VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA SKRIPTŮ PRO ÚČELY SHLUKOVÉ ANALÝZY

ANALYSIS OF SCRIPTS FOR USE IN CLUSTERING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MICHAL PLANIČKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Planička Michal**
Program: Informační technologie
Název: **Analýza skriptů pro účely shlukové analýzy**
Analysis of Scripts for Use in Clustering
Kategorie: Překladače

Zadání:

1. Studujte skriptovací jazyky (JavaScript, VBScript atd.), především z pohledu jejich strojové analýzy, identifikace a extrakce unikátních vlastností.
2. Seznamte se se službou Clusty, která ve společnosti Avast slouží ke shlukové analýze souborů podle jejich podobnosti.
3. Navrhněte rozšiřitelný analyzátor skriptů, který bude umět rozpoznat, o jaký typ skriptu se jedná, analyzovat jej a poskytnout kolekci vlastností využitelných ke shlukové analýze ve službě Clusty. Analyzátor by měl umět zpracovat soubory napsané v alespoň pěti skriptovacích jazycích. Podporované typy skriptů vyberte po domluvě s vedoucím a konzultantem.
4. Analyzátor navržený v předchozím bodě implementujte.
5. Vytvořené řešení důkladně otestujte sadou jednotkových a integračních testů.
6. Zhodnoťte svou práci a diskutujte možný budoucí vývoj.

Literatura:

- M. Lutz: *Learning Python*, O'Reilly Media, (2013, 5th edition), ISBN 978-1449355739
- T. Powell, F. Schneider: *JavaScript The Complete Reference*, McGraw-Hill Education (2012, 3rd edition), ISBN 978-0071741200
- W. Stanek, J. O'Neill, J. Rosen: *Microsoft PowerShell, VBScript and JScript Bible*, Wiley (2009), ISBN 978-0470386804
- Interní dokumentace společnosti Avast.

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání a rozpracování čtvrtého bodu.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Křivka Zbyněk, Ing., Ph.D.**
Konzultant: Zemek Petr, Ing., Avast
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Tato práce se zabývá analýzou různých typů skriptů a extrakcí vlastností, které mohou být použity ke shlukové analýze prováděné službou Clusty. Vstupem analyzátoru je soubor obsahující zdrojový kód skriptu. Soubor je nejprve dekódován, poté je na základě jeho obsahu určen typ skriptu. Jedná-li se o podporovaný typ skriptu, dojde k analýze zdrojového souboru. Výsledkem analýzy je kolekce vlastností, která je následně převedena do požadovaného formátu (např. JSON). Analyzátor podporuje pět typů skriptů – JavaScript, VBScript, PowerShell, Python a dávkový soubor – přičemž byl navržen tak, aby jej bylo možné rozšířit o nové typy.

Abstract

This paper deals with the analysis of various script types and the extraction of traits which can be used for the cluster analysis performed by Clusty service. A file containing source code of the script serves as an input of the analyzer. The file is decoded and its content is then used for identifying the script type. If the script type is successfully identified, the analysis of the source code will be performed. The result of the analysis is a collection of traits which is then represented in the desired format (e.g. JSON). There are five supported types of scripts: JavaScript, VBScript, PowerShell, Python and batch file. The analyzer is designed such that support for new script types can be added.

Klíčová slova

analýza, dávkový soubor, Javascript, Powershell, Python, shluková analýza, shlukování, skriptovací jazyky, VBScript

Keywords

analýza, batch file, cluster analysis, clustering, Javascript, Powershell, Python, scripting languages, VBScript

Citace

PLANIČKA, Michal. *Analýza skriptů pro účely shlukové analýzy*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Analýza skriptů pro účely shlukové analýzy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Další informace mi poskytl pan Ing. Petr Zemek, Ph.D., jenž byl mým konzultantem ve firmě Avast Software s.r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Planička
13. května 2019

Poděkování

Rád bych poděkoval panu doktoru Křivkovi, pod jehož vedením jsem bakalářskou práci tvořil. Dále bych chtěl poděkovat panu doktoru Zemkovi, který mi během vývoje analyzátoru poskytoval odborné rady.

Obsah

Kapitola 1

Úvod

Antivirovým společnostem, mezi něž se řadí také Avast Software, denně přichází velké množství souborů, u kterých vzniklo podezření, že jejich účelem je odcizit, poškodit či jinak negativně ovlivnit data uživatele. Aby antivirové společnosti mohly před škodlivými soubory vyvinout patřičnou ochranu, je nutné tyto soubory analyzovat. Ruční analýza tolika souborů by ovšem znamenala příliš velkou časovou zátěž pro virové analytiky, proto je příhodné provádět analýzu automaticky pomocí počítačového programu.

Ještě předtím, než program s analýzou začne, je však potřeba soubory roztřídit do skupin tak, že každá skupina bude obsahovat soubory, jež si jsou podobné. Díky tomuto roztřídění bude analýza co nejefektivnější – na každou skupinu mohou být aplikovány specifické metody analýzy.

Ve firmě Avast Software se o třídění souborů stará služba Clusty. Ta dokáže rozpoznat mnoho typů souborů, například spustitelné programy operačních systémů Windows, Linux a Android, nezvládá však rozpoznat skripty.

Skript je soubor psaný ve skriptovacím jazyce. Má různá využití – může vykonávat časově náročné úlohy v počítači, čímž šetří náš čas, může běžet na webových stránkách a usnadňovat nám vyplňování formulářů. Jsou ovšem i skripty, které mají za cíl tajně sledovat naše aktivity, zcizit naše soukromá data nebo jiným způsobem škodit. Existence těchto škodlivých skriptů je dostatečným důvodem, proč přidat jejich rozpoznávání.

Cílem této práce je vyvinout analyzátor skriptů, jež může být použit právě pro rozpoznání skriptů ve službě Clusty, a reagovat tak na hrozby zmíněné v předchozím odstavci. Úkolem aplikace, které se tato práce věnuje, je extrahovat ze skriptů takové vlastnosti, pomocí kterých bude možné dané skripty zařadit do příslušných skupin. Jako první dochází k určování typu skriptu. Aplikace na základě jeho obsahu vyhodnotí, zda se jedná o podporovaný typ skriptu, či nikoliv. Podporovaným typem skriptu může být např. skript určený pro webové stránky a psaný v jazyce JavaScript. Pokud aplikace pracuje se skriptem podporovaného typu, začne s extrakcí vlastností. Vlastnostmi jsou například počet řádků skriptu a seznam řetězců vyskytujících se ve skriptu. Výstupem aplikace je typ a množina vlastností ve formátu, který lze předat na vstup programu pro automatickou analýzu.

Práce je členěna následovně. V kapitole 2 jsou uvedeny důležité informace k podporovaným typům skriptů (skriptovacím jazykům), zejména k jejich lexikální struktuře a syntaxi. Tématem kapitoly 3 jsou vlastnosti, jež mají být extrahovány ze skriptů a na jejichž základě mají být dané skripty roztříděny. Kapitola 4 popisuje, jakým způsobem dochází ke třídění souborů do skupin ve firmě Avast Software a jak se na této činnosti podílí služba Clusty. Z kapitoly 5 se lze dozvědět, jak vypadá obecný návrh aplikace – na jaké části se dělí, co která část vykonává, jaké jsou vstupy a výstupy jednotlivých částí a jak postupovat

v případě přidání nového typu skriptu, který má být podporován. Kapitola 6 se věnuje implementačním detailům aplikace. Jsou v ní uvedeny způsoby, jakým byly řešeny jednotlivé části návrhu z předchozí kapitoly. V kapitole 7 jsou zaznamenány výsledky proběhlých testů a jejich vyhodnocení. Závěr (kapitola 8) obsahuje souhrn všeho řečeného a konečné zhodnocení výsledků práce.

Kapitola 2

Skriptovací jazyky

Tato kapitola pojednává o skriptovacích jazycích, v nichž jsou skripty psány a na jejichž základě jsou určovány typy skriptů. Každému jazyku je věnována jedna podkapitola, která obsahuje charakteristiku daného jazyka, pravidla lexikální analýzy pro lexémy, které jsou významné pro analyzátor, a vybrané syntaktické konstrukce, pomocí kterých jej lze identifikovat.

Celkem pět podkapitol je věnováno jazykům JavaScript, VBScript, PowerShell, Python a dávkovým souborům operačního systému Windows.

Zmíněné jazyky a typy byly vybrány po konzultaci s virovými analytiky z Avast Software. Skripty vybraných typů přicházejí k analýze nejčastěji, a je proto nutné zavést jejich podporu ve službě Clusty (více o této službě v kapitole 4).

2.1 JavaScript

Tato podkapitola se věnuje programovacímu jazyku JavaScript. Informace zde uvedené (charakteristika, lexikální struktura a syntaxe jazyka) pochází z [5], pokud není uvedeno jinak.

2.1.1 Charakteristika

JavaScript je programovací jazyk primárně užívaný k vytváření klientských skriptů pro WWW. Drtivá většina moderních webových stránek používá JavaScript a všechny moderní webové prohlížeče obsahují jeho interpret, což z něj dělá jeden z nejužívanějších programovacích jazyků v historii. Řadí se mezi vysokoúrovňové, dynamické a netypované programovací jazyky a je vhodný k objektově orientovanému i funkcionálnímu programování.

Jeho nevýhodou z hlediska bezpečnosti je fakt, že je jedním z hlavních prostředků k neoprávněným zásahům do webových aplikací, a to právě kvůli jeho převaze a popularitě při vývoji interaktivních webových aplikací. Jak je uvedeno v [6], nástup jazyka JavaScript a jeho pozdějších rozšíření (např. AJAX) umožnil rozvoj metod, jež mají za cíl zcizení citlivých dat, neoprávněný přístup k prohlížeči klienta či narušení integrity systému. Tyto metody jsou označovány zkratkou XSS (angl. *cross-site scripting*).

Ačkoliv je název JavaScript používán jako souhrnné označení pro programovací jazyk používaný na straně klienta webových aplikací, ve skutečnosti existuje několik variant tohoto jazyka. Standardizovaná verze se nazývá ECMAScript; tento standard je udržován

společností ECMA[1]. Na základě tohoto standardu vznikly následující implementace: JavaScript (dříve LiveScript) od firmy Netscape a JScript od firmy Microsoft. Tato informace je důležitá pro identifikaci skriptu, neboť při integraci do HTML může být skript importován i pod těmito názvy (odlišný typ standardu MIME[2] u párového elementu `<script>`, jež skript integruje).

2.1.2 Lexikální struktura

Bílé znaky, které se nacházejí mezi tokeny, jsou interpretem ignorovány. Výjimku tvoří odřádkování u příkazů `return`, `break`, `continue` a operátorů inkrementace a dekrementace. Kromě běžné mezery jsou mezi bílé znaky řazeny také posun o řádek (angl. *line feed*, zkráceně LF), návrat vozíku (angl. *carriage return*, zkráceně CR), tabelátor, vertikální tabelátor, posun o stránku (angl. *form feed*), nedělitelná mezera, označení pořadí bajtů (angl. *byte order mark*) a další znaky z kategorie Zs znakové sady Unicode. Konec řádku je reprezentován buď znakem návratu vozíku, nebo znakem posunu o řádek nebo oběma těmito znaky v tomto pořadí.

Komentáře lze zapisovat dvěma způsoby. Jednořádkové komentáře se zapisují mezi dvě lomítka (`//`) a konec řádku a víceřádkové mezi znaky `/*` a `*/`. Víceřádkové komentáře nelze vnořovat. Oba typy komentářů jsou při interpretaci ignorovány.

Řetězcové literály se z každé strany ohraničují odpovídajícím párem uvozovek (`"`) nebo apostrofů (`'`). Aby znak ohraničující literál mohl být součástí daného literálu, je nutné před něj vložit zpětné lomítko (\). Aby zpětné lomítko neplnilo funkci únikového znaku (angl. *escape character*), musí mu předcházet další zpětné lomítko.

JavaScript patří k jazykům, které rozlišují mezi velkými a malými písmeny. Pro klíčová slova platí, že musí být zapsána malými písmeny. Názvy proměnných, funkcí a další identifikátory musí být zapsány s odpovídající velikostí písma, proto slova `foo`, `FOO` a `Foo` představují tři různé identifikátory.

Zápis identifikátorů musí splňovat další dvě pravidla. Prvním je, že identifikátor smí začínat jen písmenem, podtržítkem nebo znakem dolaru. Druhým je, že zbývající část identifikátoru smí obsahovat pouze znaky z předchozího pravidla a čísla. V identifikátoru lze uvádět libovolná písmena znakové sady Unicode (všechny znaky z kategorií začínajících písmenem L).

2.1.3 Syntaxe jazyka

Syntaxe jazyka JavaScript je odvozena od jazyka Java. Podobnou syntaxi sdílí i další skriptovací jazyky (např. PHP), proto byla pro identifikaci vybrána pouze malá množina konstrukcí, které jsou buď samostatně, nebo v kombinaci s ostatními pro JavaScript unikátní. Mezi tyto konstrukce se řadí: deklarace proměnných, definice funkcí, vytváření objektů pomocí literálů, operátor `typeof`, globální funkce knihovny jQuery[3] a metody a proměnné pro práci s objektovým modelem dokumentu.

Deklarace proměnných probíhá pomocí klíčových slov `var` a `let`. Za nimi následuje proměnná či více proměnných oddělených čárkou. Proměnným může být při deklaraci přiřazena hodnota (mohou být inicializovány):

```
var a;
```

[1] Podrobnosti na https://www.ecma-international.org/publications/standards/Ecma-262.htm
[2] Podrobnosti viz https://cs.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions
[3] Více informací o knihovně je uvedeno na https://jquery.com/

```
let b = "string", c;
```

Funkce lze v jazyce JavaScript definovat více způsoby. Nejčastěji se tak děje pomocí klíčového slova `function`, za nímž následuje název funkce (neuvádí se u anonymních funkcí), seznam parametrů a tělo funkce. Anonymní funkce mohou být také definovány pomocí operátoru `=>`, který očekává seznam parametrů na levé a tělo funkce na pravé straně:

```
function sum(a, b) { return a + b; }
sum2 = function(a, b) { return a + b; }
sum3 = (a, b) => a + b;
```

Jedním ze způsobů, jak lze vytvářet objekty v jazyce JavaScript, jsou objektové literály. Jedná se o dvojice `název: hodnota` oddělené čárkami a uzavřené do složených závorek:

```
obj = {num: 1, str: "string"};
```

Operátor `typeof` je unární a očekává na pravé straně libovolný literál či objekt:

```
tp = typeof 1;
```

Knihovna jQuery je nejčastěji užívané rozšíření jazyka JavaScript. Knihovna využívá jedné globální funkce, jež je běžně zkracována na `$`:

```
$(document)
$.getJSON()
```

K manipulaci s objektovým modelem dokumentu slouží předdefinované metody a proměnné. Jedná se o:

- proměnné `console`, `document` a `window`,
- atributy `innerHTML` a `outerHTML`,
- metody `querySelector` a `querySelectorAll` a
- metody `getElementById`, `getElementsByClassName` a `getElementsByTagName`.

2.2 VBScript

Tato podkapitola se věnuje vlastnostem a lexikálním a syntaktickým pravidlům skriptovacího jazyka VBScript.

2.2.1 Charakteristika

Jak je uvedeno v [16], Visual Basic Script Language (zkráceně VBScript) je jeden ze skriptovacích jazyků firmy Microsoft, jehož hlavním účelem je vývoj klientských, ale i serverových skriptů pro webové aplikace. VBScript je odvozen od programovacího jazyka Visual Basic, totéž platí o jeho lexikální struktuře a syntaxi.

Skripty vyvíjené v jazyce VBScript jsou určeny pro prohlížeč Internet Explorer [17]; ostatní webové prohlížeče VBScript implicitně nepodporují [2]. Skripty jsou stejně jako JavaScript do HTML začleněny pomocí párového elementu `<script>`, a to buď přímým zápisem kódu do elementu, nebo jeho exportováním ze souboru.

