Techniki wykrywania złośliwych stron WWW

Łukasz Juszczyk
CERT Polska/NASK
lukasz.juszczyk@cert.pl

25 marca 2010

Wstęp

Wraz ze wzrostem zainfekowanych stron internetowych pojawiają się również narzędzia służące do identyfikacji oraz analizy złośliwego kodu. Potrafią one wydobywać kod (głównie JavaScript, ale także VBScript) z dokumentów HTML i PDF oraz z animacji Flash. Następnie, w badanym kodzie, wyszukiwane są exploity. Mogą to być standardowe przepełnienia bufora jak i pływające ramki pobierające szkodliwy program, czy automatyczne przekierowania do złośliwych stron. Do takich narzędzi należą min. PhoneyC, jsunpack oraz wepawet.

Wymagania

Narzędzia, które mają wykrywać złośliwe strony internetowe, powinny spełniać pewne założenia. Przede wszystkim *malware*, występujący w Internecie, wykorzystuje zazwyczaj JavaScript (dokumenty HTML, ale też dokumenty PDF i animacje Flash). Jednakże sam interpreter języka nie wystarczy do pełnej analizy skryptów na stronach WWW. Każda przeglądarka internetowa posiada obiekty JavaScript (takie jak `document` lub `window`), które nie są częścią języka. Obiekty te pozwalają na dostęp do obiektowego modelu dokumentu HTML (ang. *Document Object Model, DOM* [10]) oraz funkcji typowych dla przeglądarek.

Ponadto, znaczna ilość szkodliwych skryptów jest zaciemniona (ang. *obfuscated*) lub spakowana. W związku z czym, narzędzie analizujące taki kod musi radzić sobie z takimi utrudnieniami.

1 Opis narzędzi

Prezentowane narzędzia posiadają podobne możliwości, jednak każde z nich różni się od pozostałych sposobem działania. Poza tym, do realizacji założonych celów, wykorzystują różne mechanizmy detekcji szkodliwego kodu.

1.1 PhoneyC

Pierwszy z nich, PhoneyC [8], jest typowym nisko interaktywnym klienckim honeypotem. Składa się z dwóch, współpracujących ze sobą, części: *crawlera* oraz analizatora HTML, JavaScript i VBScript. PhoneyC napisany jest w pythonie, jednak do parsowania skryptów korzysta z zewnętrznych programów. Do interpretowania JavaScriptu wykorzystuje silnik SpiderMonekey [7], natomiast skrypty VBS konwertowane są do pythona za pomocą vb2Py. Przed rozpoczęciem analizy tworzone jest środowisko przeglądarki, natomiast samo parsowanie skryptów odbywa się w zewnętrznym, odizolowanym, procesie SpiderMonkey.

Jego podstawową funkcją jest udawanie podatności występujących zarówno w silniku przeglądarki, jak i w kontrolkach ActiveX. Oprócz tego, PhoneyC, wykorzystuje ClamAV [4] do wykrywania wirusów w analizowanych plikach. Dodatkiem jest wizualna reprezentacja odwiedzonych stron na grafie, z zaznaczeniem, które witryny zostały sklasyfikowane jako szkodliwe.

Za pomocą tzw. *vulnerability modules*, PhoneyC, pozwala na dynamiczną analizę zawartości stron WWW pod kątem występowania w nich złośliwego kodu. Identyfikacja zagrożeń realizowana jest poprzez dopasowanie obiektów JavaScript lub VB Script do zbioru reguł definiujących znane exploity. Przykładowa reguła została przedstawiona na listingu 1.

Listing 1: Przykładowa reguła PhoneyC

```
1  // BitDefender Online Scanner ActiveX Control
2  // CVE-2007-5775
3  function BitDefender() {
4    this.initx=function(arg) {
5      if (arg.length > 1024) {
6        add_alert('BitDefender Online Scanner InitX() overflow');
7      }
8    }
9    this.InitX=this.initx;
10 }
```

1.2 jsunpack

Drugim, omówionym, narzędziem jest jsunpack, przez autora określane jako *Generic JavaScript Unpacker*, czyli „rozpakowywacz" JavaScript ogólnego przeznaczenia [3]. Jsunpack dostępny jest w formie serwisu internetowego oraz programu (`jsunpack-n`) napisanego w pythonie.

Jak sama nazwa wskazuje, jsunpack potrafi wydobywać kod JavaScript z plików w innym formacie. W bieżącej wersji programu są to dokumenty HTML, PDF oraz animacje Flash. Narzędzie radzi sobie z zaciemnionymi skryptami, potrafi także analizować skompresowane dokumenty PDF. Ponadto, jako dane wejściowe, jsunpack może wykorzystać interfejs sieciowy lub wcześniej utworzony plik `pcap`. Program potrafi także sam pobierać strony internetowe na podstawie dostarczonego adresu URL lub znalezionych odnośników.

W procesie parsowania kodu JavaScript, podobnie jak w przypadku PhoneyC, tworzone jest środowisko imitujące przeglądarkę internetową. W tym celu wykorzystane są dwa skrypty: `pre.js` oraz `post.js`. Ich zadaniem, odpowiednio, jest przygotowanie zestawu funkcji specyficznych dla przeglądarki oraz wyszukiwanie w zdefiniowanych zmiennych (utworzonych podczas wykonywania skryptu) *shellcode*'ów lub adresów URL.

