Prezentujemy Wam dziewiąty numer naszego S.M.S.-owego biuletynu bezpieczeństwa komputerowego „Z firewall’a wzięte”.

Spis treści

Wstęp

Przygotowaliśmy dla Was dziewiąty numer naszego biuletynu bezpieczeństwa komputerowego „Z firewall’a wzięte” analizujący zagrożenia 2020 roku. Znów macie możliwość zajrzenia do naszej infrastruktury i przekonania się z jakimi podatnościami mierzymy się codziennie. Zapraszamy również do zapoznania się z pierwszym numerem biuletynu, gdzie szczegółowo opisujemy powody dla, których postanowiliśmy tworzyć takie materiały. Standardowo biuletyn składa się z 3 głównych części – ogólnej analizy statystycznej incydentów w naszej infrastrukturze, analizy przypadku jednego z wybranych przez nas zagrożeń oraz podsumowania całego zebranego przez nas materiału. Na końcu dodaliśmy również sekcję „Poprzednie numery”, co pozwoli Ci łatwo znaleźć wcześniejsze wydania biuletynu. Numer kwietniowy również powstawał we współpracy z Fudo Security. Dzięki temu, że umożliwili nam testowanie swojego rozwiązania Fudo PAM, dalej mogliśmy monitorować nasze sesje oraz sprawdzać co uda nam się złowić na naszym firewall’u. Aby dowiedzieć się, więcej o samym Fudo Security oraz sprawdzić nad czym pracują wpadnijcie na ich profile w social mediach – LinkedIn, Twitter, Facebook. Koniecznie dajcie znać, że przysłało Was S.M.S.! Chcielibyśmy również zachęcić naszych kolegów z branży do przyłączenia się do naszej inicjatywy i współpracy przy tworzeniu kolejnych wydań biuletynu. Jeśli masz pomysł jak wykorzystać Twój potencjał, pomysł lub produkt w materiale serdecznie zapraszamy do kontaktu mailowego w celu ustalenia szczegółów: blog@s-m-s.pl.

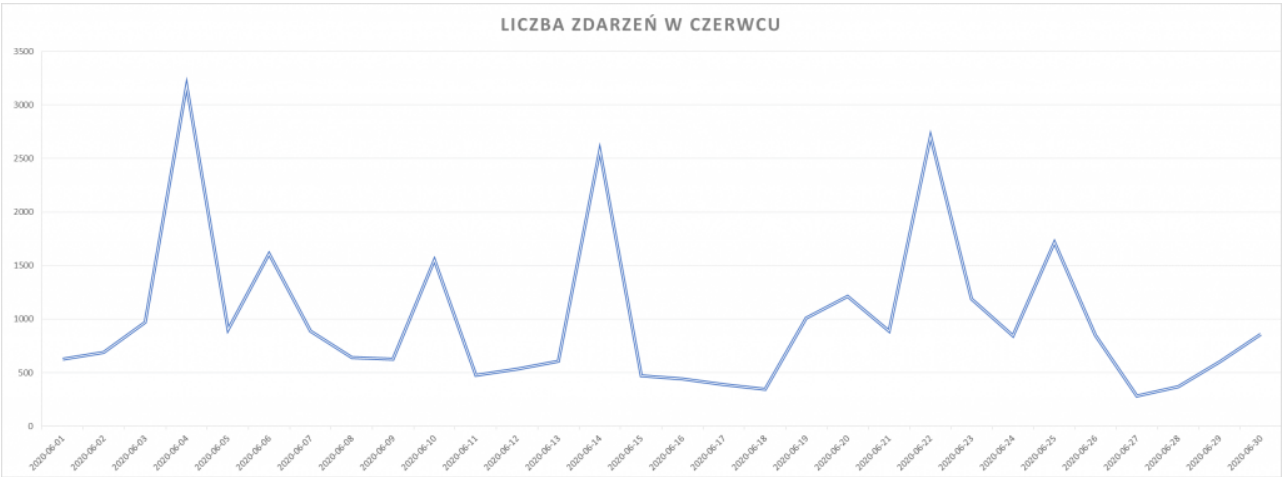
Mamy nadzieję, że zapoznanie się z materiałem sprawi Ci tyle satysfakcji ile nam sprawiło jego przygotowanie. Zapraszamy również do dyskusji na jego temat we wszystkich

