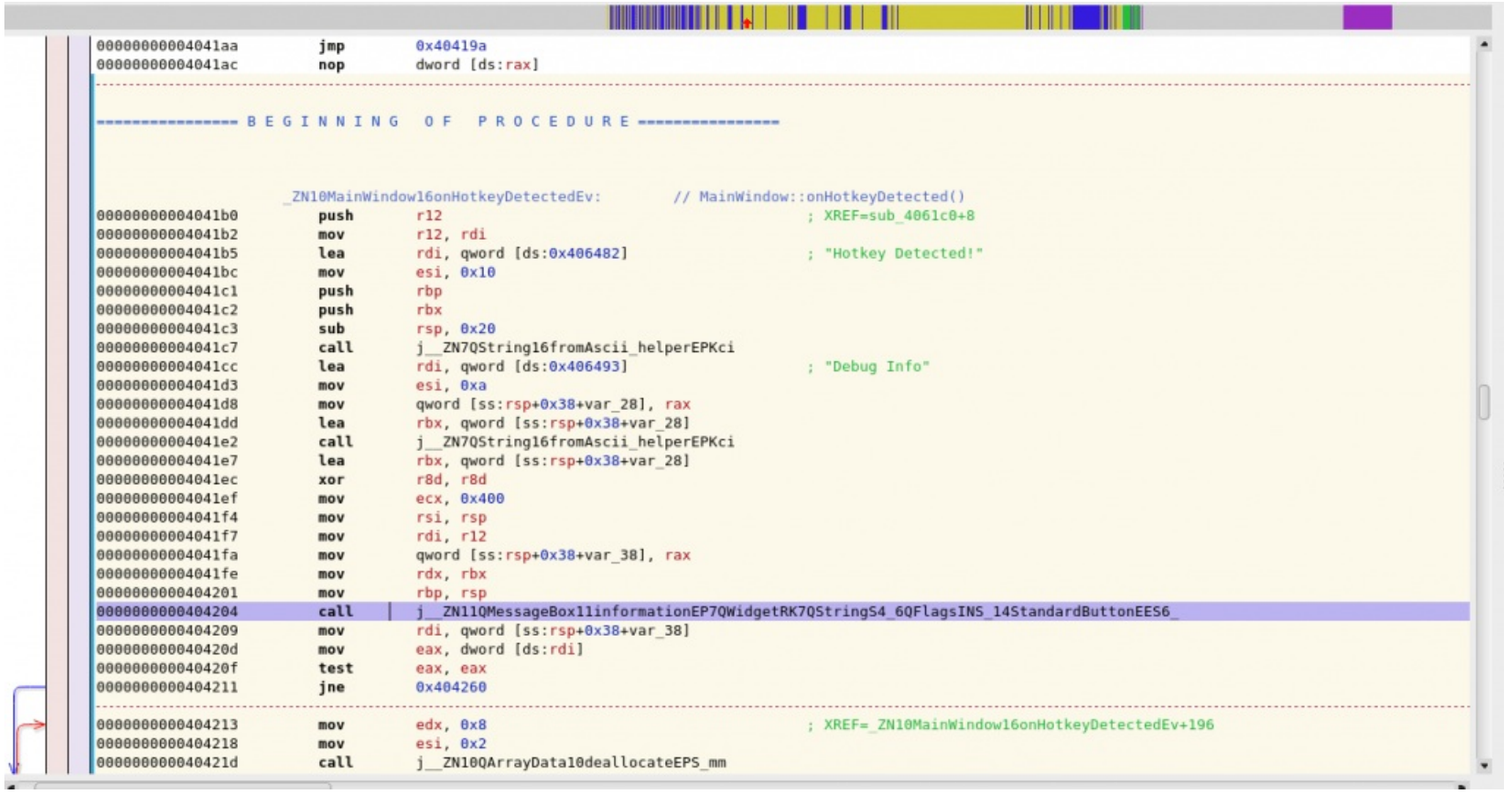
W panelu po lewej wybieram zakładkę *Strings* wyszukuję *„HotkeyDetected”*, wybieram go i główny ekran z kodem przeniósł się do miejsca sekcji z linią alokującą ten string.