I přes to, že VBScript nemá při vývoji klientských skriptů tak dominantní postavení jako JavaScript, lze i jej potencionálně zneužít k útokům pomocí metod XSS [6].

2.2.2 Lexikální struktura

Informace zmíněné v této sekci pocházejí z [17].

Bílé znaky jsou v jazyce VBScript určeny k oddělení jednotlivých tokenů. Při interpretaci jsou ignorovány. Koncem řádku může být buď znak návratu vozíku, nebo znak posunu o řádek nebo kombinace těchto znaků v tomto pořadí. Jednotlivé příkazy jsou od sebe odděleny buď koncem řádku, nebo dvojtečkou.

VBScript umožňuje zapisovat pouze jednořádkové komentáře. Komentář začíná buď znakem apostrofu, nebo klíčovým slovem `Rem`. Za komentář je považován veškerý text od jeho začátku až po znak konce řádku. Během interpretace jsou veškeré komentáře ignorovány.

K uvození řetězcových literálů jsou používány uvozovky (`"`). Pokud má být uvozovka součástí literálu, musí jí předcházet další uvozovka. Platným literálem obsahující uvozovky je tedy například `"řetězec ""obsahuje"" uvozovky"`.

Jak pro identifikátory proměnných a funkcí, tak pro klíčová slova platí, že nezáleží na velikosti písmen; VBScript se tedy řadí mezi tzv. *case-insensitive* jazyky.

Pro tvorbu identifikátorů jsou stanovena dvě pravidla. Prvním je, že identifikátor nesmí překročit délku 254 znaků. Druhým je, že identifikátor smí obsahovat pouze alfanumerické znaky[4] či podtržítko, přičemž počátečním znakem nesmí být číslo.

2.2.3 Syntaxe jazyka

Informace o syntaxi vybraných příkazů a operátorů byly čerpány z [16] a [17].

Poněvadž je VBScript podmnožinou jazyka Visual Basic, odvíjí se od něj i jeho syntaxe. Tato syntaxe má mnoho unikátních rysů, díky kterým lze VBScript poměrně snadno rozpoznat. Mezi tyto rysy patří: vymezení bloků pomocí klíčových slov, deklarace proměnných a redeklarace polí, podmíněné příkazy, cykly, správa výjimek, vytváření objektů a klíčové slovo `Nothing`.

K vymezení bloku příkazů jsou použita klíčová slova namísto páru závorek či odsazení. Jedná se o klíčové slovo `End` ukončující kupříkladu funkce a procedury a klíčová slova `Loop`, `Wend` a `Next`, jež ukončují cykly.

K deklaraci proměnných slouží klíčové slovo `Dim`, zatímco realokace pole se provádí klíčovým slovem `ReDim`. Při větším počtu proměnných či polí je lze oddělit čárkou, proměnné lze navíc inicializovat:

```
Dim a, b = 1
ReDim c(3), d(5)
```

Mezi vybrané podmíněné příkazy patří jednoduchý podmíněný příkaz `If - Then` a přepínač `Select - Case`:

```
If variable < 0 Then
    ' todo
End If

Select Case variable
    Case 1
        ' todo
    Case 2
```

[4] Alfanumerickými znaky jsou myšleny znaky A až Z, a až z a 0 až 9.

```
        ' todo
End Select
```

VBScript obsahuje celkem pět typů cyklů, každý z nich má vlastní jedinečnou syntaxi. Jedná se o cykly `For Each - In`, `For - To`, `Do - Loop While|Until`, `Do While|Until - Loop` a `While - Wend`:

```
For Each variable in array
    ' todo
Next

For variable = 0 To 9
    ' todo
Next

Do
    ' todo
Loop While variable < 0

Do Until variable < 0
    ' todo
Loop

While variable < 0
    ' todo
Wend
```

Ke správě výjimek jsou použita klíčová slova `On Error`, za nimiž následuje buď `GoTo`, nebo `Resume Next`:

```
On Error GoTo 0
On Error Resume Next
```

Vytvořit objekt lze funkcí `CreateObject`. Reference na vytvořený objekt je přiřazena klíčovým slovem `Set`:

```
Set object = CreateObject("type of object")
```

Klíčové slovo `Nothing` se používá v případech, kdy nemá proměnná či parametr funkce uchovávat referenci na objekt:

```
Set object = Nothing
```

2.3 PowerShell

Tato podkapitola je věnována interpretu příkazů a skriptovacímu jazyku PowerShell. Charakteristika jazyka pochází z [1], zatímco podrobnosti k lexikální struktuře a syntaxi byly získány z [8].

2.3.1 Charakteristika

Názvem PowerShell je označován interaktivní interpret příkazů (angl. *shell*) s vlastním skriptovacím jazykem, jehož účelem je automatizace často prováděných úloh systému i uživatelů. Je vyvíjen firmou Microsoft a pracuje na platformě .NET. Mezi jeho hlavní výhody oproti běžnému interpretu příkazů patří nový typ strukturovaných příkazů (tzv. *cmdlets*), integrace objektů a objektového řetězení příkazů a funkčnost napříč různými technologiemi (vyjma .NET například XML či Active Directory). Pro účely této práce bude dále PowerShell označován pouze jako „jazyk".

Jazyk PowerShell může být zneužit k vytváření hrozeb nazývaných bezsouborový malware (angl. *fileless malware*). Definice převzaná z [12] říká, že se jedná o škodlivý kód, který po spuštění skriptu narušuje paměť procesu bez zanechání stop na disku; konkrétními příklady tohoto typu malwaru jsou Powerliks nebo PowerSniff.

2.3.2 Lexikální struktura

Zdrojový kód jazyka PowerShell je tvořen posloupností znaků ze znakové sady Unicode. Z těchto znaků je překladačem jazyka PowerShell vytvářena posloupnost tokenů. Posloupnost tokenů, která je ukončena odřádkováním (návrat vozíku, posun o řádek či obojí), tvoří jeden příkaz.

Účelem bílých znaků ve zdrojovém kódu je separace jednotlivých tokenů; žádný další význam nemají. V jazyce PowerShell jsou za bílé znaky považovány všechny znaky znakové sady Unicode patřící do skupiny Zs, Zl či Zp, dále horizontální a vertikální tabelátor, posuv o stránku a zpětný apostrof (angl. *backtick*) bezprostředně následovaný odřádkováním. Odřádkování, jemuž nepředchází zpětný apostrof, není považován za bílý znak.

PowerShell definuje celkem tři typy komentářů. Prvním typem jsou jednořádkové komentáře začínající mřížkou (`#`) a končící koncem řádku. Druhým typem jsou víceřádkové komentáře začínající znaky `<#` a končící znaky `#>`. Třetím typem jsou jednořádkové komentáře `#requires` specifikující podmínky, jež musí být splněny, aby došlo ke spuštění skriptu. Pro komentáře platí, že je nelze vnořovat a že z hlediska překladače mají stejný význam jako bílé znaky (mohou oddělovat tokeny).

Řetězcové literály mohou být ohraničeny uvozovkami nebo apostrofy, přičemž je-li před levým hraničním a za pravým hraničním znakem uveden znak `@`, literál akceptuje odřádkování, lze tudíž vytvořit víceřádkový literál. Pokud mají být hraniční znaky součástí literálu, musí být zapsány v jednom z následujících tvarů: `""` či `‘"` u literálů ohraničených uvozovkami a `’’` či `‘‘` u literálů ohraničených apostrofy. Zpětný apostrof je v obou typech literálů nutné zapsat jako `‘‘`, pokud nemá sloužit jako únikový znak pro uvozovku/apostrof.

U příkazů, proměnných a klíčových slov není rozlišována velikost písmen. Identifikátory proměnných mohou mít neomezenou délku a začínají znakem dolaru. Zbývající část identifikátoru smí obsahovat čísla, podtržítko, otazník a znaky Unicode, které patří do skupiny začínající písmenem L (např. `$číslo`). Je-li posloupnost znaků za znakem dolaru uzavřena do složených závorek, lze uvádět jakékoliv znaky Unicode kromě pravé složené závorky a zpětného apostrofu (např. `${zcela validní identifikátor!}`).

2.3.3 Syntaxe jazyka

Jazyk PowerShell umožňuje kromě interpretace příkazů také řízení toku programu pomocí řídicích struktur a cyklů, vytváření vlastních funkcí a používání proměnných a objektů. Níže

jsou popsány vybrané konstrukce jazyka, které byly použity k identifikaci skriptů psaných v tomto jazyce.

Klíčovou konstrukcí jazyka PowerShell jsou strukturované příkazy nazývané *cmdlets*. Jsou psány ve tvaru `Verb-Noun` a za nimi mohou následovat parametry či konstanty:

```
Get-Process
Where-Process -Value $obj
$json = "{"json": 1}" | ConvertFrom-Json
```

Speciální tvar zápisu mají některé operátory (např. logické a relační). Mohou být unární či binární a zapisují se ve tvaru `-operátor`:

```
if(1 -eq 2) {}
if(1 -in $array) {}
if(-not $variable1) {}
```

K definici funkcí slouží klíčové slovo `function`, jenž je následováno názvem a tělem funkce (parametry jsou uváděny uvnitř):

```
function Write-Help {
    # todo
}
```

Parametry skriptu, funkce či metody se zapisují do kulatých závorek, kterým předchází klíčové slovo `param`:

```
param([int]param1, [int]param2)
```

K iteraci nad všemi prvky kolekce slouží cyklus `foreach`:

```
foreach($i in $collection) {
   # todo
}
```

Přístup ke statickým atributům či metodám určité třídy je realizován pomocí operátoru `[třída]::`, za nímž následuje identifikátor:

```
[Math]::Pi
```

2.4 Python

Tato podkapitola se věnuje programovacímu jazyku Python.

2.4.1 Charakteristika

Jak je uvedeno v [11], Python je víceúčelový programovací jazyk, který v sobě spojuje procedurální, funkcionální a objektově orientovaná paradigmata. Díky širokému spektru dostupných knihoven může být využit v celé řadě oblastí. Mezi nejčastější úlohy vykonávané skripty psanými v jazyce Python patří automatizace systémových úloh, počítačová komunikace, rozhraní pro webové aplikace na straně serveru a numerické a vědecké výpočty.

Víceúčelovost, rozsáhlá dostupnost knihoven třetích stran a snadné pochopení syntaxe dělá z Pythonu vhodný jazyk pro vývoj různých analyzátorů a nástrojů za účelem odhalení bezpečnostních trhlin (např. analyzátor TCP a UDP portů, cracker hesel apod.). Zmíněné nástroje mohou být využity za účelem poškození či kompromitování cíleného systému a uživatelů, ale také k cílenému testování bezpečnosti pomocí penetračních testů. O těchto a dalších informacích je pojednáváno v [14].

2.4.2 Lexikální struktura

Informace o lexikální struktuře byly zjištěny z [3].

Bílé znaky nacházející se mimo řetězcové literály mohou nabývat více významů. Znaky CR, LF a sekvence znaků CR LF slouží k ukončení řádků. Mezery a tabelátory před začátkem příkazu určují úroveň odsazení. Všechny zmíněné bílé znaky mohou také sloužit k oddělení tokenů, pokud již neodsazují příkaz. K oddělení dvou tokenů jsou bílé znaky nezbytné pouze tehdy, když by spojení těchto tokenů bylo interpretováno jako odlišný token.

Jednořádkový komentář začíná mřížkou (`#`) a je zakončen koncem řádku. Python nedefinuje zápis víceřádkových komentářů. Při interpretaci jsou komentáře ignorovány.

Řetězcové literály se dělí na dva typy: jednořádkové a víceřádkové. Jednořádkové jsou ohraničeny buď uvozovkami, nebo apostrofy. Víceřádkové mohou být ohraničeny buďto trojicemi uvozovek, nebo trojicemi apostrofů (např. `"""řetězec"""` či `'''řetězec'''`). Pouze druhý typ smí být rozdělen na více řádků. Pokud se má v literálu nacházet znak, který jej zároveň ohraničuje, je nutné před ním uvést zpětné lomítko. Zpětné lomítko neplní funkci únikového znaku pouze tehdy, když mu předchází další zpětné lomítko.

U klíčových slov a identifikátorů je nutné rozlišovat velká a malá písmena. Všechna klíčová slova musí být psána malými písmeny; výjimkou jsou slova `True`, `False` a `None`, ty musí začínat velkým písmenem.

Identifikátory proměnných, funkcí a dalších konstrukcí je nutné zapisovat s odpovídající velikostí písmen. Délka identifikátoru není omezena. Prvním znakem identifikátoru může být podtržítko nebo znak patřící do jedné z následujících kategorií znakové sady Unicode: Lu, Ll, Lt, Lm, Lo nebo Nl. Ve zbývající části identifikátoru se smí nacházet předchozí znaky a navíc ještě znaky z kategorií Mn, Mc, Nd a Pc.

2.4.3 Syntaxe jazyka

Syntaxe jazyka Python se v několika aspektech liší od ostatních programovacích jazyků. Blok příkazů je určen úrovní odsazení, výrazy u složených příkazů jsou zakončeny dvojtečkou a nemusí být uzavřeny do kulatých závorek. Některá klíčová slova a některé operátory jsou unikátní.

Mezi syntaktické konstrukce, jimiž lze Python dobře rozpoznat, patří:

- definice funkce a metody,
- cyklus `for`,
- vyvolání výjimky `raise`,
- blok `try` a `except`,
- importování funkce z modulu pomocí `from` - `import`,
- lambda funkce,

- bloky `if`, `elif` a `else` a
- klíčové slovo `None`.

Funkce a metody jsou definovány stejným klíčovým slovem – `def`. Za ním následuje identifikátor a seznam parametrů v kulatých závorkách. Prvním parametrem metody bývá `self`, jenž odkazuje na identitu objektu:

```
def func():
    pass

def method(self):
    pass
```

Cyklus `for` má za úkol projít všechny prvky kolekce, jež mu byla předána:

```
for item in collection:
    pass
```

Výjimku lze vyvolat klíčovým slovem `raise`. Zachytit ji lze v bloku `try` a zpracovat v bloku či blocích `except`:

```
raise ValueError("Invalid value")

try:
    pass
except ValueError:
    pass
except (OSError, FileNotFoundError) as e:
    pass
```

K importování modulu slouží klíčové slovo `import` a volitelně také `from`, mají-li být importovány pouze konkrétní funkce, třídy či proměnné:

```
import re
from re import search, IGNORECASE
```

Python umožňuje vytvářet anonymní lambda funkce, a to pomocí klíčového slova `lambda`, za nímž následují parametry, dvojtečka a tělo:

```
sum = lambda a, b: a + b
dd = defaultdict(lambda: "default value")
```

Podmíněné větvení lze vytvořit klíčovými slovy `if`, `elif` a `else`, z nichž první dvě jsou následovány podmínkou:

```
if a < b:
    pass
elif a == b:
    pass
else:
    pass
```

Poslední popisovanou konstrukcí je klíčové slovo `None`. Hodnota `None` je přiřazována proměnným či parametrům funkcí, jež momentálně nemají nést žádnou hodnotu:

```
a = None
```

2.5 Dávkový soubor

Následující sekce pojednávají o dávkových souborech používaných v operačním systému Windows. Informace z nich byly nastudovány či převzaty z [9], pokud není uvedeno jinak.

2.5.1 Charakteristika

Dávkový soubor (angl. *batch file* či *batch script*) je textový soubor s příponou `.bat`, jenž je tvořen sadou příkazů operačního systému Windows. Při spuštění souboru jsou příkazy sekvenčně vykonány interpretem příkazové řádky, přičemž interpretace probíhá shora dolů. Hlavním účelem dávkových souborů je automatizace vykonávání kolekce příkazů, což vede k usnadnění práce a ušetření času uživatelů i administrátorů operačního systému.

Dávkový soubor může být zneužit k vytvoření malwaru, jenž má za cíl provést méně či více závažné změny v operačním systému. Může se jednat o odstranění záložních souborů, skryté spuštění jiných závadných programů či samovolné vypínání počítače. Konkrétním příkladem, jak je možné takové akce provést, je *command injection*, což je dle [13] zranitelnost softwaru, která je na seznamu 25 nejnebezpečnějších zranitelností umístěna na druhém místě.