Dynamiczne wykrywanie zagrożeń jest lepsze, aczkolwiek nie zawsze skuteczne, ponieważ może zawierać błędy. Z tego powodu jsunpack oprócz reguł JavaScript implementuje także reguły statyczne w formacie projektu YARA [1]. Razem z programem dostarczane jest 26 reguł. Przykładowa reguła przedstawiona jest na listingu 2.

Listing 2: Przykładowa reguła jsunpack

```
1  rule MSsetSlice
2  {
3    meta:
4      ref = "CVE-2006-3730"
5      impact = 7
6    strings:
7      $cve20063730_1 = "setSlice" nocase fullword
8      $cve20063730_2 = "WebViewFolderIcon.WebViewFolderIcon.1" nocase fullword
9    condition:
10     1 of them
11 }
```

W czasie analizy stron internetowych badane jest wiele plików zawierających dokumenty HTML, animacje Flash, pliki wykonywalne. Te pliki (domyślnie jedynie zawierające szkodliwy kod) zapisywane są na dysku z odpowiednim prefiksem. Dzięki czemu można poddać je dalszej analizie z

wykorzystaniem narzędzi trzecich. Dodatkowo wszystkie operacje wykonywane przez jsunpack zapisywane są w pliku dziennika o nazwie `decoded.log`. Można je także przedstawić na grafie.

1.3 wepawet

Ostatnim prezentowanym narzędziem jest wepawet (*Web Engine to Protect from and Analyze Widespread and Emerging Threats*) [2]. Wepawet, w odróżnieniu do dwóch pozostałych, dostępny jest jedynie przez przeglądarkę jako serwis internetowy pod adresem `http://wepawet.iseclab.org/`.

Serwis jest zbiorem narzędzi do statycznej oraz dynamicznej analizy zawartości sieci w celu zidentyfikowania szkodliwego zachowania. Na dzień dzisiejszy do wspieranych formatów należą JavaScript, Flash oraz PDF. Na podstawie analizy, badany plik klasyfikowany jest jako niegroźny (ang. *benign*, podejrzany (ang. *suspicious*) lub szkodliwy (ang. *malicious*).


This is a summary of what was observed on **hentairulz.com**.

Network Information

IP	ASN	Country
9[redacted]9	44571	VG

(Click on any AS number to see its FIRE report)

Other domains on this IP:

- [redacted]ivda.com
- [redacted]ka.com
- [redacted]a.com
- [redacted]com

Registration Information

No information available at this time.

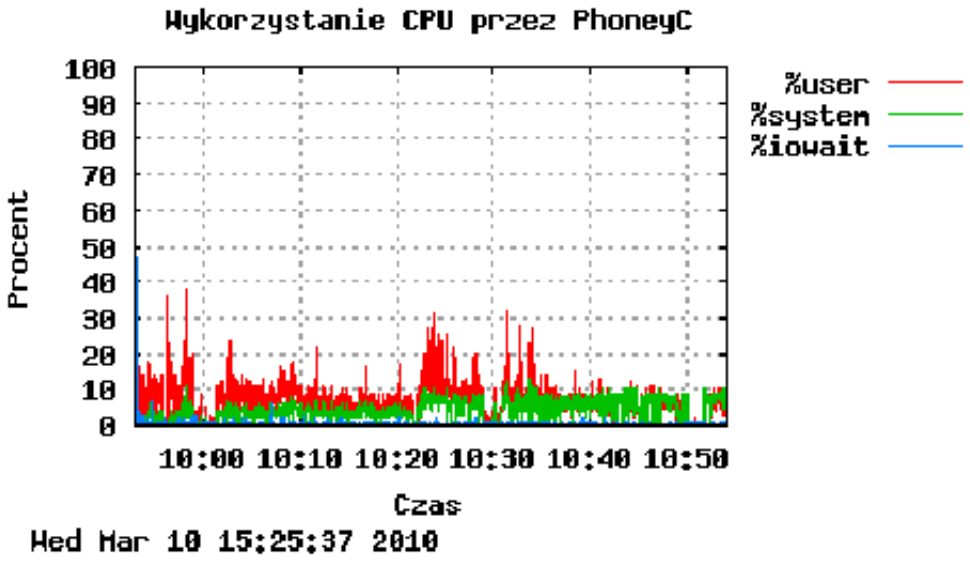
Rysunek 1: wepawet: Informacje o domenie

W przypadku określenia pliku jako szkodliwy wepawet dostarcza opisy wykrytych exploitów oraz odnośniki do CVE i Secunii. W raporcie przedstawione są także informacje dotyczące AS-ów, adresów IP oraz nazw do-

menowych zawierających *malware* (Rys. 1). Jeżeli znaleziony zostanie plik wykonywalny możliwe jest bezpośrednie poddanie go analizie zewnętrznej w serwisach Anubis lub Virus Total.

W plikach PDF wyszukiwany jest szkodliwy kod JavaScript. Z kolei sam JavaScript sprawdzany jest głównie pod kątem wykorzystywania ataków *drive-by downloads*. Wepawet nie wykrywa fałszywego oprogramowania: systemów antywirusowych, czy kodeków audio/wideo.