dostępnych kanałach – sekcja komentarzy na naszym blogu, nasze profile w social mediach (Facebook oraz Twitter) czy też pod adresem mailowym: blog@s-m-s.pl. Każda opinia na ten temat jest dla nas ważna i pomoże nam ulepszyć kolejne wydania biuletynu.

Miłej lektury!

Ogólna analiza statystyczna incydentów

W celu określenia skali i częstotliwości występowania zdarzeń w infrastrukturze najlepszym będzie przeanalizowanie dostępnych danych statystycznych. Dzięki takiemu zabiegowi będziemy mogli w sposób kompleksowy przedstawić kwestie cyberbezpieczeństwa naszej infrastruktury. Na potrzeby przygotowania tej części materiału wykorzystaliśmy technologię umożliwiającą nam stałe monitorowanie ruchu do naszych serwerów. Pozwoliło nam to wyszczególnić zdarzenia, które zostały przedstawione poniżej. Podobnie jak w poprzednich wydaniach badaniu poddane zostały dane zebrane z własnych narzędzi służących do administrowania ruchem do serwerów. Przeanalizowaliśmy dane za okres od 01.06.2020 do 30.06.2020. Do analizy statystycznej użyte zostały takie parametry jak dzienna liczba zdarzeń, najczęściej występujące incydenty, podział zagrożeń ze względu na rodzaj oraz potencjalną dotkliwość zdarzenia. Udało nam się wyodrębni 30.105 zdarzeń w czerwcu. Średnia liczba zagrożeń w tym miesiącu wyniosła 1003,5 zdarzeń dziennie. Liczbę zdarzeń występujących w każdym dniu zeszłego miesiąca obrazuje *Wykres nr 1.*



Wykres nr 1 Liczba wykrytych zdarzeń w czerwcu

W czerwcu ponownie liczba prób kompromitacji naszych serwerów była bardzo zmienna.

Często liczba zdarzeń jednego dnia nie przekraczała 1.000. Można wyraźnie zaważyć, że w dniach 04.06.2020, 14.06.2020 oraz 22.06.2020 liczba zdarzeń zdecydowanie wzrosła. Wtedy odnotowaliśmy znaczący wzrost prób zagrożenia naszym systemom odpowiednio 3.181, 2.574 oraz 2.705.. Poniżej prezentujemy tabele, w których przedstawione zostały rodzaje zagrożeń oraz liczba takich zdarzeń w danym dniu.

Rodzaj zagrożenia	Liczba zdarzeń
HTTP SQL Injection Attempt	1503
MySQL Authentication Brute-force Attempt	1100
OpenSSH AES-GCM Auth Remote Code Execution Vulnerability	489
UNIX Portmapper Remote Infomation Retrieving Attempt	57
PHP CGI Query String Parameter Handling Information Disclosure and DoS Vulnerability	17
HTTP Non RFC-Compliant Response Found	7
HTTP Unauthorized Brute-force Attack	5
FTP: login Brute-force attempt	2
WordPress Cuckootap Theme Arbitrary File Download Vulnerability	1

Tab. nr 1 Typy zdarzeń oraz ich liczba w dn. 04.06.2020 r.

Rodzaj zagrożenia	Liczba zdarzeń
MySQL Authentication Brute-force Attempt	2200
OpenSSH AES-GCM Auth Remote Code Execution Vulnerability	260
UNIX Portmapper Remote Infomation Retrieving Attempt	58
HTTP SQL Injection Attempt	42
HTTP Non RFC-Compliant Response Found	10
WordPress Cuckootap Theme Arbitrary File Download Vulnerability	2
FTP: login Brute-force attempt	1
DNS Zone Transfer AXFR Response	1

Tab. nr 1 Typy zdarzeń oraz ich liczba w dn. 14.06.2020 r.