(https://i0.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/hopper-1.png)

Jeżeli zwrócimy uwagę na komentarz wygenerowany przez Hopper'a, to zobaczymy nazwy funkcji/referencje do miejsc użycia tego string'a.

```
Assembly (x86)
1 0000000000406482          db          "Hotkey Detected!", 0                    ; XREF=_ZN10MainWindow16onHotkeyDetectedEv+5
```

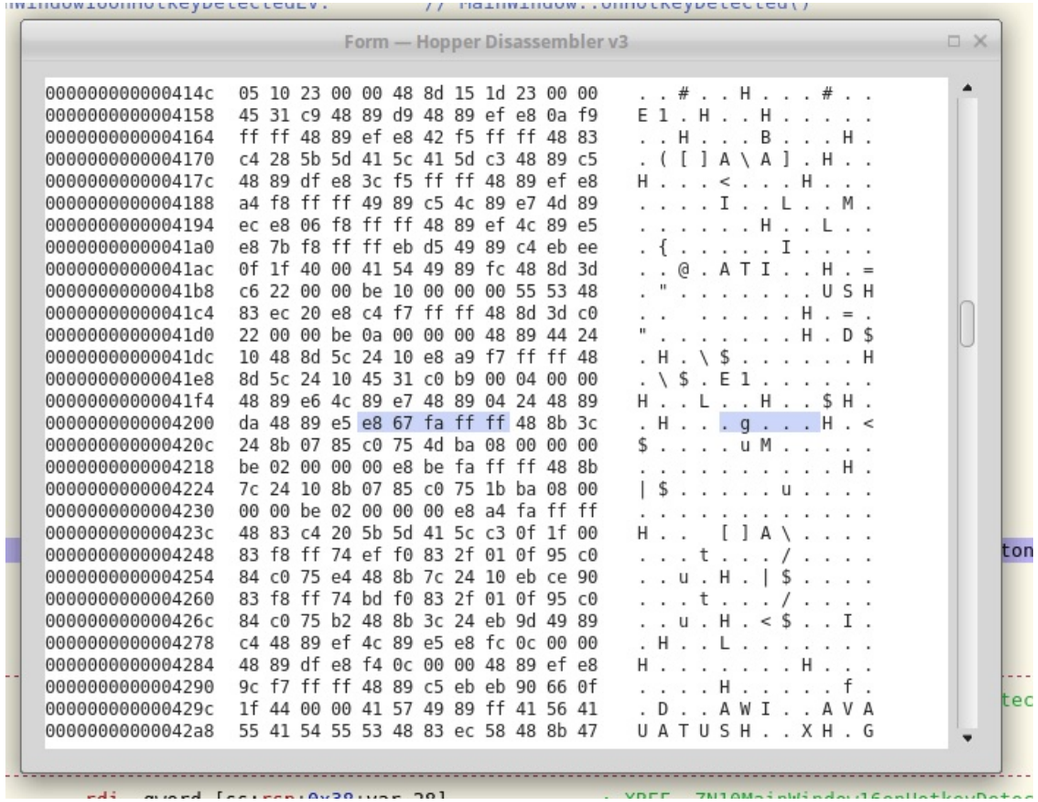
Co ważne: referencja do miejsca jest klikalna, dzięki czemu możemy się przenieść w łatwy sposób do interesującej nas funkcji.



(https://i0.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/hopper-2.png)

Ponownie nie interesuje nas większość tego co się tutaj dzieje, interesują nas głównie wszelkie call'e. Widzimy przygotowanie argumentów i wrzucenie ich na stos. Oprócz tego widzimy naszą znienawidzoną funkcję:

```
Assembly (x86)
1 0000000000404204         call       j__ZN11QMessageBox11informationEP7QWidgetRK7QStringS4_6QFlagsINS_14StandardButtonEES6_
```



(https://i2.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/hopper-3.png)