2.5.2 Lexikální struktura

K oddělení jednotlivých komponentů příkazu slouží bílé znaky. Příkazy jsou od sebe odděleny koncem řádku (CR, LF či CR LF), lze ovšem využít i operátory `&`, `&&`, `|`, `||` a `()`, které mohou zřetězit vykonávání několika příkazů.

V dávkových souborech lze zapisovat jednořádkové komentáře, které začínají příkazem `REM` a končí znakem konce řádku. Alternativně lze místo příkazu `REM` využít dvou po sobě jdoucích znaků dvojtečky (`::`).

Návěští vždy začínají dvojtečkou a končí znakem konce řádku. Před návěštím se smí nacházet pouze bílé znaky, za ním pouze bílé znaky a komentáře.

Při interpretaci nedochází k rozlišování velikosti písmen u příkazů, proměnných ani u návěští. Pro identifikátor proměnné a pojmenování návěští platí, že smí obsahovat libovolné znaky kromě znaků `%`, `=` a `:`. Dále platí, že proměnné nesmí začínat bílými znaky a návěští nesmí začínat ani končit bílými znaky.

2.5.3 Syntaxe jazyka

Interpret dokáže kromě vykonávání příkazů dávkového souboru také podmíněné i nepodmíněné skoky na návěští, vytváření proměnných a cyklů. Dále jsou uvedeny syntaktické konstrukce, které jsou dostatečně unikátní na to, aby se podílely na identifikaci dávkových souborů. K těmto konstrukcím patří příkazy `ECHO`, `SET` a `EXIT`, podmíněný a nepodmíněný skok na návěští, příkaz cyklu `FOR`, relační operátory a kontrola chybového stavu a existence souboru.

Příkazem `ECHO` lze vypsat zprávu na standardní výstup či vypnout vypisování provedených příkazů:

```
ECHO OFF
ECHO.
ECHO text
```

Pomocí příkazu `SET` jsou vytvářeny proměnné souboru. Za příkazem následuje název proměnné, rovnítko a hodnota. Pro uložení numerických hodnot musí být za příkaz doplněn parametr `/A`:

```
SET "var1="
SET var2=text message
SET /A num_var=1
```

Příkaz `EXIT` ukončuje činnost souboru. Za ním lze uvést parametry a číslo chybového stavu, kterým má soubor skončit:

```
EXIT 1
EXIT /B 2
```

Podmíněný skok na návěští lze provést dvojicí příkazů `IF - GOTO`. Nepodmíněné skoky zajišťují příkazy `GOTO` a `CALL`:

```
IF %var1%=="" GOTO :label_1
GOTO :label_1
CALL :label_1
```

Procházet kolekci hodnot je možné příkazem `FOR`. Za příkazem následuje další příkaz, který provede příslušnou akci nad aktuálním prvkem kolekce:

```
FOR %%a IN (item1, item2) DO
                              ECHO %%a
```

K porovnání hodnot proměnných, případně proměnných a konstant v slouží v dávkových souborech relační operátory `EQU`, `NEQ`, `LSS`, `LEQ`, `GTR` a `GEQ`:

```
IF %ERRORLEVEL% EQU 0 ECHO Without error
IF %a% NEQ %b% ECHO No match
```

Kontrolu chybového stavu či existence souboru lze provést pomocí příkazu `IF` a konstanty `ERRORLEVEL`, respektive `EXIST`. Oběma může předcházet konstanta `NOT`:

```
IF ERRORLEVEL 3
IF NOT EXIST test.txt
```

Kapitola 3

Extrahované vlastnosti

V této kapitole jsou podrobněji rozepsány vlastnosti, jež jsou extrahovány ze skriptů a jež budou využity při shlukové analýze. Vlastnosti mohou být buď společné pro všechny typy skriptů, nebo specifické pro jeden či několik typů skriptů. Společné vlastnosti jsou uvedené v podkapitole 3.1, specifické v podkapitole 3.2.

3.1 Společné vlastnosti

Společné vlastnosti se vyskytují ve všech skriptech nehledě na jejich typ. Způsob jejich extrakce se ovšem může pro každý typ lišit. Příkladem mohou být komentáře. Každý podporovaný typ skriptu podporuje zápis komentářů, ale způsob jejich zápisu je u každého typu rozdílný, proto nelze zvolit jednotnou metodu pro jejich získání.

Za společné vlastnosti jsou považovány:

- důvěryhodnost toho, že se jedná o určený typ skriptu,
- kódování souboru,
- chyba vzniklá při dekódování souboru,
- řetězec prvních x znaků skriptu,
- řetězec posledních x znaků skriptu,
- počet řádků skriptu,
- nejdelší řádek skriptu,
- zpráva o chybě vzniklé při identifikaci,
- řádkové komentáře a jejich počet a
- blokové komentáře a jejich počet.

Důvěryhodnost (`confidence`) je vypočítávána při identifikačním procesu (viz kapitola 5). Jedná se o číslo v rozsahu 0 až 100. Důvěryhodnost je ve výsledku analyzátoru zahrnuta vždy a u všech identifikovaných typů skriptů. Tato vlastnost není jako jediná určena k využití při shlukování, nýbrž pro možné srovnání s důvěryhodnostmi dalších identifikovaných typů.

Kódování (`encoding`) je získáváno z modulu pro načtení vstupních dat. Hodnota této vlastnosti je pro všechny typy skriptů vždy společná, totéž platí o metodě použité při její extrakci. Pokud by se vstupní soubor nepodařilo dekódovat, výsledek analyzátoru by obsahoval jedinou vlastnost nazvanou `encoding_error`, jejímž obsahem by bylo sdělení o chybě vzniklé během dekódování.

Řetězce prvních (`start_chars`) a posledních (`end_chars`) x znaků skriptu jsou extrahovány vždy a ze všech skriptů nehledě na jejich typ. Způsob extrakce se u žádného typu neliší. Pro skripty, jejichž délka je větší nebo rovna čtyřnásobku požadované délky řetězců, mají řetězce zvolenou délku. U kratších skriptů odpovídá délka extrahovaného řetězce čtvrtině z celkové délky skriptu. Pokud má být délka řetězců 30 znaků a skript má délku 200, budou mít výsledné řetězce skutečně 30 znaků. Pokud by ale skript byl kratší – měl by kupř. jen 80 znaků – tvořilo by každý řetězec pouze 20 znaků.

Počet řádků skriptu (`line_count`) je určen pro každý identifikovaný typ zvlášť. Během extrakce této vlastnosti je vypočtena i délka každého řádku a počet znaků nejdelšího řádku je uložen jako samostatná vlastnost s názvem `longest_line`.

Zpráva o chybě vzniklé při identifikaci (`external_error`) se nemusí projevit, a proto se ve výsledné kolekci vlastností nemusí nacházet. Přítomnost chyby naznačuje, že se ve skriptu může nalézat syntaktická chyba. Ačkoliv analyzátor umožňuje tuto chybu vygenerovat u všech typů skriptů, ve stávající implementaci se objevuje pouze u typů JavaScript a Python.

Možnost zápisu poznámek ve formě komentářů umožňují všechny typy skriptů, a proto jsou komentáře vždy ve výsledné kolekci vlastností. Komentáře se obecně dělí na řádkové (`line_comments`), které zpravidla nelze rozdělit na více řádků, a blokové (`block_comments`), které v sobě mohou, ale nemusí zahrnovat více řádků. U skriptů typu dávkový soubor, Python a VBScript jsou podporovány pouze řádkové komentáře, skripty typu JavaScript a PowerShell podporují jak řádkový, tak blokový zápis komentářů. Z komentářů jsou odvozeny ještě další dvě vlastnosti: počet řádkových (`line_comments_count`) a počet blokových (`block_comments_count`) komentářů.

3.2 Specifické vlastnosti

Specifické vlastnosti jsou takové vlastnosti, k jejichž získávání dochází u jednoho typu skriptu nebo několika typů, nikdy však u všech. Specifické vlastnosti vyžadují rozdílný způsob extrakce pro každý typ.

Mezi specifické vlastnosti se řadí:

- řetězcové literály a jejich počet,
- klíčová slova a počet jejich výskytů,
- příkazy systému Windows a počet jejich výskytů,
- textové zprávy příkazu `ECHO` a jejich počet,
- návěští pro skok a jejich počet,
- použité příkazy cmdlet a počet jejich výskytů,
- informace o tom, zda je skript součástí kódu HTML a
- informace o tom, zda se v kódu HTML nachází neidentifikovatelný skript.

Řetězcové literály (`strings`) jsou extrahovány u skriptů typu JavaScript, VBScript, PowerShell a Python. Způsob jejich extrakce se u zmíněných typů různí na základě toho, jak jsou literály uvozeny. Vlastnost obsahuje seznam všech získaných literálů. Jejich celkový počet je uložen jako vlastnost (`strings_count`).

Klíčová slova (`keywords`) se nacházejí ve skriptech typu JavaScript, VBScript, PowerShell a Python. Každý z těchto typů má vyhrazena vlastní klíčová slova. Pokud typ definuje i jiná rezervovaná slova, jež mohou nabývat speciálního významu až v budoucích verzích, jsou do seznamu extrahovaných slov zahrnuty též. Vlastnost uchovává ke každému klíčovému slovu počet jeho výskytů ve skriptu. Pokud se klíčové slovo ve skriptu nenalézá ani jednou, není ve vlastnosti uvedeno.

Příkazy systému Windows (`commands`) jsou první specifickou vlastností dávkových souborů. Seznam extrahovatelných příkazů zahrnuje příkazy z aktuální verze tohoto operačního systému, ze starších verzí a také z operačních systémů DOS[1] a OS/2[2]. Ve výsledku analýzy jsou zahrnuty příkazy, jež se v analyzovaném skriptu nachází, a počet jejich výskytů; nepoužité příkazy se neuvádí.

Textové zprávy příkazu `ECHO` (`echo_texts`) jsou druhou specifickou vlastností dávkových souborů. Součástí zprávy mohou být kromě běžného textu také proměnné nebo znaky se speciálním významem (např. tečka či dvojtečka bezprostředně za příkazem pro výpis prázdného řádku). Počet extrahovaných zpráv je uložen jako vlastnost s názvem `echo_texts_count`.

Návěští pro skok (`labels`) jsou třetí a poslední specifickou vlastností dávkových souborů. Kromě explicitně uvedených návěští je extrahováno i implicitní návěští `EOF`, ale pouze v případě, je-li na něj proveden skok příkazy `GOTO` nebo `CALL`. Počet návěští je uložen jako vlastnost nazvaná `labels_count`.

Obdobně jako se v dávkových souborech používají systémové příkazy, pro skripty typu PowerShell jsou charakteristické příkazy cmdlet (`cmdlets`). Počet podporovaných příkazů tohoto typu se různí na základě verze interpretu jazyka PowerShell. Analyzátor umí rozpoznat příkazy ze všech doposud vydaných verzí interpretu. I v této vlastnosti jsou uchovány pouze využité příkazy a počty jejich použití.

Skripty typu JavaScript a VBScript jsou jedinečné v tom, že mohou existovat jako samostatný skript, nebo být integrovány do kódu psaného v jazyce HTML. Údaj o tom, zda skript je nebo není součástí HTML, je ve výsledku analýzy zahrnut jako samostatná vlastnost pojmenovaná `in_html`) a nabývající jedné z hodnot pravda, nepravda.

Pro výše zmíněné skripty včleněné do kódu HTML platí, že jich obecně může být více a že se nemusí jednat pouze o skripty typu JavaScript a VBScript. Analyzátor se snaží rozpoznat všechny včleněné skripty, pokud ovšem nalezne takový, který neodpovídá ani jednomu z podporovaných typů, uloží tuto informaci jako vlastnost `unidentified_script`.

[1] Disk Operating System, podrobnosti na https://cs.wikipedia.org/wiki/DOS
[2] Více na https://cs.wikipedia.org/wiki/DOS

Kapitola 4

Shluková analýza a služba Clusty

Tématem této kapitoly je služba Clusty, která je ve společnosti Avast Software využívána ke shlukování souborů. Kromě informací o této službě jsou stručně uvedena i fakta o shlukové analýze. Informace o službě Clusty (podkapitola 4.2) a konkrétním využití shlukové analýzy (poslední odstavec podkapitoly 4.1) pocházejí z interní dokumentace společnosti Avast Software.

4.1 Shluková analýza

Informace z tohoto odstavce byly získány z [4]. Termínem shluková analýza jsou označovány metody, jejichž cílem je klasifikovat objekty na základě jejich vlastností a zařadit tyto objekty do skupin (shluků) dle jejich podobnosti. Shluk je tedy skupina objektů s podobnými vlastnostmi a klasifikace zařazení objektu do určitého shluku. Ve většině případů[1], kdy je užita shluková analýza, je cílem rozdělení jednotlivých objektů do shluků tak, aby každý objekt byl zařazen právě do jednoho shluku. Neexistuje tedy objekt, jenž by nebyl součástí některého shluku.

Ve firmě Avast je shluková analýza aplikována při klasifikaci příchozích souborů. Shlukování umožňuje hromadné zpracování všech souborů nacházejících se v daném shluku a dovoluje provést odlišné operace pro každý shluk.

4.2 Služba Clusty

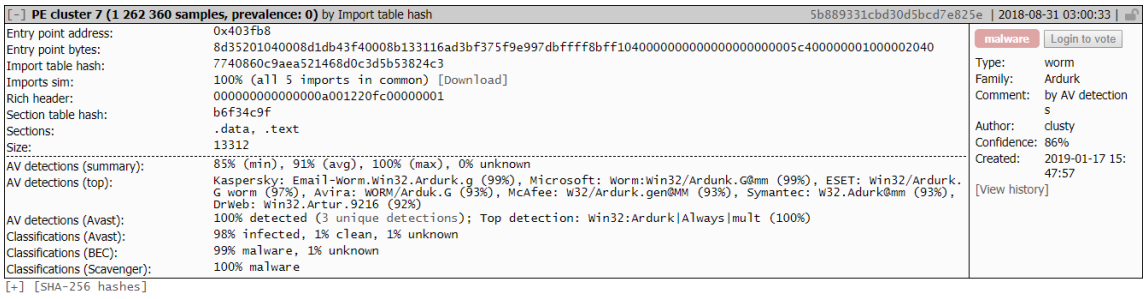
Denně přijde do společnosti Avast 300 000 až 1 000 000 souborů (dále jen „vzorků“) z různých zdrojů – od uživatelů antivirového programu, ale i služeb jako VirusTotal. Mezi vzorky se nemusí nacházet pouze škodlivý software neboli malware (např. CryptoLocker), může se jednat i o neškodný nástroj (např. instalátor prohlížeče Google Chrome). Každý z těchto vzorků by ovšem měl být klasifikován a zařazen do příslušného shluku tak, aby bylo při další manipulaci se vzorkem jasné, jakou konkrétní hrozbu představuje. Pro lepší představu – zmíněný vzorek CryptoLocker by měl být klasifikován jako *malware* (závažnost), *ransomware* (specifický typ malwaru) a *CryptoLocker* (název rodiny malwaru).

Pouhé manuální rozřazování vzorků by vyžadovalo práci velkého množství analytiků a bylo by časově náročné a neefektivní. O rozřazení těchto vzorků do shluků se proto stará služba Clusty. Clusty zpracovává přijatý vzorek tak, že jej analyzuje, extrahuje z něj vlastnosti a na základě nich jej zařadí do vhodného shluku. Podporuje různé typy souborů (např.

[1] Neplatí vždy, služba Clusty kupříkladu nevyžaduje, aby byl soubor zařazen do nějakého shluku.

PE[2], ELF[3] a APK[4] soubory, archivy či dokumenty) a ke klasifikaci využívá široké spektrum vlastností (např. statické vlastnosti, dynamické chování, YARA pravidla[5] či antivirové detekce).