Rodzaj zagrożenia	Liczba zdarzeń
MySQL Authentication Brute-force Attempt	2100
OpenSSH AES-GCM Auth Remote Code Execution Vulnerability	409
UNIX Portmapper Remote Infomation Retrieving Attempt	129
HTTP SQL Injection Attempt	48
HTTP Non RFC-Compliant Response Found	10
WordPress Cuckootap Theme Arbitrary File Download Vulnerability	3
FTP: login Brute-force attempt	2
DNS Zone Transfer AXFR Attempt	1
DNS RRSIG Query Type Packet	1
HTTP OPTIONS Method	1
DNS Zone Transfer AXFR Response	1

Tab. nr 1 Typy zdarzeń oraz ich liczba w dn. 22.06.2020 r.

Przedstawiamy Wam też nasz Top 20, czyli listę najbardziej popularnych zagrożeń w lipcu. Są to najczęściej wykorzystywane typy zagrożeń, poprzez użycie których atakujący próbowali skompromitować naszą infrastrukturę. Poniżej tabela zagrożeń wraz z liczbą zdarzeń.

Rodzaj zagrożenia	Liczba zdarzeń
OpenSSH AES-GCM Auth Remote Code Execution Vulnerability	10648
MySQL Authentication Brute-force Attempt	7300
FTP: login Brute-force attempt	3200
HTTP SQL Injection Attempt	2546
UNIX Portmapper Remote Infomation Retrieving Attempt	2500
SSH User Authentication Brute-force Attempt	1600
MS-RDP Brute-force Attempt	1300
HTTP Non RFC-Compliant Response Found	397
HTTP Directory Traversal Vulnerability	155
PHP CGI Query String Parameter Handling Information Disclosure and DoS Vulnerability	146
DNS RRSIG Query Type Packet	97
HTTP Unauthorized Brute-force Attack	66
HTTP OPTIONS Method	35
WordPress Cuckootap Theme Arbitrary File Download Vulnerability	34
ZmEu Scanner Detection	25
MailEnable IMAP Server Long Tag anomaly	17
DNS Zone Transfer AXFR Attempt	13
DNS Zone Transfer AXFR Response	13
Invalid HTTP Version Found	13