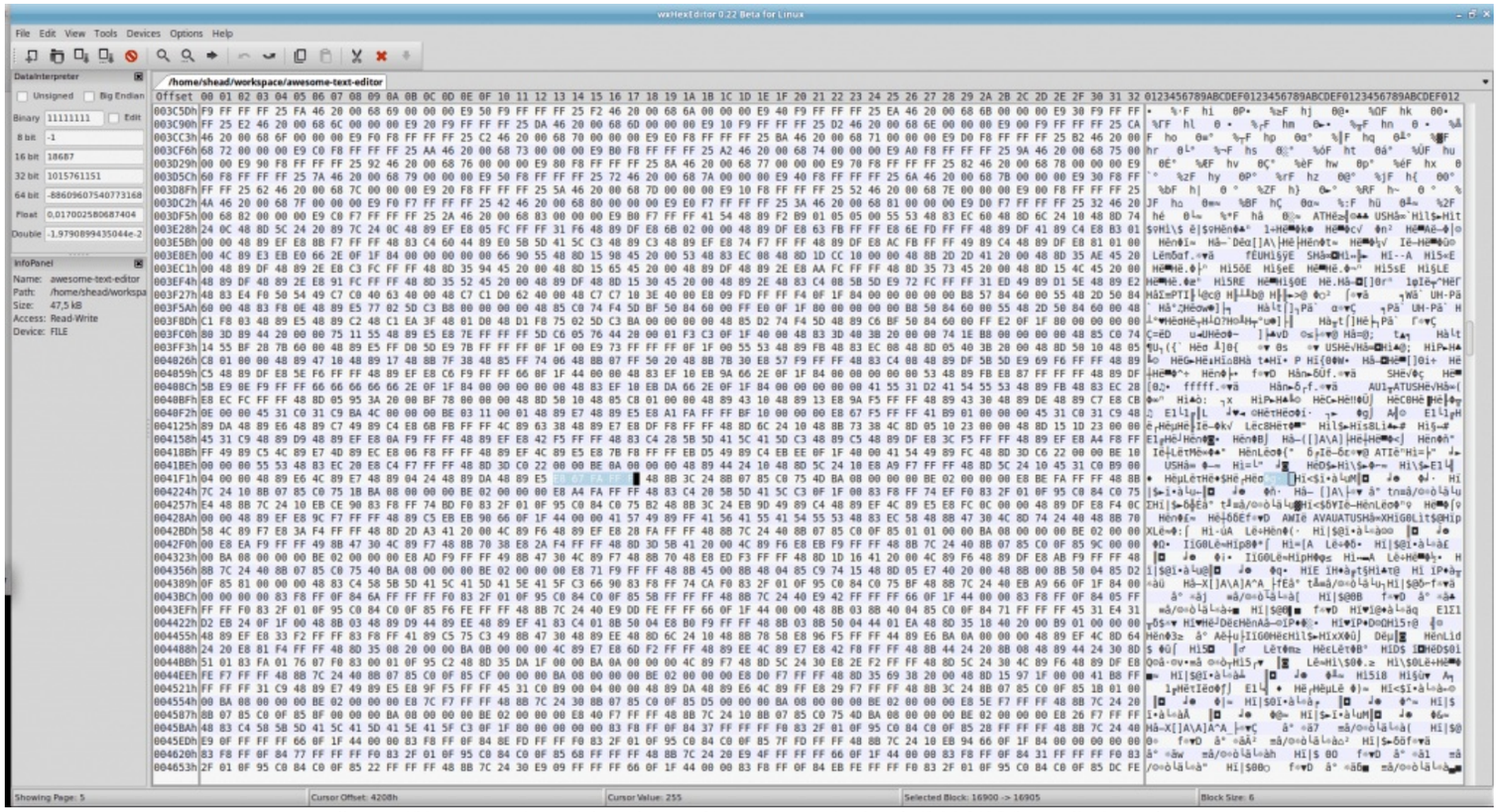
W końcu widzimy miejsce, w którym jest wywołanie tworzące MessageBoxa, znamy jej lokalizację więc możemy się jej pozbyć!