Každý shluk je klasifikován automaticky, ale služba Clusty podporuje i možnost manuální klasifikace. Za automatickou klasifikaci je zodpovědný klasifikátor. Analytici ovšem mohou provést i manuální analýzu shluku a poté hlasovat o tom, jak by měl být klasifikován. Hlasování je považováno za manuální klasifikaci a má přednost před automatickou. Příklad vzniklého shluku je možné vidět na obrázku 4.1.



Obrázek 4.1: Ukázka jednoho ze shluků služby Clusty

Každý shluk se skládá z několika částí: záhlaví (horní šedý řádek), sdílené vlastnosti (pod záhlavím), statistiky detekcí (pod sdílenými vlastnostmi), informace o klasifikaci (pravý boční panel) a manipulace se vzorky (dolní dvě tlačítka pod rámečkem shluku).

Levá část záhlaví obsahuje základní fakta o daném shluku: kategorie (typ souboru), počet vzorků ve shluku, součet počtů zobrazení jednotlivých vzorků uživateli Avastu (*prevalence*) a vlastnost, pomocí které byly vzorky do shluku zařazeny. Pravá část záhlaví je tvořena unikátním identifikátorem shluku, datem jeho vytvoření a ikonkou zámku, pomocí níž lze shluk označit jako právě zpracovávaný (jeho obsah by tudíž neměl být měněn).

Pod záhlavím se nachází sekce, ve které jsou uvedeny vlastnosti společné pro všechny vzorky. Ačkoliv jsou vzorky do shluku řazeny pouze na základě jedné vlastnosti, mohou jich obecně sdílet více. Každá společná vlastnost je uvedena na zvláštním řádku (dvojice *Název vlastnosti: hodnota*). Primární vlastnost, jež slouží k zařazování vzorků do shluku, je vybrána na základě priority; vlastnost s nejvyšší prioritou je vybrána jako primární.

Za společnými vlastnostmi následují tzv. statistiky detekcí. V této sekci je uveden souhrn výsledků antivirových detekcí nejpoužívanějších antivirů (označené *top*), přičemž detekce od antiviru Avast (označené *Avast*) jsou uvedeny zvlášť. Výsledky jsou tvořeny seznamem hrozeb, za něž jsou vzorky považovány, a procentuálním zastoupením takových vzorků ve shluku. Poslední tři řádky pak zobrazují celkové procentuální zastoupení škodlivých, neškodných, neznámých, případně i podezřelých vzorků ve shluku.

Pravý boční panel obsahuje informace o klasifikaci shluku, mezi které patří závažnost (na obrázku *malware* v červeném poli), typ (dle závažnosti), rodina, komentář k tomu, jakým nástrojem byl shluk klasifikován, autor shluku, počet procent, na kolik se dá zvolené

[2]Portable Executable, viz https://en.wikipedia.org/wiki/Portable_Executable

[3]Executable and Linkable Format, viz https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

[4]Android Package, viz https://en.wikipedia.org/wiki/Android_application_package

[5]Podrobnosti k YARA pravidlům zde: https://yara.readthedocs.io/en/v3.8.1/

klasifikaci věřit (*confidence*), a datum vytvoření shluku. Panel je zakončen tlačítkem na zobrazení historie klasifikací daného shluku.

Pod rámečkem shluku jsou uvedena dvě tlačítka; pomocí tlačítka se znakem plus lze zobrazit 50 náhodných vzorků ze shluku, pomocí tlačítka *SHA-256 hashes* lze získat seznam hašů jednotlivých vzorků. U vzorků jsou uvedeny údaje o velikosti, počtu, zobrazení, počtu detekcí antiviry a typech hrozeb, za něž jsou považovány. Seznam vzorků lze vidět na obrázku 4.2.

Links	SHA-256 hash	Size	Prevalence	AV	Avast	ESET	Microsoft	Kaspersky
	000020050...	13 KB	0	6/6	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	00004dac5...	13 KB	0	9/10	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	00004e98b...	13 KB	0	10/11	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000059f6a...	13 KB	0	10/11	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000071d3d...	13 KB	0	10/11	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000074adf...	13 KB	0	10/11	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000078640...	13 KB	0	10/11	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000078f5f...	13 KB	0	10/11	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000089db2...	13 KB	0	9/10	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g
	000093999...	13 KB	0	6/6	Win32:Ardurk|Always|mult (+1)	Win32/Ardurk.G worm	Worm:Win32/Ardunk.G@mm	Email-Worm.Win32.Ardurk.g

Obrázek 4.2: Seznam vzorků shluku

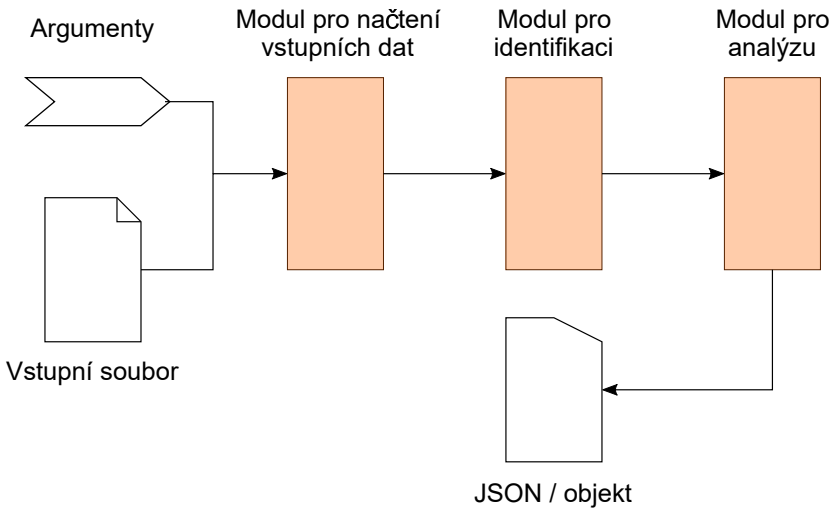
Kapitola 5

Návrh

Tato kapitola se věnuje obecnému návrhu analyzátoru. V samostatných podkapitolách jsou uvedeny podrobnosti k jeho účelu a členění (podkapitola 5.1), k funkcím jednotlivých částí a jejich vzájemnému propojení (podkapitoly 5.2, 5.3 a 5.4) a k rozšiřitelnosti analyzátoru (podkapitola 5.5). Implementační detaily jsou uvedeny v kapitole 6.

5.1 Účel a členění

Účelem analyzátoru je rozpoznat, v jakém skriptovacím či programovacím jazyce je skript psán (jinými slovy určit typ skriptu), a následně provést analýzu tohoto skriptu na základě rozpoznaného typu. Vzhledem k tomu, že analyzátor vykonává více úloh a že je kladen důraz na jeho snadnou rozšiřitelnost, je rozdělen na několik částí (modulů), které se o tyto úlohy starají. Členění analyzátoru je znázorněno ve formě diagramu na obrázku 5.1.



Obrázek 5.1: Diagram znázorňující členění analyzátoru včetně vstupů a výstupů

Vstupem analyzátoru je soubor se zdrojovým kódem. Modul pro načtení vstupních dat má za úkol načíst a dekódovat obsah souboru, extrahovat z něj skripty v případě, že jsou součástí kódu psaného v jazyce HTML, a zpracovat argumenty zadané při spuštění. Načtený zdrojový kód je následně předán modulu pro identifikaci, jenž rozhodne, zda se jedná

o kód podporovaného typu skriptu, či nikoliv. Výsledky identifikace jsou spolu s kódem předány modulu pro analýzu. Pokud nebyl rozpoznán žádný typ, analýza kódu neprobíhá. V opačném případě provede modul analýzu kódu a její výsledky si uloží. Výstupem modulu pro analýzu a také analyzátoru je kolekce vlastností extrahovaných z poskytnutého kódu a převedených do požadovaného formátu.

5.2 Modul pro načtení vstupních dat

Při čtení vstupního souboru modul předpokládá, že soubor mohl být zakódován pomocí libovolného standardu pro kódování znaků (např. UTF-8, Windows-1252 pro západoevropské jazyky nebo EUC-JP pro japonštinu). Proto je obsah souboru načten jako sekvence bajtů, u které může být provedeno rozpoznání kódování. Modul se nejprve pokusí dekódovat sekvenci pomocí UTF-8. V případě, že tento pokus selže, dojde ke zmíněnému rozpoznání. V případě úspěchu je sekvence dekódována na základě určeného kódování. Nepodaří-li se kódování určit, analyzátor zaznamená tuto chybu, uloží ji jako vlastnost a vrátí ji jako výsledek.

Po úspěšném dekódování je obsah souboru zpracován analyzátorem HTML kódu, který určí, zda se jedná o kód jazyka HTML. Pokud ano, pokusí se z něj extrahovat všechny vložené skripty. Pokud ne, je celý obsah souboru považován za skript.

Pomocí argumentů zadaných při spuštění lze analyzátoru předat informace nezbytné pro jeho činnost (např. cesta ke vstupnímu souboru), ale také dodatečné informace, jež upřesňují chování některých jeho modulů (např. výstupní formát vlastností). Analyzátor lze spustit s následujícími argumenty:

- `src_file`,
- `--max-file-size`,
- `--unlimited-size`,
- `--identification-process`,
- `--start-end-chars`,
- `--output-format` a
- `--python2-interpreter`.

Prvním, povinným argumentem je cesta ke vstupnímu (zdrojovému) souboru. Pokud je cesta k souboru neplatná nebo uživatel nemá právo k souboru přistupovat, dojde k ukončení analyzátoru. Všechny ostatní argumenty jsou volitelné.

Argument `--max-file-size` očekává celé číslo určující maximální povolenou velikost vstupního souboru. Překročí-li velikost vstupního souboru tuto hodnotu, modul činnost analyzátoru ukončí. Výchozí hodnotou je 50 MB (50 milionů bajtů).

Argumentem `--unlimited-size` lze omezení maximální velikosti vstupního souboru vypnout; v takovém případě je argument `--file-size` ignorován.

Pomocí argumentu `--identification-process` lze určit, jak má probíhat proces identifikace. Je-li zadána hodnota `first-match` (výchozí), identifikace skriptu skončí ve chvíli, kdy je určen první typ. Hodnota `full` naopak zajistí, že budou identifikovány všechny typy, kterým skript odpovídá.

Argumentem `--start-end-chars` je možné určit počet znaků, které mají být extrahovány ze začátku a z konce skriptu. Výchozí hodnotou je 30 znaků.

Argument `--output-format` určuje formát výstupu analyzátoru. Platnými hodnotami jsou `json` (výchozí) a `object`. Při hodnotě `json` jsou získané vlastnosti serializovány; výsledek je ve formátu JSON[1]. Při hodnotě `object` je vrácena instance třídy, jež uchovává získané vlastnosti a poskytuje veřejný protokol pro práci s nimi.

Posledním argumentem je `--python2-interpreter`, jímž lze specifikovat příkaz sloužící ke spuštění interpretu jazyka Python verze 2.x. Tato starší verze interpretu je potřebná při identifikaci skriptů typu Python ve verzi 2.x. Výchozí hodnotou argumentu je `python2`.

5.3 Modul pro identifikaci

Načtený skript je podroben identifikačnímu procesu. Analyzátor podporuje celkem pět typů skriptů – jmenovitě skripty napsané v jazycích JavaScript, VBScript, PowerShell, Python a dávkové soubory – identifikace je proto nutná k výběru konkrétních tříd, které budou provádět analýzu daných typů.

Modul provádí identifikaci dvěma způsoby: pomocí externích aplikací a na základě vyhodnocování množiny pravidel. Je-li možné pro rozpoznání typu skriptu použít již existující aplikaci, je na vstup této aplikace přiveden zdrojový kód skriptu a její výstup je poté zpracován modulem. Identifikace vyhodnocováním množiny pravidel se v takovém případě přeskakuje.

V případech, kdy externí aplikaci nelze použít (žádná vyhovující není volně dostupná, žádná z vybraných se neosvědčila apod.) nebo kdy tento způsob identifikace selže, dochází k identifikaci na základě množiny pravidel, která je fixně zaznamenána v kódu analyzátoru.

Každé pravidlo se skládá ze vzoru a váhy a patří právě do jedné kategorie. Váha pravidla může nabývat hodnot -100 až 100 a jejím smyslem je určit důvěryhodnost daného pravidla. Záporná váha je užita tehdy, když má pravidlo snížit důvěryhodnost toho, že se jedná o daný typ skriptu.

Kategorie seskupuje jedno nebo více pravidel. Dle významnosti má přidělenou maximální hodnotu v rozsahu 0 až 100, kterou se může podílet na konečném výsledku identifikace[2]. Minimální hodnota je implicitně nastavena na -100.

Vyhodnocování pravidel a kategorií probíhá v následujícím pořadí:

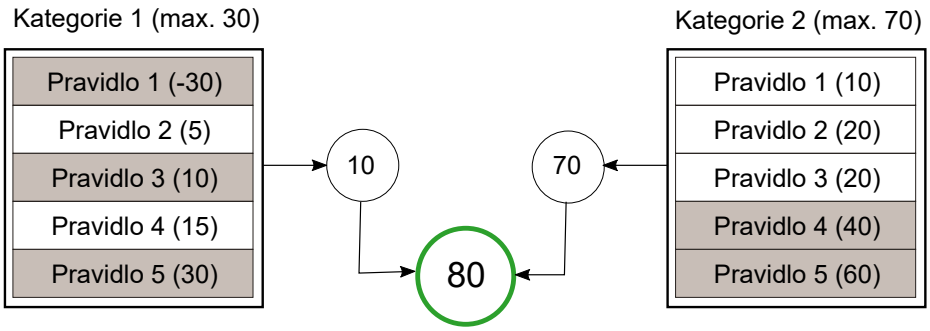
1. Jsou vyhodnocena pravidla v rámci jednotlivých kategorií.
2. Jsou sečteny váhy detekovaných pravidel. Pokud je výsledná hodnota menší než -100 nebo větší než maximální hodnota kategorie, je místo ní použita jedna z hraničních hodnot intervalu.
3. Je proveden součet výsledných hodnot kategorií. Podle potřeby je tento výsledek upraven tak, aby spadal do intervalu 0 až 100.

Celkový součet hodnot kategorií reprezentuje důvěryhodnost toho, že se jedná o daný typ skriptu. Rozsah platných hodnot pro důvěryhodnost je 0 až 100. Každý typ má vlastní konfigurovatelný práh, kterého musí důvěryhodnost dosáhnout, aby byl typ považován za úspěšně identifikovaný.

Pro lepší pochopení je uveden příklad na obrázku 5.2.

[1] JavaScript Object Notation, více o tomto formátu na https://cs.wikipedia.org/wiki/JavaScript_Object_Notation

[2] Analyzátor povoluje nastavit maximální hodnotu na 0, ačkoliv to z praktického hlediska nemá význam.



Obrázek 5.2: Příklad výpočtu důvěryhodnosti

Pravidla v šedě zbarvených rámečcích představují detekovaná pravidla. Váhu má každé pravidlo uvedenou v závorce. Nejprve dojde k sečtení vah detekovaných pravidel v rámci kategorií. Součet kategorie 1 je 10, což je hodnota spadající do intervalu -100 až 25, a není tedy nutné ji upravovat. Součet kategorie 2 je 100, nicméně tato hodnota je nad vrchní hranicí intervalu -100 až 70, proto je místo ní použita hodnota 70. Výsledkem součtu hodnot kategorií je číslo 80, které patří do intervalu 0 až 100 a které představuje výslednou důvěryhodnost.

U nejednoznačného výsledku (důvěryhodnost není rovna 100) dochází k porovnání výsledné důvěryhodnosti a prahu pro daný typ. Pokud důvěryhodnost překročí daný práh, je typ považován za úspěšně identifikovaný. Práh je konfigurovatelný pro každý typ zvlášť.

Identifikační proces probíhá buď do doby, než je rozpoznán jeden z podporovaných typů, nebo pro všechny podporované typy nehledě na dílčí výsledky. Pouze u druhé možnosti může být skriptu přiřazeno více typů.

Typy jsou identifikovány na základě nastavitelné priority. Priorita smí nabývat pouze nezáporných hodnot. Čím menší je hodnota, tím větší má typ prioritu.