2 Testy

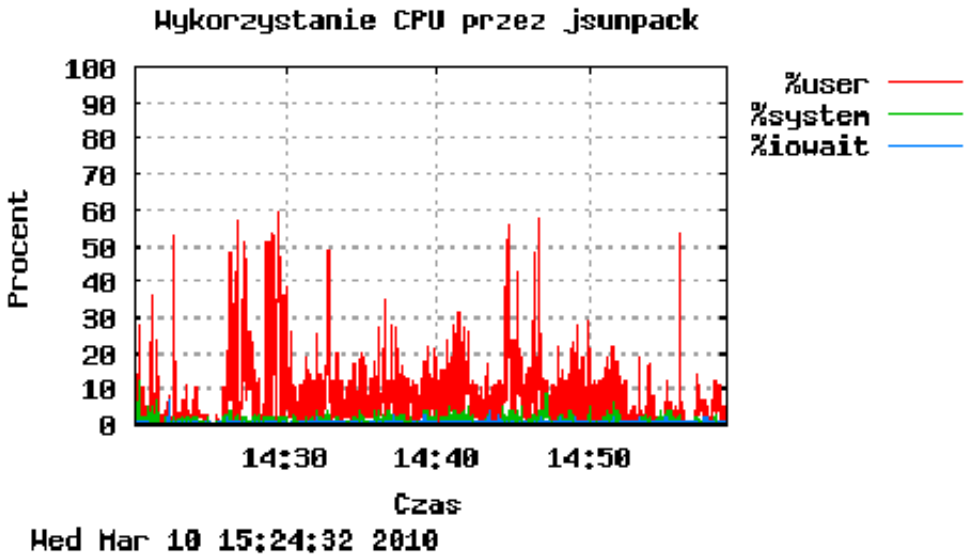


Rysunek 2: Wykorzystanie CPU przez PhoneyC

Podczas pracy narzędzi zostało zmierzone obciążenie systemu. Test został przeprowadzony jedynie dla PhoneyC oraz jsunpack'a. Ze względu na fakt, iż wepawet jest serwisem internetowym, nie ma możliwości uruchomienia go na własnym sprzęcie. Konfiguracja maszyny testującej jest następująca:

```
Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz
2048 MB DDR2 RAM @ 800 MHz
Linux 2.6.32, Python 2.6.4, JavaScript-C 1.7.0
```

Na wykresie (Rys. 2) można zaobserwować, iż PhoneyC podczas pracy generuje obciążenie na poziomie 15%. Trochę bardziej „zasobożerny” jest jsunpack, jednak średnie zużycie procesora nie przekracza 20% (Rys. 3). W obu przypadkach wąskim gardłem jest pobieranie danych z Internetu, a nie sam proces analizy.



Rysunek 3: Wykorzystanie CPU przez jsunpack

W drugim teście została zbadana szybkość analizy. Nie da się bezpośrednio porównać ze sobą tych narzędzi, ponieważ działają one na innych zasadach. Dlatego dla każdego z nich został przeprowadzony inny test, a ich wyniki należy traktować jedynie jako wyobrażenie o szybkości danego programu.

Dla pierwszego z nich, PhoneyC, został zmierzony czas oraz ilość odwiedzonych adresów URL dla danej głębokości(ang. *depth*).[1]

depth:1 $\Rightarrow$ walked 144 URLs in 551.384185 seconds (ok.10m; 3,83s/URL)

depth:2 $\Rightarrow$ walked 344 URLs in 1163.629450 seconds (ok. 20m; 3,38s/URL)

depth:4 $\Rightarrow$ walked 2609 URLs in 9916.836634 seconds (ok. 2h45m; 3,80s/URL)

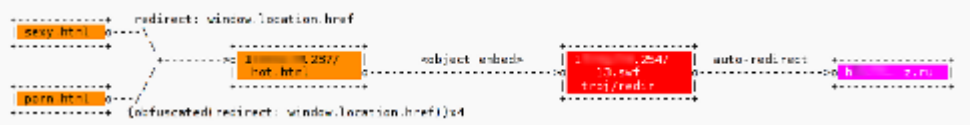
[1]Głębokość określa jak daleko (ilość „hopów”) honeypot podąża za linkami na stronie WWW.

W przypadku jsunpack'a został zmierzony czas przetwarzania kolejno stu dokumentów PDF oraz stu animacji Flash. Obydwa typy plików zostały pobrane z serwisu *Malware Domain List* [6]. Ponadto pliki zostały przeanalizowane w trybie aktywnym, który zapewnia odwiedzanie znalezionych łączy.