Rozwiązanie problemu – podejście 1

Istnieje kilka potencjalnych rozwiązań dręczącego nas problemu:

1. (do którego dążymy) zaNOPowanie wywołania metody tworzącej komunikat, sposób niezwykle prosty polegający na podmianie instrukcji CALL („wywołaj funkcję i zapisz adres powrotu”), na NOP („nic nie rób, idź do kolejnej instrukcji”). Aby to zrobić musimy zamienić odpowiednie bajty na *0x90*, który jest właśnie instrukcją NOP (*no operation*);
2. można znaleźć miejsce przypisania skrótu klawiszowego i zamienić skrót lub usunąć jego stworzenie, problemem może być to że nie wiemy czy sam skrót nie został stworzony jakoś inaczej i czy nie zostały wykonane wtedy dodatkowe akcje, to by tłumaczyło brak skrótu *[AltGr]+[L]* w panelu ze string’ami.

A więc otwórzmy naszą binarkę w ulubionym hex edytorze i zamieńmy naszego calla na *0x90*.

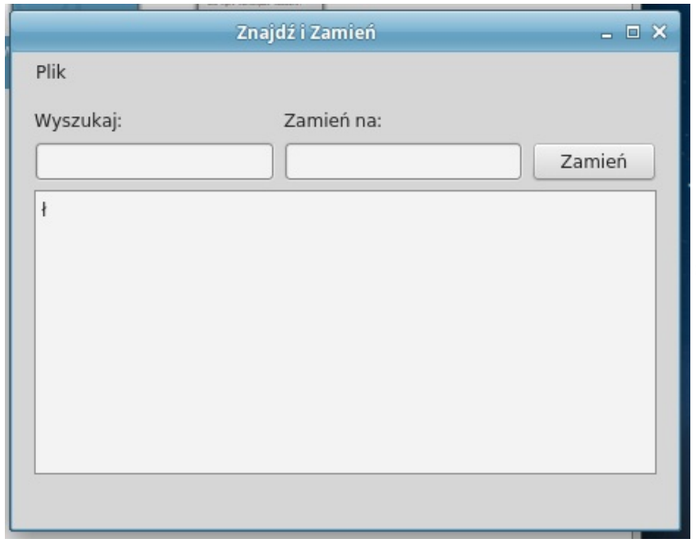


(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/hexedit-1.png)

Zaznaczony ciąg zamieniamy na 90 90 90 90 90

Aby to zrobić, to w przypadku *wxHexEditor* wystarczy: zaznaczyć interesujący nas fragment -> otworzyć menu kontekstowe -> wybrać opcję: *Fill Selection*, a w oknie dialogowym wpisać ’90’ (bez znaków , ,). Następnie zapisujemy plik, wyłączamy hex editor (*wxHexEditor* ma brzydki zwyczaj blokowania binarki) i uruchamiamy program.

Program uruchomił się, więc jest ok. Wywołujemy skrót *[AltGr] + [L]* i… nic się nie dzieje, wywołujemy skrót szybko drugi raz: pojawia się litera „ł”.



(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/scrn-success-and-failure.png)

Wnioski: udało nam się, ale tylko połowicznie. Jak można było przypuszczać program wciąż przechwytuje skrót, jednak z wiadomego powodu nie tworzy już komunikatu (usunęliśmy odpowiedzialny za to fragment). Jedno wywołanie kombinacji idzie na skrót klawiszowy, drugie na wypisanie litery. Aby rozwiązać problem (chyba że zadowalamy się faktem, że do napisania ‚ł' musimy wcisnąć kombinację dwukrotnie) musimy powrócić do analizy i zlikwidować tworzenie się skrótu.

Rozwiązanie problemu – podejście 2

Po chwili researchu i czytania o *Qt* dowiadujemy się że skróty klawiszowe mogą powstać przez stworzenie obiektu *QKeySequence* oraz muszą być połączone przez funkcję *connect*. Stosując intuicję i nasze doświadczenie w programowaniu stwierdzamy, że te czynności mogły zostać zrobione w np. konstruktorze, więc postanawiamy go znaleźć.

(https://i2.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/hopper-4.png)

Udało nam się go zlokalizować i co za radość! Znaleźliśmy fragment tworzący obiekt wcześniej wspomnianej klasy oraz łączący *event* aktywowania go z najprawdopodobniej głównym oknem. Co ciekawe: autor kodu w tym przypadku zamiast skrótu w postaci stringa użył kodu klawiszy, przez co nie byliśmy znaleźć tego skrótu w tabeli z tekstem.