Přijde-li na vstup modulu pro identifikaci více než jeden skript, je identifikaci podroben každý z nich. Tato situace může vzniknout tehdy, když modul pro načtení vstupních dat extrahuje z kódu psaného v jazyce HTML více než jeden skript.

Jakmile modul pro identifikaci dokončí svou činnost, je řízení předáno modulu pro analýzu, který pokračuje volbou vhodné třídy či vhodných tříd pro analýzu a následně analýzou samotnou.

5.4 Modul pro analýzu

Před začátkem analýzy probíhá kontrola, zda došlo k rozpoznání alespoň jednoho typu skriptu. Pokud by nebyl rozpoznán ani jeden typ, analýza kódu by neproběhla.

Během analýzy jsou ze zdrojového kódu skriptu extrahovány vlastnosti potřebné k tomu, aby soubor mohl být klasifikován službou Clusty. Tyto vlastnosti mohou být společné pro všechny podporované typy skriptů nebo specifické pro konkrétní typ či typy. Vzhledem k odlišným způsobům analýzy každého typu má každý typ vlastní třídu pro analýzu.

Některé vlastnosti nejsou získávány během analýzy tohoto modulu, nýbrž při procesech vykonávaných předchozími moduly. V těchto případech jsou vlastnosti předány na vstup modulu pro analýzu spolu se zdrojovým kódem a identifikovaným typem skriptu. Zpráva o chybě vzniklé při identifikaci je příkladem vlastnosti, která se získává v jiném modulu.

Získané vlastnosti jsou výstupem jak modulu pro analýzu, tak celého analyzátoru. Analyzátor umožňuje reprezentovat výstup jako řetězec ve formátu JSON nebo jako objekt (instanci třídy). První možnost je využita tehdy, když mají být vlastnosti zpracovány dalšími aplikacemi. Formát JSON byl zvolen pro svou jednoduchost a podporu napříč používanými programovacími jazyky. Druhá možnost je vhodná v situacích, kdy má s výsledkem pracovat jiný skript psaný v jazyce Python; tento skript importuje analyzátor jako modul, provede analýzu a s výsledným objektem uchovávajícím extrahované vlastnosti komunikuje přes jeho veřejný protokol.

5.5 Rozšiřitelnost analyzátoru

Ačkoliv v této chvíli analyzátor podporuje pouze pět typů skriptů, v budoucnosti může nastat situace, kdy bude potřeba přidat podporu pro další typy. Přidání podpory nového typu by mělo být z programátorského hlediska co nejjednodušší. Z tohoto důvodu byl při vývoji analyzátoru kladen důraz na jeho modulárnost a na automatizaci některých úloh.

Rozšíření analyzátoru o nový typ skriptu vyžaduje následující kroky:

1. Vytvoření souboru se třídou pro identifikaci nového typu skriptu.
2. Implementace třídy pro identifikaci.
3. Určení prahu úspěšné identifikace a priority.
4. Vytvoření souboru se třídou pro analýzu nového typu skriptu.
5. Implementace třídy pro analýzu.

Názvy obou souborů musí dodržet stejnou konvenci pojmenování, aby s nimi mohl analyzátor korektně pracovat. Názvy souborů musí začínat názvem nového typu skriptu a za ním musí následovat přípona `.py`. Název typu smí obsahovat pouze písmena a až z. Pro podporu typu s názvem Perl by tedy bylo nutné vytvořit dva soubory pojmenované `perl.py` (každý v odpovídajícím adresáři), ve kterých by se nacházely příslušné třídy pro identifikaci a analýzu.

Ostatní úkony obstará analyzátor sám. Veškeré třídy pro identifikaci jsou automaticky importovány. Jakmile je identifikován typ skriptu, dojde k výběru příslušné třídy, jež provede analýzu, a to právě na základě názvu souboru.

Kapitola 6

Implementace

Tato kapitola popisuje závislosti analyzátoru a způsob, jakým byl implementován. V podkapitole 6.1 lze nalézt použité technologie a balíčky třetích stran, zatímco podkapitola 6.2 je věnována struktuře balíčku nejvyšší úrovně. Čtyři zbývající podkapitoly pojednávají o implementačních detailech modulů analyzátoru a rozšiřitelnosti; jedná se o podkapitoly 6.3, 6.4, 6.5 a 6.6.

Návod, jenž popisuje, jak lze analyzátor nakonfigurovat a spustit, případně jak jej použít jako knihovnu, je uveden v příloze C.

Aby nedošlo k záměně pojmů, je nutné vymezit význam slova modul. Zatímco v kapitole 5 bylo slovo modul používáno výhradně pro označení části analyzátoru, v této kapitole je slovo modul uvedeno i v kontextu jazyka Python – značí zdrojový soubor, který lze importovat v jiných modulech či skriptech [11]. Z tohoto důvodu bude v této kapitole část analyzátoru označována slovním spojením „modul analyzátoru“.

6.1 Použité technologie a balíčky

Popisovaný analyzátor byl vyvinut v programovacím jazyce Python. Pro správnou funkčnost analyzátoru je vyžadován interpret jazyka Python verze 3.6 nebo vyšší. Jeden ze skriptů, jenž je součástí analyzátoru, vyžaduje také existenci interpretu verze 2.x.

K úspěšnému spuštění analyzátoru či jeho testů je též nutné stáhnout externí balíčky, jež nejsou nativní součástí interpretu jazyka Python. Jmenovitě se jedná o balíčky `chardet`[1], `esprima`[2], `flake8`[3], `mypy`[4] a `nose2`[5].

Analyzátor byl vyvíjen s ohledem na to, aby byl multiplatformní. Mezi podporované platformy patří operační systémy Windows a Linux.

6.2 Struktura balíčku

Vzhledem k velkému množství zdrojových souborů jsou tyto soubory členěny do balíčků a modulů. Balíčkem nejvyšší úrovně je `script_analysis`. Níže jsou popsány balíčky a moduly, které tvoří podstatnou část analyzátoru. Kompletní struktura se stručnými popisky je uvedena v příloze A.

[1] https://github.com/chardet/chardet
[2] https://github.com/Kronuz/esprima-python
[3] https://github.com/PyCQA/flake8
[4] https://github.com/python/mypy
[5] https://github.com/nose-devs/nose2

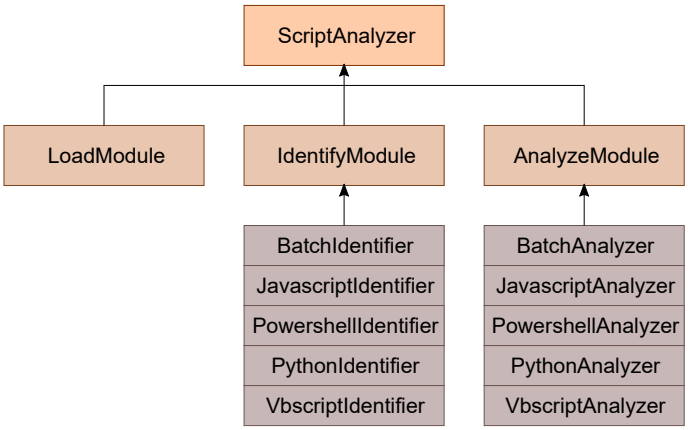
Modul obsahující třídu analyzátoru `ScriptAnalyzer` je nazván `script_analyzer.py`. Během instanciace této třídy jsou vytvořeny tři objektové atributy, které uchovávají instance tříd modulů analyzátoru.

Třídy modulů analyzátoru se nachází v balíčku `modules/`. Jedná se o třídy `LoadModule`, `IdentifyModule` a `AnalyzeModule`.

Ve třídě `IdentifyModule` dochází k importování všech tříd určených k identifikaci typů skriptů. Soubory těchto tříd jsou uloženy v balíčku `identify/`. Název každého souboru odpovídá názvu typu skriptu, který identifikuje.

Třída `AnalyzeModule` importuje konkrétní třídu pro analýzu na základě identifikovaného typu skriptu. Soubory s třídami pro analýzu typů skriptů se nacházejí v balíčku `analyze/` a jejich názvy odpovídají názvům typů skriptů, které analyzují.

Závislost jednotlivých tříd je znázorněna na obrázku 6.1.



Obrázek 6.1: Diagram znázorňující závislost jednotlivých tříd. Nejedná se o třídní dědičnost

6.3 Modul pro načtení vstupních dat

V této podkapitole jsou uvedeny implementační detaily k jednotlivým úlohám vykonávaných modulem pro načtení vstupních dat. Načítání a dekódování vstupního souboru je popsáno v sekci 6.3.1, extrakci skriptů z kódu HTML je věnována sekce 6.3.2 a zpracování argumentů se věnuje sekce 6.3.3.

6.3.1 Načtení vstupního souboru

Vstupní soubor je otevřen ve dvou režimech: v režimu čtení a v binárním režimu (`rb`). Obsah souboru je tedy získán v binární, nedekódované formě, která odpovídá datovému typu `bytes`.

Modul analyzátoru se nejprve pokusí dekódovat posloupnost bajtů pomocí kódování UTF-8, a to metodou `decode`. Pokud se dekódování nezdaří a metoda vyvolá výjimku `UnicodeDecodeError`, přejde se k rozpoznávání kódování.

Automatické rozpoznávání kódování bylo přidáno kvůli potřebě analyzovat vzorky, které mohou pocházet z různých zdrojů, a které tudíž mohou využívat různá, mnohdy i exotická kódování. Ke zjištění kódování, kterým byl vstupní soubor zakódován, je použit modul `chardet`.

Tento modul umí rozpoznat několik desítek podporovaných druhů kódování textových souborů [15]. Jmenovitě se jedná o:

- ASCII, UTF-8, UTF-16 (2 varianty), UTF-32 (4 varianty),
- Big5, GB2312, EUC-TW, HZ-GB-2312, ISO-2022-CN (tradiční a zjednodušená čínština),
- EUC-JP, SHIFT-JIS, CP932, ISO-2022-JP (japonština),
- EUC-KR, ISO-2022-KR (korejština),
- KOI8-R, MacCyrillic, IBM855, IBM866, ISO-8859-5, windows-1251 (cyrilice),
- ISO-8859-5, Windows-1251 (bulharština),
- ISO-8859-1, Windows-1252 (jazyky západní Evropy),
- ISO-8859-7, Windows-1253 (řečtina),
- ISO-8859-8, Windows-1255 (různé varianty hebrejštiny) a
- TIS-620 (thajština).

Detekci provádí instance třídy `UniversalDetector`. Vzniklá instance rozpoznává kódování přírůstkově – pomocí metody `feed` jsou jí postupně předávány části z přečtené sekvence bajtů, ale pouze do doby, než je schopna kódování spolehlivě určit. Tento způsob nemusí vyžadovat zpracování celé sekvence, a je proto vhodný pro objemné soubory.

Výsledek je uložen v atributu `result` této instance. Je reprezentován jako slovník, který obsahuje tři klíče – `encoding`, `confidence` a `language` – uchovávající údaje o detekovaném kódování.

Hodnotou klíče `encoding` je detekované kódování. Tato hodnota je použita při dekódování posloupnosti bajtů. V případě, že se nepodařilo kódování určit, vrací funkce hodnotu `None`.

Hodnoty zbylých klíčů `confidence` (důvěryhodnost, že se skutečně jedná o detekované kódování) a `language` (konkrétní detekovaný jazyk) nejsou v analyzátoru použity, a není proto nutné se jim dále věnovat.

Pokud selhal i pokus o rozpoznání kódování nebo se soubor i přes to nepodařilo dekódovat, je soubor prohlášen za nedekódovatelný. Takový stav vede k uložení této chyby jako vlastnost `encoding_error`.

6.3.2 Extrakce skriptů z HTML

Zdrojový kód je po dekódování podroben analýze, jež vyhodnotí, zda se jedná o kód psaný v jazyce HTML a zda se v něm nacházejí skripty. Analýzu provádí instance třídy `ScriptHTMLParser`, která dědí ze třídy `HTMLParser` importované ze standardního modulu `html.parser`.

K detekci skriptů dochází tak, že je při procházení stromu elementů kontrolován název každého elementu. Tuto úlohu vykonává metoda `handle_starttag`. Je-li nalezen element s názvem `script`, nastaví metoda hodnotu objektového atributu `in_script` na `True`.

Dále je kontrolován obsah atributů `type` a `language`, jsou-li v elementu přítomny. Pomocí nich lze již v tomto modulu analyzátoru určit, zda se jedná o skript podporovaného

typu. Obsah atributů je srovnán s typy standardu MIME, které jsou vyhrazeny pro JavaScript a VBScript. Pokud se obsah atributu shoduje s některým typem, je určený typ skriptu zaznamenán. Vyhrazené typy standardu MIME jsou uvedeny v tabulce 6.1.

	`type`	`language`
JavaScript	application/javascript	javascript
	application/x-javascript	x-javascript
	application/ecmascript	ecmascript
	application/x-ecmascript	x-ecmascript
	text/javascript	jscript
	text/x-javascript	livescript
	text/ecmascript	javascript1.0
	text/x-ecmascript	javascript1.1
	text/jscript	javascript1.2
	text/livescript	javascript1.3
	text/javascript1.0	javascript1.4
	text/javascript1.1	javascript1.5
	text/javascript1.2	
	text/javascript1.3	
	text/javascript1.4	
	text/javascript1.5	
VBScript	application/x-vbs	vbs
	text/vbs	vbscript
	text/vbscript	

Tabulka 6.1: Tabulka uvádějící podporované typy standardu MIME pro atributy `type` a `language` elementu `script`

Uložení obsahu elementu `script` probíhá v metodě `handle_data`. K uložení dojde jen tehdy, je-li hodnota `in_script` rovna `True`. Určený typ skriptu (pokud byl určen, jinak prázdný řetězec) a obsah elementu jsou přidány do seznamu všech skriptů, který je uložen v objektovém atributu `scripts`.

Metoda `handle_endtag` nastaví po zpracování elementu hodnotu `in_script` zpět na `False`.

Jakmile je kód HTML zpracován, dojde ke sloučení skriptů stejného typu; skripty bez určeného typu jsou ponechány bez sloučení. Výsledkem je slovník se sloučenými a neidentifikovanými skripty. V případě, že žádný skript nebyl nalezen, je vrácen prázdný slovník.

Vedlejším produktem analyzátoru HTML je také vlastnost `in_html`, která nese informaci o tom, zda skripty jsou, nebo nejsou součástí HTML.

6.3.3 Zpracování argumentů

Argumenty, které upravují chování modulů analyzátoru, lze zadávat při spuštění. Více informací o účelu jednotlivých argumentů se lze dočíst v podkapitole 5.2.

Zpracování argumentů probíhá pomocí třídy `ArgumentParser` ze standardního modulu `argparse`. V modulu pro načtení vstupních dat je vytvořena instance této třídy, do níž jsou následně přidávány jednotlivé argumenty pomocí metody `add_argument`. Po přidání všech požadovaných argumentů je zavolána metoda `parse_args`.

Instance zmíněné třídy se kromě zpracování očekávaných argumentů postará i o veškeré chybné stavy. V případě, že bude chybět povinný argument či bude zadán nepodporovaný argument, dojde k ukončení analyzátoru a vyvolání výjimky. Totéž nastane v situaci, kdy nebude odpovídat datový typ daného argumentu (např. pro argument `--file-size` bude zadána jiná než celočíselná hodnota).

6.4 Modul pro identifikaci

Tato podkapitola se zabývá implementačními detaily jednotlivých procesů modulu pro identifikaci. Sekce 6.4.1 popisuje automatické importování identifikačních tříd, v sekci 6.4.2 se uvádí postup při identifikaci jednoho a více skriptů a sekce 6.4.3 a 6.4.4 jsou věnovány způsobu identifikace.

6.4.1 Importování identifikačních tříd

Před začátkem samotné identifikace je nejdříve nutné importovat všechny třídy identifikující typy skriptů. Třídy nejsou do kódu importovány manuálně programátorem, ale automaticky analyzátorem, a to proto, aby bylo případné rozšíření analyzátoru o podporu nového typu skriptu co nejsnazší. Tuto činnost vykonává metoda `_import_identifiers`.