100 plików PDF $\Rightarrow$ 226 sekundy (2,26s/plik)

100 plików SWF $\Rightarrow$ 15 sekund (0,15s/plik)

2.1 Studium przypadku #1



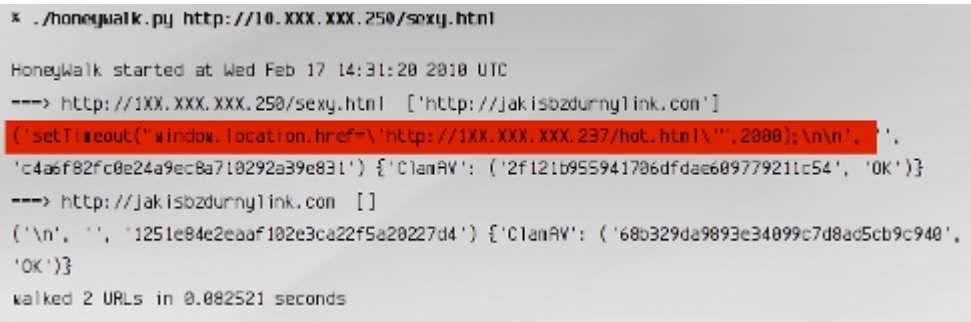
Rysunek 4: Środowisko testowe

Poza testami wydajnościowymi zostały przeprowadzone testy jakościowe opisywanych narzędzi. W tym celu wykorzystano wewnętrzne środowisko testowe opisane na rysunku 4 oraz zasoby *Malware Domain List*.

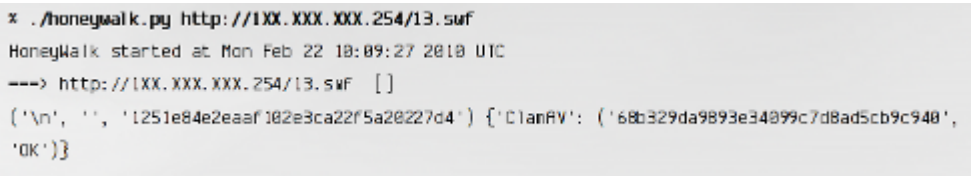
Użyty fragment środowiska testowego przekierowuje przeglądarkę internetową na dokument HTML zawierający animacje Flash, która z kolei automatyczne przekierowuje na szkodliwą stronę w Internecie. Celem tego testu jest sprawdzenie, czy testowane narzędzia radzą sobie z interpretacją zaciemnionego kodu JavaScript oraz potrafią podążać za linkami osadzonymi w skryptach oraz plikach SWF. Ponieważ środowisko dostępne jest jedynie lokalnie do wepawet przekazano jedynie pojedyncze pliki.

Uruchomienie PhoneyC (Rys. 5) pokazuje, że mimo tego, iż znalazł on łącze w JavaScripcie nie podążył on za nim. Osobne sprawdzenie pliku SWF nie wykazało żadnych anomalii (Rys. 6).

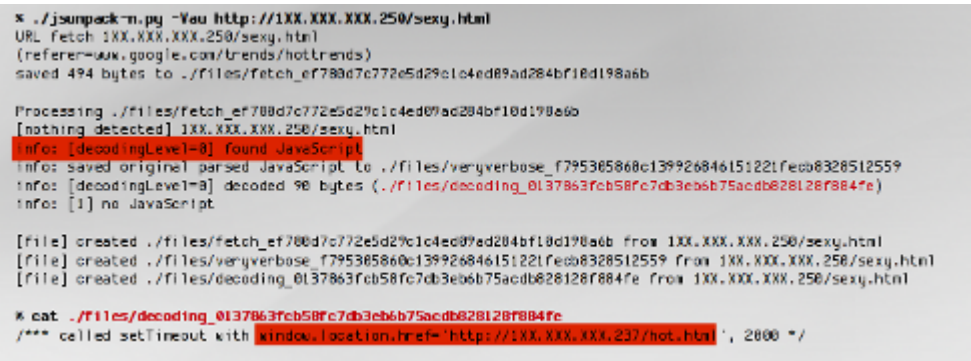
Jsunpack, w tym przypadku zachowuje się podobnie do PhoneyC: przekierowanie JavaScript zostaje znalezione, jednak (pomimo trybu aktywnego) narzędzie nie uznaje go jako łącze, które należy odwiedzić (Rys. 7). Bezpośrednia analiza animacji Flash wykazuje zawartość JavaScript (Rys. 7), ale w tym przypadku także nie następuje sprawdzenie, co kryje się pod znalezionym adresem. Podczas dodatkowej analizy pliku SWF (Rys. 7) można zobaczyć akcje odpowiedzialną za otwieranie adresów URL (`ActionGetURL2`), natomiast nie widać samego adresu.