Tab. nr 3 Typy zdarzeń oraz ich liczba w czerwcu

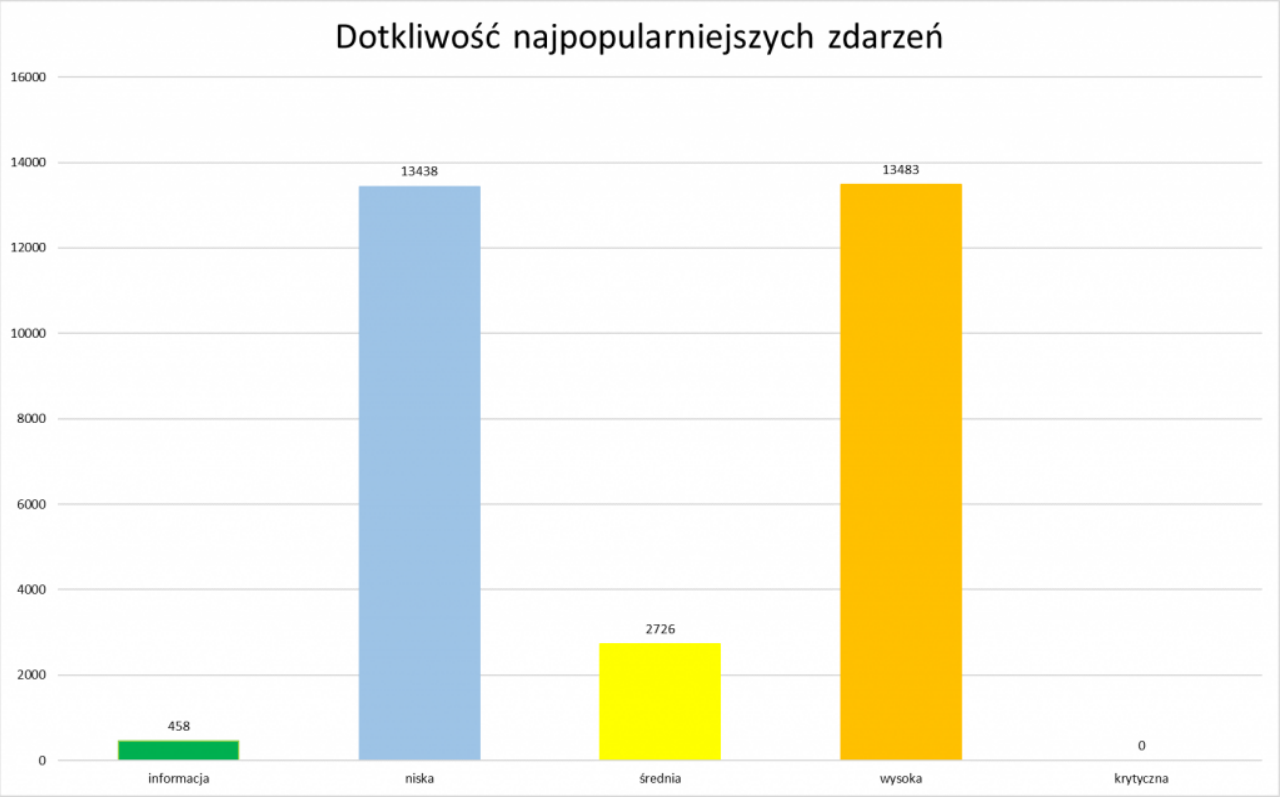
Wyżej wspominamy o pojęciu potencjalnej dotkliwości zdarzenia. Jak sama nazwa wskazuje jest to szacowany zakres szkód jakie może wyrządzić dana podatność w naszej infrastrukturze o ile dojdzie do jej pomyślnego wykorzystania. Samą dotkliwość można podzielić na 5 różnych poziomów:

- Informacja – podejrzane zdarzenie, które nie stanowi bezpośredniego zagrożenia, ale poprzez samo jego zgłoszenie uwaga administratora może zostać zwrócona na głębsze problemy infrastruktury, które mogą zaistnieć w przyszłości.
- Niska – najniższy poziom dotkliwości wymagający ostrzeżenia. Zagrożenie ma znikomy wpływ na infrastrukturę organizacji. Zazwyczaj wymagają lokalnego bądź fizycznego dostępu do systemu i często mogą powodować problemy z prywatnością ofiary lub problemy powiązane z DoS oraz możliwy wyciek danych.
- Średnia – niewielkie zagrożenie, którego wpływ na infrastrukturę jest minimalny. Następstwem wykorzystania podatności z tej kategorii mogą być ataki typu DoS, które nie zagrażają celowi lub exploity, które od osoby atakującej wymagają przebywania

w tej samej sieci LAN co ofiara. Zagrożenia poziomu średniego mogą mieć wpływ jedynie na niestandardowe konfiguracje oraz mało znane aplikacje. Zapewniają atakującemu bardzo ograniczony dostęp.

- Wysoka – zagrożenie, które potencjalnie może stać się krytycznym, jednak dzięki występowaniu czynników łagodzących nie jest możliwa jego eskalacja. Do kategorii zagrożeń poziomu wysokiego można zaliczyć zagrożenia, które są trudne do wykorzystania, nie dają podwyższonych uprawnień lub są w stanie dotknąć małej ilości ofiar.
- Krytyczna – zagrożenie poważne, które jest stanie dotknąć domyślnych instalacji szeroko rozpowszechnionego oprogramowania. Skutkuje kompromitacją serwera, a kod exploitacji jest powszechnie dostępny. Atakujący zwykle nie potrzebuje żadnych specjalnych danych uwierzytelniających ani wiedzy na temat poszczególnych ofiar, a cel nie musi być zmanipulowany w celu wykonywania jakichkolwiek specjalnych funkcji.