Soubory s třídami se nacházejí ve složce `identify/`. Každý soubor musí být pojmenován po typu skriptu a malými písmeny a končit příponou `.py`; název, jenž by neodpovídal těmto podmínkám, by mohl způsobit selhání analyzátoru.

Seznam souborů je získán pomocí funkce `listdir` ze standardního modulu `os`. Soubory jsou zbaveny přípon a po jednom importovány jako moduly pomocí funkce `import_module` ze standardního modulu `importlib`. Následně probíhá kontrola, zda k importovanému souboru existuje odpovídající soubor ve složce `analyze/`. Pokud neexistuje, je vyvolána výjimka `IdentifierError` informující o chybějící třídě pro analýzu.

Jakmile je soubor importován, dojde k vytvoření názvu třídy, která z něj má být získána. Název třídy je odvozen od názvu souboru – název souboru je spojen s řetězcem `"Identifier"` a první písmeno vzniklého názvu je převedeno na velké. Název třídy ze souboru `perl.py` by byl `PerlIdentifier`.

Název třídy je použit ke kontrole, zda se třída nachází v importovaném souboru. Pokud ne, dojde k vyvolání výjimky `IdentifierError`. Pokud ano, je získána funkcí `getattr`. Z této třídy je navíc získána její priorita.

Priorita třídy, třída a název souboru (respektive typu skriptu) jsou jako trojice uloženy do seznamu získaných tříd. Jakmile seznam obsahuje všechny třídy, je pomocí metody `sort` seřazen vzestupně podle priority. Tím modul pro identifikaci zajistí, že budou typy rozpoznávány v pořadí dané prioritou.

Vzniklý seznam obsahující všechny získané a seřazené třídy je návratovou hodnotou metody `_import_identifiers`.

6.4.2 Proces identifikace skriptů

Jakmile má modul pro identifikaci k dispozici seznam s importovanými třídami, může zahájit rozpoznávací proces. O ten se stará metoda `identify_script_types`.

Způsoby rozpoznávání se mírně liší za základě toho, zda se jedná o samostatný skript, nebo skripty získané z kódu jazyka HTML. Odlišit je lze na základě datového typu para-

metru `script`. Jedná-li se o řetězec (dat. typ `str`), byl předán samostatný skript. Jedná-li se o slovník (dat. typ `dict`), byly předány skripty z HTML.

Každý skript je rozpoznáván buď všemi třídami, nebo tolika z nich, dokud není jeden z podporovaných typů identifikován[6]. Z každé třídy je vytvořena instance a z instance je volána metoda `identify`, které jsou jako argumenty předány skript určený k identifikaci a režim identifikace (interní, externí či obojí). Návratová hodnota metody představuje důvěryhodnost toho, že se jedná o typ skriptu, jehož rozpoznávání třída implementuje. Pokud je důvěryhodnost větší nebo rovna hodnotě definovaného prahu pro daný typ, je do slovníku identifikovaných typů skriptů přidán seznam obsahující dva prvky: prvním je skript, druhým jsou vlastnosti skriptu, jež byly získány buď během identifikace, nebo v modulu pro načtení vstupních dat. Klíči slovníku jsou názvy typů skriptů.

Identifikace skriptů z HTML se oproti postupu v odstavci výše liší ve dvou bodech. Prvním rozdílem je, že skripty vyžadují identifikaci jen tehdy, nebyl-li jejich typ určen již v modulu pro načtení vstupních dat. Druhý rozdíl se týká výsledku identifikace. Zatímco při úspěšné identifikaci samostatného skriptu je do výsledného slovníku přidán nový seznam obsahující skript a vlastnosti, u skriptů z HTML se manipuluje s předdefinovaným seznamem. Skript je sloučen se skripty v seznamu a vlastnosti jsou doplněny k vlastnostem v seznamu.

Pokud nastane situace, kdy jednomu či více skriptům nelze přiřadit žádný z podporovaných typů, je k vlastnostem dosavadně identifikovaných typů přidána vlastnost pojmenovaná `unidentified`, jež o vzniklé situaci informuje. Neidentifikovatelné skripty nejsou předávány modulu pro analýzu.

Vzniklý slovník identifikovaných typů může vypadat kupříkladu takto:

```
{
    "javascript": [
                   "var a;\r\nvar b;",
                   {"confidence": 95, "in_html": True}
                  ],

    "vbscript": [
                  "\n\nDim a\nDim b\n\n",
                  {"confidence": 100, "in_html": True}
                 ]
}
```

6.4.3 Identifikace skriptů externími aplikacemi

Prvním způsobem, jak identifikovat typ skriptu, je použití externích aplikací, které znají syntaktická pravidla pro daný typ a z poskytnutého zdrojového kódu vytvářejí abstraktní syntaktický strom. Narazí-li na nepodporovanou syntaxi či neznámý token, je to signál toho, že se o daný typ skriptu nejedná.

Tento způsob je také označován jako externí a musí být při volání metody `identify` povolen.

[6] Preferovaná možnost může být zvolena během spuštění analyzátoru.

JavaScript

K identifikaci skriptů typu JavaScript byla využita knihovna `esprima`. Jak je uvedeno v [7], je to knihovna, jejímž účelem je provést lexikální a syntaktickou analýzu programů psaných v jazyce JavaScript. Tato knihovna byla zvolena jako ta nejvhodnější, a to z následujících důvodů:

- pravidelné aktualizace na základě vývoje standardu ECMAScript,
- dostupnost jak lexikální, tak syntaktické analýzy,
- podpora syntaxe pro JSX[7].

Funkce, jež pochází z této knihovny a jež jsou použity při identifikaci, jsou nazvány `parseScript` a `parseModule`. Modul pro identifikaci se nejdříve pokusí vytvořit abstraktní syntaktický strom funkcí `parseScript`. Pokud tento pokus selže (dojde k vyvolání výjimky definované knihovnou), při druhém pokusu je použita funkce `parseModule`. Pokud selže i druhý pokus, přejde modul pro identifikaci k rozpoznávání na základě vyhodnocování množiny pravidel.

Selháním některé z funkcí dojde k vygenerování chybového hlášení, které upřesňuje vzniklou lexikální či syntaktickou chybu. Pokud k selhání dojde, vyprodukované hlášení je uloženo jako vlastnost `external_error`.

Oběma funkcím jsou jako argumenty předány zdrojový kód skriptu a slovník se dvěma prvky – `jsx` a `tolerant`, oba nabývající hodnoty `True`. Prvek `jsx` zajišťuje podporu syntaxe pro JSX, prvek `tolerant` aktivuje při syntaktické analýze tzv. tolerantní režim, během kterého jsou některé syntaktické chyby ignorovány (viz [7]).

Python

Jazyk Python má vlastní prostředky, pomocí kterých lze určit, zda je zdrojový kód psaný v něm validní. Těmito prostředky jsou standardní modul `ast` a funkce `compile`.

Modul `ast` je popsán jako prostředek umožňující zpracovávat abstraktní syntaktické stromy aplikací psaných v jazyce Python [3]. Tento modul nebyl analyzátorem využit, a to kvůli jeho nadbytečné funkcionalitě. Modul pro identifikaci nepotřebuje metody pro práci se vzniklým stromem, stačí informace o tom, že byl ze zdrojového kódu vytvořen.

Funkce `compile` vytváří ze zdrojového kód buď objekt spustitelný funkcemi `exec` či `eval`, nebo objekt stromu [3]. Modul pro identifikaci využívá tuto funkci k ověření správného syntaktického zápisu zdrojového kódu.

Výše uvedená funkce je spuštěna s následujícími hodnotami parametrů:

- za `source` je dosazen zdrojový kód skriptu,
- za `filename` je dosazen prázdný řetězec a
- za `mode` je dosazen řetězec `"exec"`.

Zbylým parametrům jsou ponechány jejich výchozí hodnoty. Díky těmto hodnotám bude ze zdrojového kódu vytvořen objekt spustitelný funkcí `exec`, ale jen v případě, že je kód validní. Vyvolá-li tato akce výjimku `SyntaxError` nebo `ValueError`, byla nalezena lexikální či syntaktická chyba.

[7] Více informací na https://en.wikipedia.org/wiki/React_(JavaScript_library)#JSX

Vyvolání výjimky ovšem neznamená, že se nejedná o skript typu Python. Zdrojový kód skriptu je po nalezení chyby znovu zpracován, tentokrát ale skriptem `python2.py`, který je spuštěn interpretem jazyka Python verze 2.x. Ve verzi 2.x je totiž několik rozdílů v syntaxi, které interpret verze 3.x považuje za chybu.

Skript je spuštěn pomocí funkce `run` ze standardního modulu `subprocess`, jako argument při spuštění má hodnotu `identify`. Skript zopakuje celý proces s funkcí `compile` tak, jak je uveden výše, a výsledek uloží do slovníku pod klíč `is_python2`. Slovník je převeden do formátu JSON a vypsán na standardní výstup. Standardní výstup je zachycen po dokončení funkce `run` a převeden zpět na slovník. Pokud klíč `is_python2` obsahuje hodnotu `True`, jedná se o zdrojový kód psaný v jazyce Python 2.x.

Jestliže došlo k vyvolání výjimky i ve verzi 2.x, přejde modul pro identifikaci k rozpoznávání na základě vyhodnocování množiny pravidel a chybové hlášení specifikující nalezenou chybu uloží jako vlastnost `external_error`.

Ostatní typy skriptů

U zbylých typů skriptů (VBScript, PowerShell a dávkový soubor) se také uvažovalo o použití externích aplikací, nicméně buď nebyly žádné nalezeny, nebo nevyhovovaly požadavkům stanoveným buď na danou aplikaci (např. pokrytí celé gramatiky daného jazyka či typu skriptu), nebo na analyzátor (např. podpora více operačních systémů).

Pro skripty psané v jazyce VBScript nebyla nalezena žádná vhodná aplikace. Byl vyzkoušen syntaktický analyzátor dostupný na adrese `https://archive.codeplex.com/?p=vbsparser`, jenže se jej nepodařilo přeložit do binární podoby a nebyla k němu nalezena žádná dokumentace.

Ani pro skripty typu PowerShell nebyly nalezeny použitelné syntaktické analyzátory. Jedinou nalezenou možností byla kontrola zdrojového kódu přímo v interpretu jazyka PowerShell, kde jsou k tomu dostupné funkce (podobně jako u jazyka Python). Takový interpret bohužel není dostupný na distribucích systému Linux, což by vedlo k tomu, že by analyzátor nebyl spustitelný na tomto operačním systému.

Pro dávkové soubory existuje nástroj pojmenovaný BatCodeCheck[8]. Cílem tohoto nástroje je hledání a zvýrazňování chyb v dávkových souborech. Nejedná se ovšem o syntaktický analyzátor, a má tedy svá omezení – nedokáže zpracovávat zdrojový kód, který je záměrně psán tak, aby programátora co nejvíce zmátl (tzv. *obfuscated code*). Vzhledem k faktu, že právě takový zdrojový kód může do antivirové společnosti přijít, není možné tento nástroj použít.

6.4.4 Vyhodnocování množiny pravidel

Pokud selže externí aplikace nebo není žádná k dispozici, je použit druhý způsob identifikace typu skriptu – vyhodnocování množiny pravidel. Podrobnosti ke způsobu vyhodnocování a rozsahu platných hodnot jsou probírány v podkapitole 5.3.

Tento způsob je v kódu analyzátoru označován jako interní a jeho použití musí být povoleno při volání metody `identify`.

Uvedený způsob identifikace probíhá v metodě `_match_rules`. Ta se skládá ze dvou částí: první obsahuje kategorie pravidel a druhá samotné vyhodnocování a korekci výsledku.

Pravidlo je reprezentováno jako dvojice skládající se ze vzoru (řetězec) a váhy (celé číslo). Vzorem je regulární výraz kompatibilní s funkcemi ze standardního modulu `re`. Váha

[8] Nástroj je dostupný na adrese `https://www.robvanderwoude.com/battech_batcodecheck.php`

představuje důvěryhodnost daného pravidla a může nabývat hodnot -100 až 100. Pokud by váha překročila stanovené limity, byla by použita příslušná hraniční hodnota.

Každé pravidlo musí být přidruženo právě do jedné kategorie. Kategorie je tvořena slovníkem se dvěma klíči: `rules` a `max_conf`. Klíč `rules` sdružuje jedno čí více pravidel, a to v n-tici. Pod klíčem `max_conf` je uložena číselná hodnota, jež určuje maximální důvěryhodnost, které může kategorie dosáhnout. Stejně jako váha pravidla je i maximální důvěryhodnost upravena tak, aby její hodnota spadala do přípustného intervalu, v tomto případě 0 až 100.

K vyčíslení důvěryhodnosti dané kategorie slouží metoda `_calc_conf`. Jejím vstupem je pět argumentů, a to zdrojový kód, n-tice s pravidly, maximální důvěryhodnost kategorie, modifikátory upravující chování regulárního výrazu a slovník, jehož obsah určuje, kolikrát musí být pravidlo ve zdrojovém kódu nalezeno.

Vyhodnocování probíhá tak, že jsou postupně procházena všechna pravidla z n-tice. Způsob, jakým je pravidlo vyhodnoceno, se liší na základě pátého argumentu. Je-li jeho hodnotou `None` (výchozí), vzor pravidla bude dosazen do funkce `search` ze standardního modulu `re`. Pokud je argumentem slovník, bude místo funkce `search` použita funkce `finditer` z téhož modulu.

První způsob hledá vzor ve zdrojovém kódu jen tak dlouho, dokud nenarazí na první shodu. Pakliže ke shodě dojde, váha pravidla je přičtena k proměnné uchovávající součet vah detekovaných pravidel.

Druhý způsob oproti prvnímu vyžaduje více než jednu shodu. Minimální počet výskytů vzoru je určen celočíselnou hodnotou uloženou ve slovníku pod klíčem `min`. Každá nalezená část zdrojového kódu, která odpovídá vzoru, musí být navíc unikátní. Pokud výsledek vyhodnocení nesplňuje tato dvě pravidla, nemůže být váha přičtena k celkovému součtu detekovaných pravidel. Uvažujme vzor `r"\b(var|let|function|return)\b"` a slovník `{"min": 3}`. Aby se váha pravidla s tímto vzorem mohla podílet na celkovém součtu, musela by být nalezena alespoň tři rozdílná klíčová slova z uvedeného vzoru. Výskyt třech klíčových slov `var` či dvou `var` a jednoho `function` by k přičtení váhy nevedlo.

Celkový součet může dosáhnout maximální hodnoty kategorie dříve, než dojde k vyhodnocení všech pravidel téže kategorie. Nastane-li taková situace, zbylá pravidla jsou přeskočena a vyhledávání jejich vzorů se neprovádí.

Výsledná důvěryhodnost kategorie je upravena tak, aby nepřekračovala stanové maximum a minimum (uloženo v třídním atributu `lower_bound`), a poté je vrácena metodou `_calc_conf`. Všechny vyčíslené důvěryhodnosti kategorií jsou sečteny a výsledek součtu je upraven tak, aby byl v rozsahu 0 až 100. Po této korekci je vrácen metodou `_match_rules`.

6.5 Modul pro analýzu

Předmětem této podkapitoly jsou detaily k implementaci analýzy skriptu. Podkapitola je rozdělena na čtyři části: sekce 6.5.1 popisuje způsob výběru požadované třídy pro analýzu a průběh analýzy, sekce 6.5.2 upřesňuje, jak probíhá analýza konkrétního typu a sekce 6.5.3 uvádí, jak je realizována transformace do výstupních formátů.

6.5.1 Proces analýzy skriptů

Modul pro analýzu zahájí svou činnost zavoláním metody `analyze_scripts`. Jejími vstupními parametry jsou slovník identifikovaných typů skriptů a argumenty zadané při spuštění.

Metoda iteruje nad všemi rozpoznanými typy ze slovníku a výsledky ukládá do slovníku analyzovaných typů skriptů.