Rysunek 5: Studium przypadku #1: PhoneyC

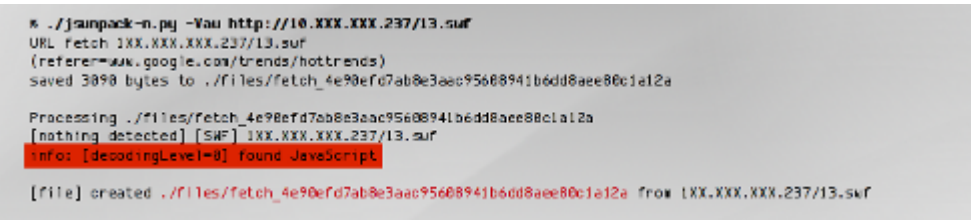


Rysunek 6: Studium przypadku #1: PhoneyC - animacja Flash

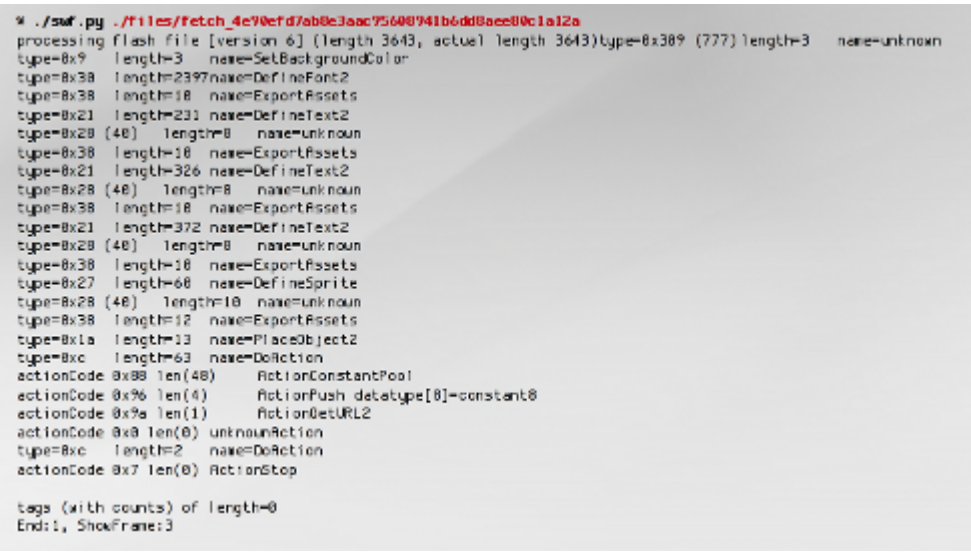


Rysunek 7: Studium przypadku #1: jsunpack

Do analizy w serwisie wepawet najpierw została przesłana animacja Flash (`MD5 (13.swf) = d019cab071ea284fcaaf7dda0d689500`). Można zaobserwować (Rys. 10), iż ze względu na automatyczne przekierowanie (*Automatically Redirects Browser*) została zaklasyfikowana jako szkodliwa. Warto także zauważyć, że podatność została wykryta podczas analizy dynamicznej (*Runtime*). Następnie, zbadano dokument HTML zawierający JavaScript (Rys. 11). Wepawet pokazuje krok po kroku jak plik został odczytany (*deob-*



Rysunek 8: Studium przypadku #1: jsunpack - animacja Flash

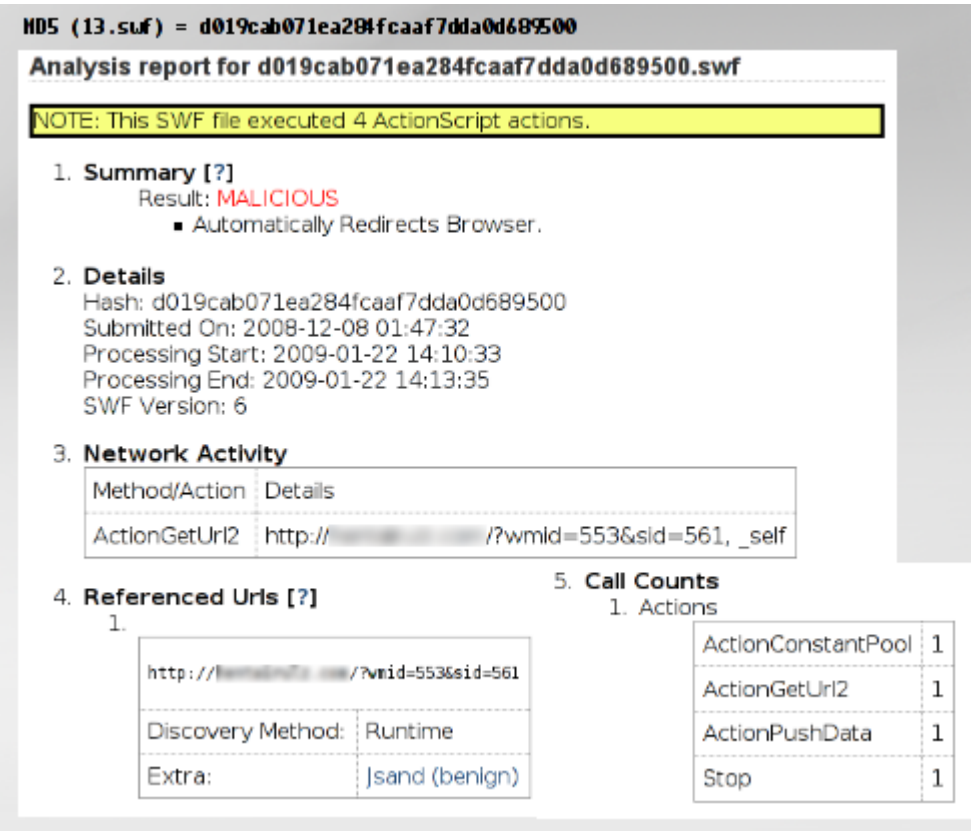


Rysunek 9: Studium przypadku #1: jsunpack - analiza SWF

fuscated) oraz czy jest on złośliwy.

2.2 Studium przypadku #2

W drugim studium przypadku pobrano (pseudo) losowy dokument PDF z *Malware Domain List*. Z uwagi na fakt, iż PhoneyC nie obsługuje PDF-ów sposób jego testowania był inny: wykonano serię żądań do adresów URL udostępnionych na MDL.