Na poniższym wykresie przedstawiliśmy potencjalną dotkliwość najpopularniejszych zdarzeń występujących w badanym okresie.



Wykres nr 2 Potencjalna dotkliwość najpopularniejszych zdarzeń w badanym okresie.

Analiza przypadku wybranych zgłoszeń

W jednym z pierwszych biuletynów zaznaczyliśmy, że w kolejnych wydaniach będziemy omawiać jedynie przypadki oznaczone w dotkliwości jako „wysoka" oraz „krytyczna". Dzisiaj jednak złamiemy tę zasadę, ponieważ podatność, której przyjrzymy się w tym numerze jest na tyle ciekawa że warto o niej wspomnieć. Zaczynajmy!

Ostatnio w naszej firmie praktykanci dostali bardzo ciekawe, a zarazem skomplikowane zadanie. Otrzymali plik tekstowy wypełniony ciągiem liter i cyfr, a treść zadania wskazywała, że jest to funkcja napisana w języku php, którą należy przywrócić do stanu najbardziej zbliżonego do oryginału. Od razu stało się jasne, że **kod jest** w jakiś sposób **zaszyfrowany** i należy odnaleźć sposób, aby go w „ludzki" sposób przeanalizować. Szybko okazało się, że kod jest zaciemniony, co sprowokowało dwa pytania odnośnie tego procesu:

1. W jakim celu zaciemnia się kod?

Pytanie najczęściej cisnące się na usta, nie tylko naszym praktykantom, ale większości osób, które słyszą pierwszy raz o tym procesie. Po co? W jakim celu? Odpowiedź na to pytanie jest bardzo prosta i można ją zamknąć w jednym zdaniu. Zaciemnienie kodu (zwane obfuskacją) stosuje się w celu ukrycia sposobu działania odpowiednio zabezpieczonego programu. Załóżmy, że przez chwilę jesteśmy wydawcą „jakiejś" gry. Celowo tutaj nie podaje jakiego typu, ponieważ obfuskację można stosować tak samo dla kodu źródłowego napisanego w języku PHP, JS, jak i C, C# itd. Po napisaniu i wypuszczeniu swojej aplikacji, już na początku czujemy swędzenie z tyłu głowy i zadajemy sobie pytania czy aby na pewno nasza aplikacja jest dobrze zabezpieczona przed skopiowaniem lub zrozumieniem jej działania. Jakie to ostatnie niesie za sobą niebezpieczeństwo? Ano takie, że potencjalny haker mógłby zrozumieć jak działają nasze zabezpieczenia i w prosty sposób je obejść. Obfuskacja znacznie utrudnia ten proces, ponieważ przekształca kod źródłowy przy zachowaniu semantyki. Utrudnia to analizę i zrozumienie kodu, co niejako chroni przed naruszeniem własności intelektualnej.
Oczywiście w myśl zasady „no system is safe" obfuskowany kod prędzej czy później można złamać, jednak dosyć często czas i zasoby potrzebne na dokonanie odkodowania są niewspółmierne do zysków.

Innym powodem do zaciemnienia kodu może być przeprowadzenie małego ataku hakerskiego, o którym przykładzie powiemy później.

Dla ciekawskich – pierwszy raz definicję konfiskacji można było znaleźć w artykule

autorstwa Christiana Collberga, pt. „A taxonomy of obfuscating transformations”.