Prvním krokem je importování třídy pro analýzu. Soubor uchovávající tuto třídu je importován z balíčku `analyze/` jako modul, a to funkcí `import_module`. Pokud soubor neexistuje, je vyvolána výjimka `AnalyzerError`. Název třídy je odvozen od typu skriptu – k názvu typu je připojen řetězec `"Analyzer"` a první písmeno vzniklého názvu je převedeno na velké. Funkcí `hasattr` je následně ověřeno, zda importovaný modul obsahuje třídu s tímto názvem. Pokud ne, dojde rovněž k vyvolání výjimky `AnalyzerError`. Po úspěšném ověření je třída získána funkcí `getattr`.

Ze získané třídy je vytvořena instance, jíž jsou předány zdrojový kód a argumenty zadané při spuštění. Vytvořená instance obsahuje metodu `analyze`, jež vykoná analýzu poskytnutého zdrojového kódu. Návratovou hodnotou metody je slovník s extrahovanými vlastnostmi.

Slovník s vlastnostmi je doplněn o vlastnosti, jež byly získány během činnosti předchozích dvou modulů analyzátoru, a je uložen do slovníku analyzovaných typů; jako klíč je použit název typu. Pokud nebyl identifikován ani jeden typ, slovník analyzovaných typů zůstane prázdný.

6.5.2 Analýza konkrétního typu

Analýzu konkrétního typu provádí již zmíněná metoda `analyze`. Metoda akceptuje volitelný řetězcový parametr `mode`, jímž lze povolit či zakázat různé způsoby analýz. Hodnota `"i"` povoluje pouze interní analýzu, `"e"` pouze externí a `"ie"`, což je výchozí hodnota, povoluje oba způsoby analýzy.

Slovním spojením interní analýza je označován způsob extrakce vlastností, který iteruje nad jednotlivými znaky zdrojového kódu skriptu a ukládá vlastnosti na základě jejich lexikálních a syntaktických pravidel. Metoda, která provádí interní analýzu, je nazvána `_extract_traits_manually`. Její kód se liší pro každý typ skriptu, poněvadž každý typ má svá vlastní lexikální a syntaktická pravidla.

Externí analýza využívá k získání vlastností externí nástroje a moduly. Způsob, jakým analyzují, je u všech společný – všechny rozdělují zdrojový kód na lexémy (reprezentované jako tokeny), které jsou na základě jejich typu přiřazeny k jednotlivým vlastnostem. Kód externí analýzy je uložen v metodě `_extract_traits_from_tokens`; vstupem metody je seznam s tokeny. Externí analýza je použita pouze u typů JavaScript a Python.

K tokenizaci je u skriptů psaných v jazyce JavaScript použita funkce `tokenize`. Funkce je schopna extrahovat veškeré vlastnosti specifické pro jazyk JavaScript.

Jazyk Python poskytuje funkci `generate_tokens` ze standardního modulu `tokenize`, jež umí generovat tokeny z vlastního zdrojového kódu. Funkce vyžaduje na vstupu paměťový proud (angl. *in-memory stream*), na který lze zavolat metodu `readline`, proto je zdrojový kód převeden na tento typ proudu pomocí třídy `StringIO` (standardní modul `io`). Z tokenů je možné získat všechny specifické vlastnosti typu Python.

Před zahájením analýzy jsou inicializovány všechny struktury, do kterých jsou vlastnosti ukládány. Kromě standardních datových struktur jako seznam či slovník je použita i datová struktura `defaultdict` ze standardního modulu `collections`.

6.5.3 Výstupní formát

Aby výsledky metody `analyze_scripts` byly použitelné, musí být převedeny do jednoho z podporovaných výstupních formátů. K převodu je použita metoda `output_format`, jejími parametry jsou slovník analyzovaných typů skriptů a požadovaný formát ve formě řetězce.

JSON

Prvním podporovaným formátem je JSON. Serializace slovníku do formátu JSON je realizována pomocí funkce `dumps` ze standardního modulu `json`. Aby ve výsledku byly jednotlivé položky odsazené a seřazené, jsou za parametry `sort_keys` a `indent` této funkce dosazeny hodnoty `True` a `4`. Pokud by metoda během serializace selhala (např. z důvodu nekompatibilní datové struktury), modul pro analýzu by vyvolal výjimku `AnalyzerError`, jež by o vzniklé chybě zpravila uživatele.

Níže je uvedena krátká ukázka výstupu převedeného do formátu JSON. Některé z vlastností nejsou uvedeny. Nezkrácená verze je k nahlédnutí v příloze B.

```
"javascript": {
    "confidence": 100,
    "keywords": {
        "var": 1
    },
    "line_comments": [
        " chrome extension background bootstrap"
    ],
    "line_comments_count": 1,
}
```

Objekt

Druhým podporovaným formátem je objekt. Na rozdíl od první varianty, která slovník analyzovaných typů skriptů transformuje na řetězec, objekt nadále uchovává vlastnosti ve slovníku a také poskytuje metody, s jejichž pomocí lze ke slovníku přistupovat.

Tento objekt je instancí třídy `TraitsContainer`. Konstruktoru je jako argument předán slovník, ten je uložen jako atribut objektu. Třída obsahuje dva typy metod: pro přístup k identifikovaným typům a pro přístup k vlastnostem konkrétního typu.

N-tici uchovávající všechny identifikované typy lze získat pomocí metody, jež je nazvána `get_identified_script_types`. Prázdná n-tice bude vrácena v případě, že nebyl identifikován ani jeden typ.

K ověření, zda se mezi identifikovanými typy skriptů nachází konkrétní typ či typy, slouží metoda `script_type_exists`. Parametrem metody může být buď řetězec, seznam nebo n-tice. V seznamu či n-tici je možné uvést více typů, jejichž existence má být ověřena. Stejného výsledku lze dosáhnout i použitím operátoru `in`.

```
result = obj.script_type_exists(("javascript", "vbscript"))
result = ("javascript", "vbscript") in obj
```

K získání vlastností konkrétního typu lze využít třech způsobů. Prvním způsobem je metoda `get_all_traits`, jež přijímá jako parametr typ skriptu a vrací slovník s vlastnostmi pro daný typ. Druhým způsobem je přístup k vlastnostem pomocí hranatých závorek a klíče,

kterým je typ; chování je totožné jako u prvního způsobu. Třetím a posledním způsobem je metoda `get_trait`, která očekává dva parametry – typ skriptu a název vlastnosti – a vrací hodnotu vybrané vlastnosti pro daný typ.

```
traits = obj.get_all_traits("javascript")
traits = obj["javascript"]
trait = obj.get_trait("javascript", "in_html")
```

Poslední metodou je `trait_exists` a jejím úkolem je ověřit existenci vlastnosti pro daný typ. Jejími parametry jsou typ a název vlastnosti.

6.6 Rozšiřitelnost analyzátoru

Obecný postup pro rozšíření analyzátoru o nový typ skriptu byl již nastíněn v kapitole 5. V této sekci jsou uvedeny konkrétní implementační detaily ke zmíněnému postupu – jaké musí třídy pro identifikaci a analýzu splňovat náležitosti, co musí definovat a kde mají být umístěny soubory, jež tyto třídy uchovávají.

Obě třídy musí být definovány ve vlastním souboru, který je umístěn v příslušné složce (`identify/` či `analyze/`). Názvy obou souborů odpovídají názvu nového typu. Příkladem mohou být soubory `perl.py`.

Třída pro identifikaci nového typu musí mít právě jednu rodičovskou třídu, a tou je abstraktní třída `AbstractIdentifier`. Název nové třídy je složen z prefixu a sufixu; prefixem je název nového typu a sufixem je slovo `Identifier`. Platným názvem je například `PerlIdentifier`.

Aby třída mohla korektně identifikovat nový typ, je nutné, aby byly implementovány tři metody – `identify`, `get_additional_traits` a `_match_rules` – a definovány dva atributy – `priority` a `threshold`. První metoda provádí externí a interní identifikaci typu, druhá vrací vlastnosti získané během identifikace a třetí vyhodnocuje množiny pravidel identifikující daný typ. Třídní atribut `priority` určuje prioritu při procesu identifikace a objektový atribut `threshold` stanovuje minimální hodnotu, které musí být během rozpoznávání dosaženo, aby byl typ považován za identifikovaný.

Třída pro analýzu dědí z abstraktní třídy `AbstractAnalyzer`. I její název má jako prefix název nového typu, ale sufixem je slovo `Analyzer`. Názvem třídy pro ukázkový typ Perl by byl `PerlAnalyzer`.

Dvě abstraktní metody musí být definovány, aby bylo možné skript nového typu analyzovat: `analyze` a `_initialize_traits`. První metoda řídí analýzu skriptu daného typu, zatímco druhá inicializuje struktury, do kterých jsou ukládány extrahované vlastnosti. Třída nevyžaduje definici žádných atributů.

Kapitola 7

Testování a výsledky

Předposlední kapitolou je kapitola věnující se testování a celkovým výsledkům. V podkapitole 7.1 se uvádí, jaké úrovně a druhy testů byly provedeny a jaké prostředky k tomu byly použity. Podkapitola 7.2 podrobně popisuje, jakých výsledků analyzátor dosahuje při identifikaci a analýze.

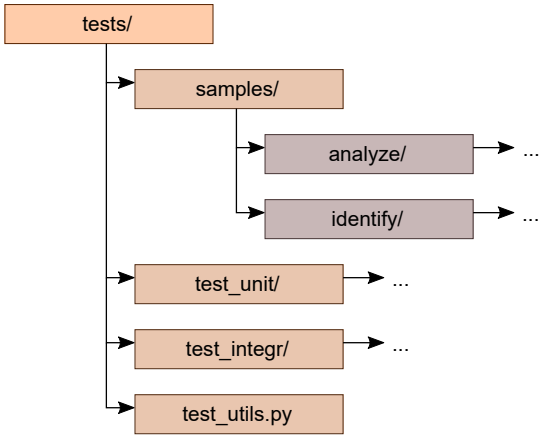
7.1 Testování a druhy testů

K ověření funkčnosti všech modulů analyzátoru je využito dvou úrovní testů – jednotkových a integračních. Jednotkové testují jednotlivé části modulů analyzátoru, integrační testují celé moduly a celý analyzátor.

Vedlejším výstupem testování je také procentuální pokrytí kódu (angl. *code coverage*), které umí odhalit fragmenty zdrojového kódu, které nejsou otestovány.

Testován je i samotný zdrojový kód analyzátoru, a to na zásady správného formátování kódu dle doporučení vývojářů jazyka Python a na správné zacházení s datovými typy (statická analýza datových typů).

Veškeré soubory určené k testování jsou uloženy v balíčku nejvyšší úrovně `tests/`. Ten se dělí na dílčí balíčky, z nichž jeden uchovává testovací vzorky (`samples/`) a zbylé dva skripty s testy (`test_unit/` a `test_integr/`). Kompletní hierarchie testovacího balíčku je vyobrazena na obrázku 7.1.



Obrázek 7.1: Diagram znázorňující hierarchii testovacích souborů

7.1.1 Jednotkové testování

Názvy skriptů s jednotkovými testy začínají prefixem `test_unit`, za nímž následuje krátký identifikátor (např. `test_unit_javascript.py`). Skripty jednotkových testů jsou umístěny v balíčku `test_unit/`.

K realizaci testů byl zvolen standardní modul `unittest`. Pro každou funkci je vytvořena třída, jež představuje jeden testovací případ. Tato třída dědí od třídy `TestCase` pocházející ze zvoleného modulu. Jeden test daného případu odpovídá jedné metodě dané třídy. Testování většího počtu podobných vzorků probíhá v rámci jedné metody, a to pomocí správce kontextu, jenž je vytvořen metodou `subTest`.

Testované vzorky jednotkových testů a očekávané výsledky jsou fixně zaznamenány ve zdrojovém kódu, žádný z nich není načítán ze souboru. K porovnání jsou použity metody začínající slovem `assert`, např. `assertEqual`, `assertLessEqual` a `assertRaises`.

Jednotkové testy byly vytvořeny pro třídy vyskytující se v modulech `misc` a `modules` a pro každou třídu, která identifikuje nebo analyzuje typ skriptu[1]. Pro každý testovací případ existuje alespoň 30 testů.

7.1.2 Integrační testování

Skripty s integračními testy jsou pojmenovány `test_integr_*.py` a nacházejí se v balíčku `test_integr/`.

K vytvoření integračních testů byly využity stejné prostředky jako u jednotkových testů – standardní modul `unittest`, její třídy a metody. Integrační testy se od jednotkových liší ve vstupu. Zatímco vstupem jednotkových je pouze krátký řetězec zdrojového kódu, integračním testům je předán zdrojový kód načtený ze souboru.

Soubory uchovávající zdrojové kódy se nacházejí v balíčku `samples/`. Ten je rozdělen na další dva balíčky, z nichž první obsahuje soubory pro analýzu (`analyze/`) a druhý soubory pro identifikaci (`identify/`).

O načtení a dekódování zdrojových kódů a provedení integračních testů se stará instance třídy `TestUtils` z modulu `test_utils.py`. Metoda `_load_test_files` získá seznam všech souborů z požadovaného adresáře (parametr `path`), dekóduje je stejným způsobem jako modul pro načtení vstupních dat a dekódovaný zdrojový kód uloží do slovníku (klíčem je název souboru). Výsledný slovník je předán metodě `perform_integration_tests`, jež nad všemi vzorky ze slovníku provede zvolený test a výsledek i s názvem souboru vrátí jako generátor.

Načtení souborů i provedení testů se liší na základě druhu testu. Druh lze určit parametrem `mode` u obou metod. Hodnota `"identification"` spustí testování identifikace, zatímco hodnota `"analysis"` testování analýzy.

Během testování analýzy jsou načteny dva stejně pojmenované soubory: první s příponou `.in`, jenž obsahuje vstup, a druhý s příponou `.out`, jenž obsahuje referenční výstup. Výsledkem testu analýzy je slovník s extrahovanými vlastnostmi a referenční výstup představuje slovník s očekávanými vlastnostmi. Test je považován za úspěšný, pokud je druhý jmenovaný slovník podmnožinou prvního.

Proces testování identifikace je jednodušší – je načten pouze jeden soubor se vstupem (bez přípony). Výsledkem identifikace je důvěryhodnost v rozsahu 0 až 100, ta je porovnána s prahem daného typu skriptu a pokud je větší nebo rovna tomuto prahu, je test prohlášen za úspěšný.

[1] Třídy `*Identifier` a `*Analyzer`, kde hvězdička zastupuje název typu.

7.1.3 Testování zdrojového kódu

Během integračního a jednotkového testování je navíc zaznamenáváno pokrytí zdrojového kódu. K tomuto účelu je využit modul `coverage`, jenž je volán při spuštění testů pomocí `nose2` s parametrem `--with-coverage`. Výsledky jsou uloženy v souboru `.coverage`, ze kterého následně lze vytvořit přehled o pokrytí jednotlivých modulů (například ve formě HTML).

Pokrytí zdrojového kódu analyzátoru činí 97 %. Moduly se třídami pro identifikaci a analýzu dosahují hodnot v rozmezí 96 až 100 %. Z 1 228 příkazů není pokryto pouhých 35; mezi nepokryté příkazy patří výjimky, jež jsou volány pouze za zvláštních okolností (např. chybějící třída pro identifikaci), a zpracování argumentů zadaných při spuštění. Z výsledku byly vyjmuty skripty, jež nejsou přímou součástí analyzátoru (tj. `setup.py`, testovací skripty apod.). Vyjmuté soubory a adresáře jsou uvedeny v konfiguračním souboru `setup.cfg`.

Modul `flake8` poskytuje prostředky pro kontrolu zdrojového kódu z hlediska správného formátování. Zásady správného formátování, kterými se modul řídí, jsou inspirovány nebo součástí PEP 8[2]. Zdrojový kód je modulem testován za účelem jeho udržitelnosti a konzistence. Konfigurační soubor `setup.cfg` upravuje chování testu; díky němu lze z vyhledávání vyjmout složky, které neobsahují skripty se zdrojovými kódy, a zvýšit maximální délku řádku ze 79 na 120.