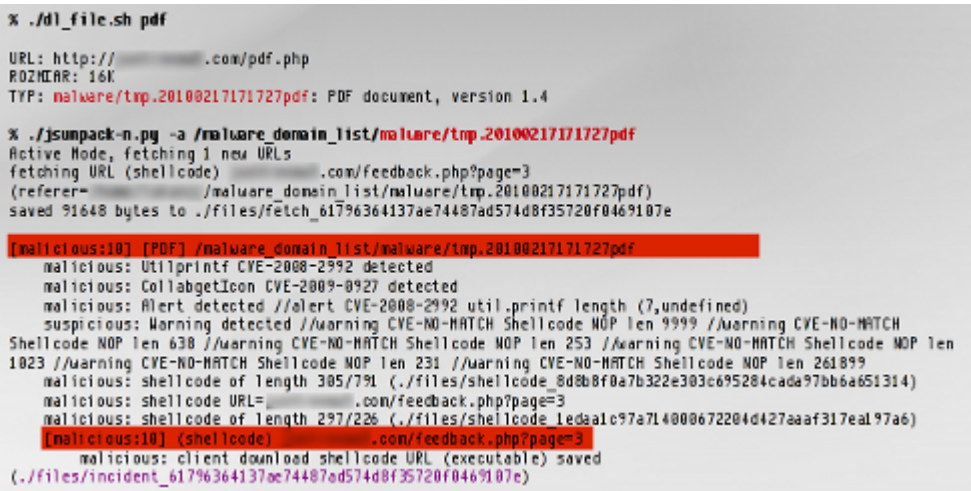


Rysunek 10: Studium przypadku #1: wepawet - analiza SWF

Rysunek 11: Studium przypadku #1: wepawet - analiza JavaScript

2.2.1 jsunpack

Analiza pobranego pliku PDF wykazała, iż jest on szkodliwy. Zostały rozpoznane trzy podatności:



Rysunek 12: Studium przypadku #2: jsunpack

CVE-2008-2992 Util.printf buffer overflow

CVE-2009-0927 Collab.getIcon buffer overflow

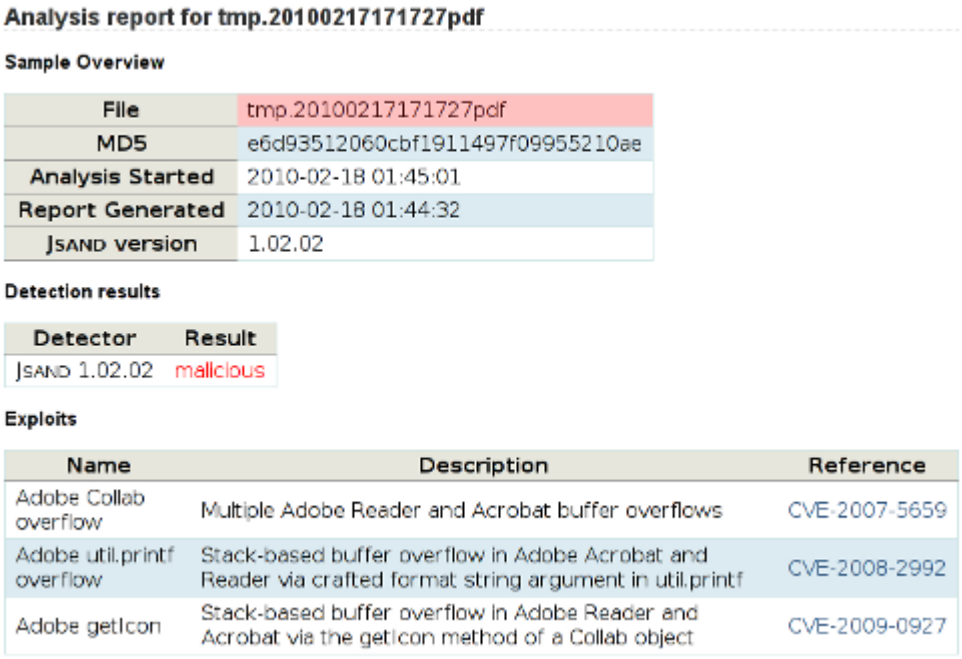
Shellcode NOP Kilka *shellcode*'ów

Każda z nich jest opisana, do dwóch z nich podane są numery z bazy CVE (ang. *Common Vulnerabilities and Exposures*). Ponadto plik został zapisany na dysku (`./files/incident_61796364137ae74487ad574d8f35720f0469107e`), co umożliwia późniejsze jego zbadanie, na przykład dedykowanymi narzędziami do PDF-ów.