Konstruktor MainWindow - Pseudo C

```
function MainWindow::MainWindow(QWidget*) {
    stack[2046] = r12;
    stack[2045] = rbp;
    QMainWindow::QMainWindow(arg0);
    *(arg0 + 0x10) = vtable for MainWindow + 0x1c8;
    *arg0 = vtable for MainWindow + 0x10;
    rax = operator new(0x78);
    *(arg0 + 0x30) = rax;
    Ui_MainWindow::setupUi(rax);
    QKeySequence::QKeySequence(rsp - 0x8 - 0x8 - 0x8 - 0x8 - 0x28, 0x1001103, 0x4c, 0x0, 0x0);
    rax = operator new(0x10);
    r12 = rax;
    QShortcut::QShortcut(rax, rsp - 0x8 - 0x8 - 0x8 - 0x8 - 0x28, arg0, 0x0, 0x0, 0x1);
    *(arg0 + 0x38) = r12;
    QKeySequence::~QKeySequence(rsp - 0x8 - 0x8 - 0x8 - 0x8 - 0x28);
    QObject::connect(arg_1, *(arg0 + 0x38), "2activated()", arg0, "1onHotkeyDetected()", 0x0);
    rax = QMetaObject::Connection::~Connection(arg_1);
    return rax;
}
```

Widzimy także 2 call’e: jeden z nich to „normalny” *connect*,a drugi to specjalny dodatek od Qt tworzący *MetaObject*, sprawdźmy co się stanie po zaNOPowaniu tego pierwszego (nie liczymy na sukces)... Nic się nie dzieje, program się nie uruchamia. Widocznie obiekty meta nie zgadzają się z faktycznym stanem obiektów, więc przy wstępnej weryfikacji program postanawia się zakończyć.

Zróbmy NOP’a także na drugim callu… Program się uruchamia, ale istnieje dokładnie ten sam problem co wcześniej: do wygenerowania polskiej litery potrzebne jest podwójne wywołanie kombinacji klawiszy. Jednak nie poddajemy się, powracamy do dokumentacji i zastanawiamy co może być tego powodem.

Po krótkiej lekturze dokumentacji dowiadujemy się, że w momencie stworzenia skrótu klawiszowego tworzy się także informacja dla programu aby zwracał szczególną uwagę na wciśnięcia klawisza AltGr. Dobra, a więc powracamy do kodu maszynowego i NOPujemy wszystkie calle (w konstruktorze) zawierające wyrażenie *„KeySequence”*, widzimy 2 takie wywołania.

Zapisujemy plik i go sprawdzamy…

(https://i2.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/03/success_kid_meme-www.memegen.com_.jpg)

Yay! Wszystko działa poprawnie, usunęliśmy pozostały „lag” na polskie znaki.

Podsumowanie

Mimo woli powstał mi taki mini-poradnik, który radzę traktować z przymrużeniem oka, bo to nie uczenie reverse-engineeringu było celem tego artykułu, a jedynie pokazanie jak ciekawe i złożonym jest zajęciem.

Chciałbym też zwrócić uwagę, że zrobienie tego co ja zrobiłem w tym przypadku jest w większości sytuacji zabronione przez prawo. Deasemblacja programu łamie prawa autorskie, nie mówiąc już nawet o modyfikacji programu (tak, ten sposób może posłużyć do omijania zabezpieczeń, np weryfikujących licencję, itp), dozwolone jest to jedynie w raptem kilku przypadkach[4]).

/*jeżeli ktoś by mógł skompilować kod pod Windowsa i podesłać mi binarke np w komentarzach to byłym wdzięczny */

Zapraszam także do zapoznania się z materiałami dodatkowymi oraz systemu komentarzy.

Materiały dodatkowe

1. [Gynvael Coldwind (http://gynvael.coldwind.pl/)]
2. [GnuWin – port programów na Windowsa (http://gnuwin32.sourceforge.net/packages.html)]
3. [Qt – opis (https://www.qt.io/developers/)]

4. [Dekompilacja, a prawo autorskie (http://www.michalstawinski.pl/blogs/item/44-dekompilacja-programu-komputerowego)]

Code ON!