K odhalení potencionálních chyb při nesprávném zacházení s datovými typy je použit modul `mypy`. Jak autor uvádí v [10], modul provádí statickou kontrolu datových typů na základě jejich anotace u funkcí, metod, proměnných či atributů (tzv. *type hints*). Anotace jsou přítomny ve všech zdrojových kódech, je proto možné kódy pomocí `mypy` testovat.

7.2 Dosažené výsledky

Ke zjištění úspěšnosti identifikace a analýzy byl analyzátor spuštěn pro necelých 2 500 vzorků. Na každý typ skriptu připadá 200 až 600 vzorků[3].

Vzorky k typům JavaScript, VBScript a dávkový soubor byly poskytnuty firmou Avast Software, zatímco vzorky pro typy PowerShell a Python byly manuálně staženy z různých webových stránek a veřejných databází[4]. Z tohoto důvodu je prvních zmiňovaných vzorků více a vyjma ojedinělých případů se jedná o skutečný škodlivý software, jenž může být záměrně psán tak, aby jeho rozpoznání bylo co nejnáročnější.

Soubor všech vzorků byl tvořen tak, aby v něm byly zastoupeny vzorky různých velikostí, kódování a zakončení řádku. Průměrná velikost nejmenších vzorků je několik stovek bajtů, velikost největších se odvíjí od typu skriptu – skripty psané v jazycích JavaScript a VBScript dosahují velikosti až 10 MB, zatímco u ostatních typů mají největší vzorky stovky kilobajtů, maximálně 1 MB.

7.2.1 Výsledky identifikace

K ověření úspěšnosti identifikace bylo využito celkem 2 491 vzorků. Každý typ skriptu byl testován zvlášť a na vlastním souboru vzorků. Analyzátor běžel s následující konfigurací:

- velikost vstupního souboru nebyla omezena,

[2] Python Enhancement Proposal 8, podrobnosti viz https://www.python.org/dev/peps/pep-0008/
[3] Konkrétní čísla jsou uvedena u jednotlivých typů níže.
[4] Jedná se například o webové stránky https://pypi.org/ či https://gallery.technet.microsoft.com/

- identifikační proces skončil u prvního rozpoznaného typu skriptu,
- výstupním formátem byl JSON a
- ke spuštění analyzátoru a identifikaci typu Python byly použity interprety verze 3.7.2 a 2.7.16.

V odstavcích níže jsou podrobně rozepsány výsledky identifikace jednotlivých typů. Souhrn všech výsledků je uveden v tabulce 7.1.

	Počet vz.	**Detek. vz.**	**Úspěšnost**	**Doba identif. [min]**
JavaScript	618	616	99,68 %	22,6
VBScript	604	601	99,5 %	2,05
PowerShell	216	212	98,15 %	0,72
Python	402	399	99,25 %	1,77
Dávkový soubor	601	572	95,17 %	0,83
Žádný typ	50	50	100 %	5,75

Tabulka 7.1: Tabulka shrnující úspěšnost identifikace pro jednotlivé typy skriptů

Pro typ JavaScript bylo shromážděno 618 vzorků; vzorky zahrnují samostatné skripty i skripty, které jsou součástí HTML. Ze souboru bylo správně rozpoznáno 616 vzorků. Zbývající dva vzorky nebyly rozpoznány, protože skripty byly součástí HTML a byly prázdné. Identifikace zabrala 22,6 minut, průměrně 2,19 sekund na vzorek.

Soubor pro typ VBScript obsahuje celkem 604 vzorků (samostatných skriptů i skriptů vnořených do HTML), z toho bylo identifikováno 601. Zbylé tři vzorky buď nebylo možné rozpoznat kvůli nedostatku detekovaných vzorů, nebo se jednalo o odnože jazyka Visual Basic, které nejsou podporovány. Čas identifikace je 2,05 minut a průměrný čas 0,2 sekund.

Celkový počet vzorků pro typ PowerShell je 216. Ze všech těchto vzorků bylo rozpoznáno 212 vzorků. Čtyři vzorky byly příliš malé (velikost do 1 kB), než aby je bylo možné rozpoznat pomocí definovaných pravidel. Rozpoznávání trvalo celkem 43,36 sekund a v průměru 0,2 sekund.

Úspěšnost identifikace skriptů psaných v jazyce Python byla otestována na 402 vzorcích, z toho 107 vzorků je kompatibilní s verzí 2.x a zbylých 295 s verzí 3.x. Celkem tři vzorky nebyly správně identifikovány. První z nich obsahoval víceřádkové řetězce obsahující strom HTML s elementem `script`, a tudíž byl zpracován a vyhodnocen jako kód HTML obsahující skript typu JavaScript. Zbývající dva vzorky obsahovaly jediný řádek s příkazem, jenž lze stejným způsobem zapsat i v jazyce JavaScript, a proto byl typ určen jako JavaScript. Identifikace všech vzorků běžela 1,77 minut a v průměru trvala 0,38 (Python 2.x) a 0,22 (Python 3.x) sekund.

Poslední podporovaný typ – dávkový soubor – byl testován na 601 vzorcích. Celkem 572 vzorků bylo správně označeno za dávkové soubory, u zbývajících 29 vzorků buď nebyl rozpoznán žádný typ, nebo byl rozpoznán nesprávně. Čtyři z těchto vzorků nebyly dávkové soubory, nýbrž Makefile a skripty SQL. Jeden byl vyhodnocen jako typ VBScript, a to proto, že obsahoval kromě příkazů i kód jazyka VBScript. Typ VBScript má větší prioritu než dávkový soubor, identifikace tudíž skončila u něj. Zbylé vzorky byly příliš krátké (do 10 řádků kódu) a neobsahovaly dostatečné množství konstrukcí, díky nimž lze typ určit. Identifikace skončila po 49,58 sekundách, jeden vzorek byl identifikován v průměru za 0,08 sekund.

K ověření falešně pozitivních identifikací bylo vybráno 50 vzorků, jež neodpovídají žádnému z podporovaných typů. Vzorky zahrnují skripty jiných jazyků (např. SQL), logy, hlavičky e-mailových zpráv, běžný text aj. Ani u jednoho z nich nedošlo k určení typu. Vzhledem k tomu, že bylo nutné projít identifikací všech pěti typů, je celková doba identifikace 5,75 minut, průměrně 6,9 sekund na vzorek.

7.2.2 Výsledky analýzy

Během ověřování úspěšnosti identifikace byly ukládány i výsledky analýzy. U žádného z provedených spuštění analyzátoru nedošlo k vyvolání výjimky či jinému stavu, ze kterého by se analyzátor nemohl zotavit. Vzhledem k rozsahu a množství testovacích vzorků nelze efektivně vyhodnotit úspěšnost analýzy, proto byla místo toho zvolena metoda náhodného výběru a ručního ověření.

Z každého souboru vzorků bylo náhodně vybráno patnáct výsledků analýz a ty byly manuálně porovnány se zdrojovými kódy vzorků. Byly vybírány takové vzorky, u nichž bylo manuální porovnání reálné, tedy vzorky menší velikosti (desítky kilobajtů). Výsledky analýz u všech pěti typů odpovídaly vlastnostem uvedeným v příslušných zdrojových kódech.

Kapitola 8

Závěr

Cílem této práce bylo popsat rozšiřitelný analyzátor skriptů, jehož výstupem je kolekce vlastností extrahovaných ze zdrojového kódu. Získané vlastnosti budou použity při shlukové analýze vzorků ve firmě Avast Software.

Analyzátor provádí dvě hlavní operace: identifikaci a analýzu. Identifikace umí rozpoznat celkem pět typů skriptů – JavaScript, VBScript, PowerShell, Python a dávkový soubor – a je zodpovědná za výběr konkrétní třídy, jež provede analýzu. Analýza extrahuje vlastnosti definované daným typem skriptu a výslednou kolekci buď převádí do formátu JSON, nebo ukládá do třídy, která poskytuje metody pro manipulaci s vlastnostmi. Analyzátor je tedy možné spustit jako samostatný skript, nebo importovat jako modul.

Funkčnost analyzátoru byla otestována pomocí jednotkových a integračních testů. Výsledkem testů je také procentuální pokrytí kódu, které dosáhlo 97 %. Úspěšnost identifikace a analýzy byla ověřena na 2 491 vzorcích. Každý typ skriptu byl ověřován zvlášť. Správně bylo identifikováno 2 450 vzorků, úspěšnost je tedy 98,35 %. Sedmdesát pět náhodně vybraných a správně identifikovaných vzorků bylo podrobeno manuálnímu ověření správnosti extrahovaných vlastností. U všech vybraných vzorků proběhla analýza správně.

Ačkoliv analyzátor nyní podporuje pouze pět typů skriptů, při vývoji byl navržen tak, že jej lze rozšířit o další typy. Pro přidání nového typu je pouze potřeba implementovat třídy vykonávající identifikaci a analýzu, žádné další úpravy nejsou nutné.

Literatura

[1] *Windows PowerShell Language Specification*. Microsoft Corporation, 2009, [Online; navštíveno 02.03.2019]. Dostupné z: https://www.microsoft.com/en-us/download/details.aspx?id=36389.

[2] *Enabling VBScript in Browsers*. Tutorials Point, 2019, [Online; navštíveno 06.05.2019]. Dostupné z: https://www.tutorialspoint.com/vbscript/vbscript_enabling.htm.

[3] *The Python Language Reference*. Python Software Foundation, 2019, [Online; navštíveno 03.04.2019]. Dostupné z: https://docs.python.org/3/reference/.

[4] Everitt, B.; Landau, S.; Leese, M.; aj.: *Cluster Analysis*. Chichester: Wiley, páté vydání, 2011, ISBN 978-0-470-74991-3.

[5] Flanagan, D.: *JavaScript: the definitive guide*. Sebastopol (California): O'Reilly, 6 vydání, 2011, ISBN 978-0-596-80552-4.

[6] Grossman, J.; Hansen, R.; Petkov, P. D.; aj.: *XSS attacks: cross-site scripting exploits and defense*. Burlington (Massachuttes): Syngress Publishing, 2007, ISBN 1-59749-154-3.

[7] Hidayat, A.: *Esprima — Esprima Master Documentation*. Read the Docs, Inc & contributors, 2019, [Online; navštíveno 07.04.2019]. Dostupné z: https://esprima.readthedocs.io/en/latest/.

[8] Holmes, L.: *Windows PowerShell cookbook*. Beijing: O'Reilly, třetí vydání, 2013, ISBN 978-144-9320-683.

[9] Jerry Lee Ford, J.: *Microsoft Windows Shell Script Programming for the Absolute Beginner*. Boston (Massachusetts): Premier Press, 2004, ISBN 1-59200-085-1.

[10] Lehtosalo, J.: *Mypy 0.701 documentation*. Read the Docs, Inc & contributors, 2019, [Online; navštíveno 23.04.2019]. Dostupné z: https://mypy.readthedocs.io/en/stable/index.html.

[11] Lutz, M.: *Learning Python*. Sebastopol (California): O'Reilly, páté vydání, 2013, ISBN 978-1-449-35573-9.

[12] Mohanta, A.; Hahad, M.; Velmurugan, K.: *Preventing Ransomware: Understand, prevent, and remediate ransomware attacks*. Birmingham: Packt Publishing, 2018, ISBN 978-1-78862-060-4.

[13] Mouzarani, M.; Sadeghiyan, B.; Zolfaghari, M.: Detecting injection vulnerabilities in executable codes with concolic execution. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, ISSN 2327-0594, doi:10.1109/ICSESS.2017.8342862.

[14] O'Connor, T.: *Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*. Waltham (Massachusetts): Syngress, 2013, ISBN 978-1-59749-957-6.

[15] Pilgrim, M.; Blanchard, D.; Cordasco, I.: *Chardet — chardet 3.0.4 documentation*. Read the Docs, Inc & contributors, 2015, [Online; navštíveno 09.05.2019]. Dostupné z: https://chardet.readthedocs.io/en/latest/.

[16] Stanek, W. R.; O'Neill, J.; Rosen, J.: *Microsoft Powershell, VBScript, and JScript bible*. Indianapolis: Wiley Publishing, 2009, ISBN 978-0-470-38680-4.

[17] Wilson, E.: *Microsoft VBScript: step by step*. Redmont (Washington): Microsoft Press, 2007, ISBN 07-356-2297-3.

Příloha A

Kompletní struktura balíčku

V této příloze je uvedena kompletní struktura balíčku nejvyšší úrovně pojmenovaného `script_analysis`.

- `analyze/` – balíček souborů s třídami pro analýzu typů skriptů
- `identify/` – balíček souborů s třídami pro identifikaci typů skriptů
- `__init__.py` – inicializační soubor balíčku
- `abstracts.py` – modul s abstraktními třídami
- `data.py` – modul s daty používanými při analýze
- `errors.py` – modul s třídami výjimek
- `misc.py` – modul obsahující třídy pro různé účely
- `modules.py` – modul s třídami modulů analyzátoru
- `script_analyzer.py` – modul s třídou analyzátoru

Příloha B

Ukázka výstupu ve formátu JSON

Tato příloha obsahuje nezkrácenou verzi ukázky výstupu analyzátoru převedeného do formátu JSON.


```
{
    "javascript": {
        "block_comments": [],
        "block_comments_count": 0,
        "confidence": 100,
        "encoding": "utf-8",
        "end_chars": "t.body.appendChild(bootstrap);",
        "in_html": false,
        "keywords": {
            "var": 1
        },
        "line_comments": [
            " chrome extension background bootstrap"
        ],
        "line_comments_count": 1,
        "line_count": 6,
        "longest_line": 61,
        "start_chars": "// chrome extension background",
        "strings": [
            "script",
            "text/javascript",
            "https://apicytiwebnet-a.akamaihd.net/gcbs2"
        ],
        "strings_count": 3
    }
}
```

Příloha C

Návod k použití analyzátoru

Analyzátor je navržen tak, aby mohl být spuštěn jako samostatný skript, jemuž jsou předány argumenty, ale také je možné jej importovat z modulu, předat mu argumenty přes vstupní parametry a využít jej v jiných modulech či skriptech.

Skript `sa.py`, jenž je přiložen k balíčku s analyzátorem, je ukázkou prvního způsobu použití a obsahuje následující kód:

```
from script_analysis.script_analyzer import ScriptAnalyzer

sa = ScriptAnalyzer()

if __name__ == "__main__":
    print(sa.perform_script_analysis())
```

Má-li být analyzátor spuštěn jako samostatný skript, nejsou metodě, jež provádí identifikaci a následně analýzu, předány žádné argumenty. Skript lze pak spustit například takto:

```
python3 sa.py samples/analyzed_script --max-file-size=1000000
```

Analyzátor provede analýzu vstupního souboru `analyzed_script`, navíc je explicitně uvedeno, že jeho velikost může být maximálně 1 milion bajtů. Zbylé argumenty (režim identifikačního procesu, výstupní formát atd.) si ponechají výchozí hodnoty.

V případě, že má být analyzátor součástí jiného skriptu či modulu psaného v jazyce Python, lze jej importovat a použít následovně:

```
from script_analysis.script_analyzer import ScriptAnalyzer

sa = ScriptAnalyzer()

traits = \
    sa.perform_script_analysis(False,
                               src_file="samples/analyzed_script",
                               max_file_size=1000000,
                               unlimited_size=False,
                               identification_process="first-match",
                               start_end_chars=30,
                               output_format="json",
                               python2_interpreter="python2")
```

Konfigurace analyzátoru je identická s tou z prvního příkladu. Metodě musí být při volání předány povinně všechny argumenty analyzátoru, navíc musí první, poziční parametr nabývat hodnoty `False`, v opačném případě by se analyzátor pokusil o zpracování argumentů zadávaných při spuštění z konzole, což by vedlo k nedefinovanému chování.

Příloha D

Obsah přiloženého DVD

Přiložené DVD obsahuje následující soubory a adresáře:

- `latex/` – zdrojové soubory této práce
- `script-analyzer/` – zdrojové soubory analyzátoru
- `xplani02_bp.pdf` – elektronická podoba této práce