2.2.2 wepawet

Ten sam dokument PDF został wysłany do serwisu wepawet. Tak samo jak w przypadku jsunpack'a został określony jako złośliwy. W sekcji ze znalezionymi exploitami znajdują się trzy pozycje, wraz z opisami:

- Adobe Collab overflow
- Adobe util.printf overflow
- Adobe getIcon



Analysis report for tmp.20100217171727pdf

Sample Overview

File	tmp.20100217171727pdf
MD5	e6d93512060cbf1911497f09955210ae
Analysis Started	2010-02-18 01:45:01
Report Generated	2010-02-18 01:44:32
JSAND version	1.02.02

Detection results

Detector	Result
JSAND 1.02.02	malicious

Exploits

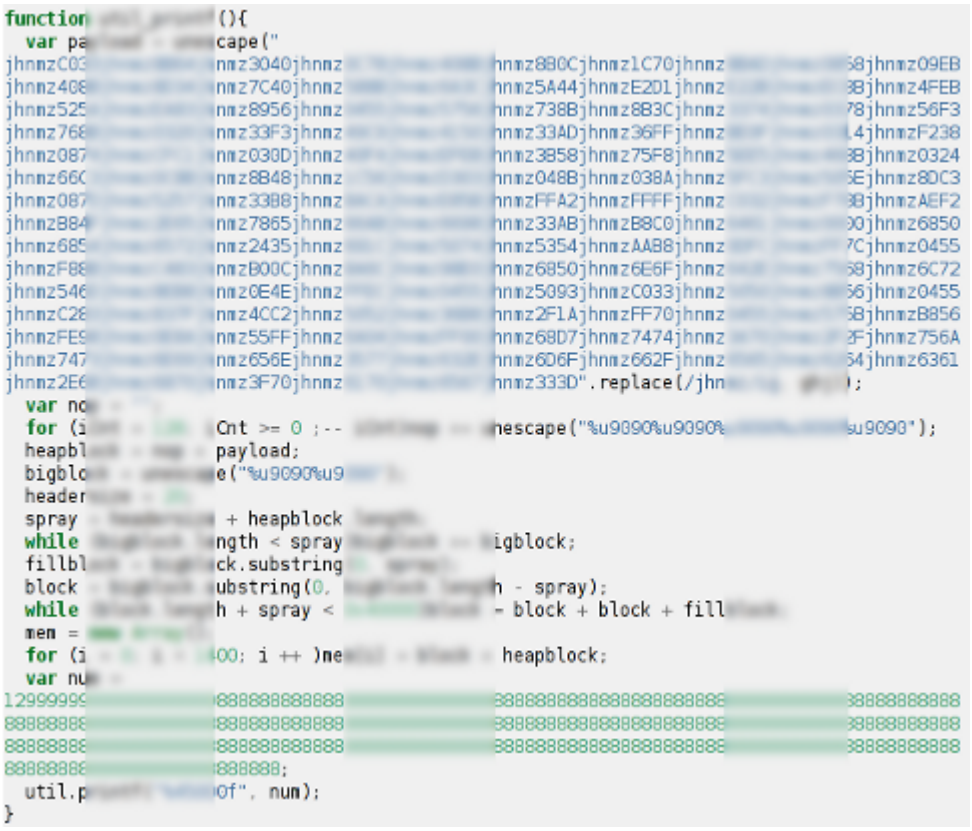
Name	Description	Reference
Adobe Collab overflow	Multiple Adobe Reader and Acrobat buffer overflows	CVE-2007-5659
Adobe util.printf overflow	Stack-based buffer overflow in Adobe Acrobat and Reader via crafted format string argument in util.printf	CVE-2008-2992
Adobe getIcon	Stack-based buffer overflow in Adobe Reader and Acrobat via the getIcon method of a Collab object	CVE-2009-0927

Rysunek 13: Studium przypadku #2: wepawet PDF

Wepawet oprócz wykazania podatności pokazuje dokładną analizę zawartości dokumentu (Rys. 14). Można zobaczyć kod JavaScript, proces jego *deobfuskacji* oraz występujące w nim *shellcody* w formacie szesnastkowym i ASCII (Rys. 15). Poniżej znajdują się także opisy potencjalnego malwaru, do którego linki znajdują się w kodzie, wraz z odnośnikami do zewnętrznej analizy w serwisach VirusTotal [9] oraz Anubis [5].

2.2.3 PhoneyC

Podczas testowania PhoneyC, wykonano szereg zapytań do stron z listy MDL (Rys. 15). Mimo, iż wiele z odpytanych adresów zostało zaklasyfikowane jako szkodliwe lub podejrzane przez inne narzędzia (Rys. 16), PhoneyC nie uznał żadnej z nich za złośliwą. Zdarzały się także przypadki, w który strona nie została przeanalizowana ze względu na błędy składniowe.

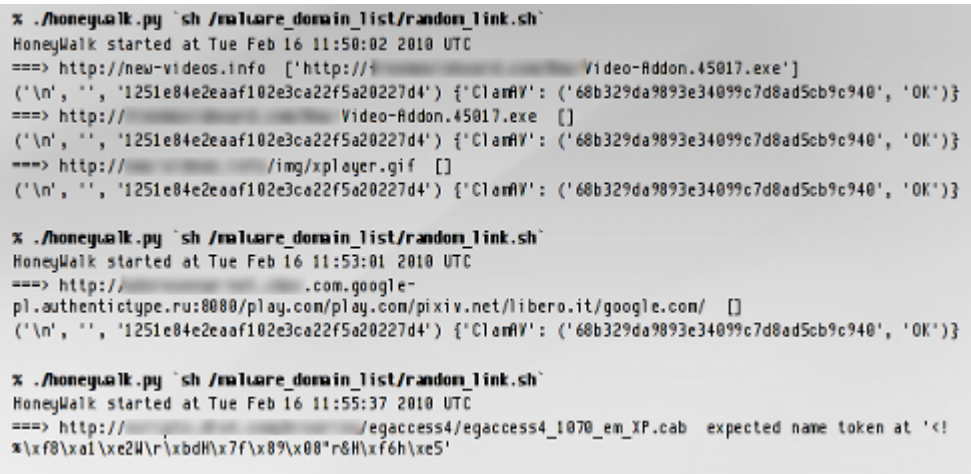


Rysunek 14: Studium przypadku #2: wepawet JavaScript

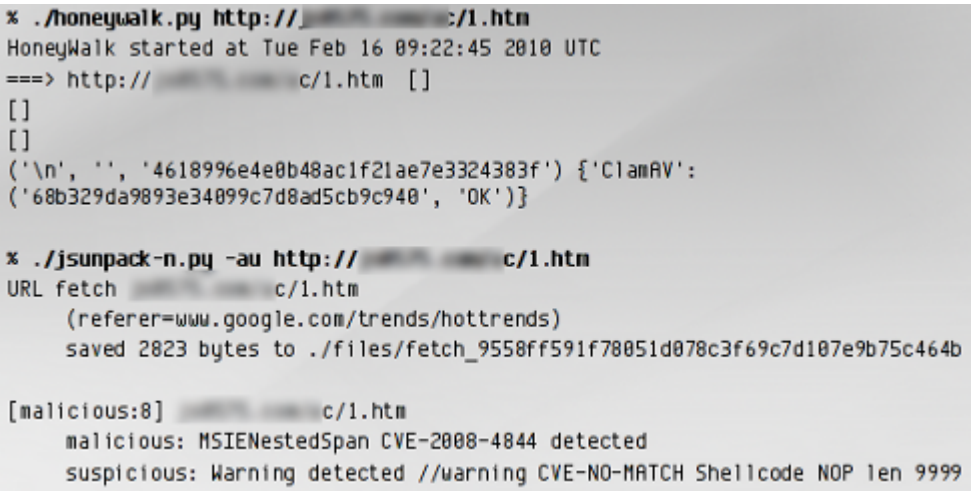
Podsumowanie

Po bliższym zapoznaniu się z narzędziami można powiedzieć, że żadne z nich nie spełnia stawianych im oczekiwań. Najdokładniejszy oraz — co ważniejsze — najskuteczniejszy, z nich okazał się wepawet. Niestety, to że jest serwisem internetowym, w zasadzie dyskwalifikuje go w codziennej pracy, gdzie istnieje potrzeba przetwarzania wielu plików wsadowo.

Pozostałe, jsunpack oraz PhoneyC, pozwalają na pracę z wieloma stronami WWW/plikami. Natomiast posiadają kilka wad: na przykład nie są odporne na błędy składni (X)HTML-a. Należy jednak winić za to parser HTML lub JavaScipt (SpiderMonkey), a nie same narzędzia. Kolejnym minusem w obu tych narzędziach jest nie podążanie za łączami znalezionymi w skryptach, mimo iż zostają one poprawnie znalezione. Co dziwniejsze, Pho-



Rysunek 15: Studium przypadku #2: PhoneyC



Rysunek 16: PhoneyC vs. jsunpack

neyC próbuje odwiedzać łącza typu `mailto:`, co nie ma większego sensu.

PhoneyC oraz jsunpack podczas statycznej analizy wykorzystują zbiór reguł, który nie jest aktualizowany. Możliwe, że skuteczność tych narzędzi byłaby większa, gdyby zbiór reguł był większy, gdyż ich standardowa ilość nie jest duża.

Oczywiście programy posiadają też zalety. Jedną z nich jest szybkość działania, która w zasadzie ograniczona jest jedynie przepustowością łącza

internetowego.

Poza tym, niewątpliwym plusem jsunpack'a jest zdolność analizy animacji Flash. Narzędzie dobrze sobie radzi także z zaciemnionymi oraz skompresowanymi PDF-ami. Ciekawą opcją jest nasłuchiwanie bezpośrednio na interfejsie sieciowym, umożliwiające badanie ruchu sieciowego w czasie rzeczywistym.

Literatura

[1] V. M. Alvarez. Yara: A malware identification and classification tool. `http://code.google.com/p/yara-project/`.

[2] S. Ford, M. Cova, C. Kruegel, and G. Vigna. wepawet. `http://wepawet.iseclab.org/`.

[3] Blake Hartstein. jsunpack. `http://jsunpack.jeek.org/dec/go`.

[4] T. Kojm. Clamav. `http://clamav.net/`.

[5] International Secure Systems Lab. Anubis: Analyzing unknown binaries. `http://anubis.iseclab.org/`.

[6] MDL. Malware domain list. `http://www.malwaredomainlist.com/`.

[7] Mozilla. Spidermonkey. `http://www.mozilla.org/js/spidermonkey/`.

[8] J. Nazario. Phoneyc. `http://code.google.com/p/phoneyc/`.

[9] Hispasec Sistemas. Virustotal - free online virus and malware scan. `http://www.virustotal.com`.

[10] W3C. Document object model. `http://www.w3.org/DOM/`.