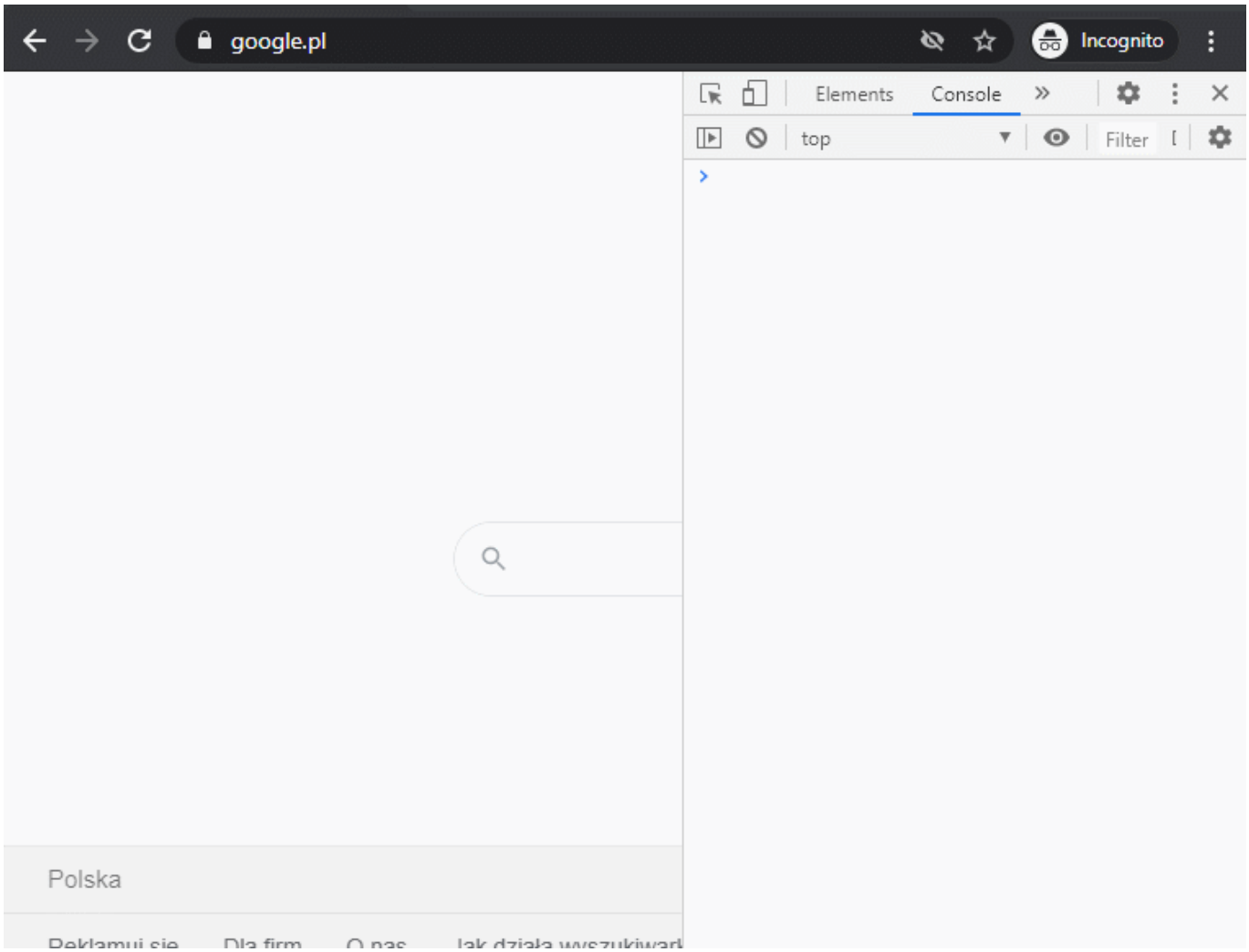
2. Jak zaciemnia się kod?

Sposobów na zaciemnienie kodu jest cała masa, a różnorodność programów do tego celu jest ogromna. Od darmowych narzędzi dostępnych w Internecie, przez aplikacje dostępne na komputery osobiste aż do dużych, komercyjnych projektów.

Do zaciemnienia kodu PHP możemy użyć np. programu Zend Guard, do obfuskacji C# – eazfuscator.net, a do Javascript narzędzia o wdzięcznej nazwie jsfuck.com.
Jak wygląda takie zaciemnianie kodu? Najczęściej po wybraniu narzędzia i sposobu zaciemniania musimy wklepać kod, który chcemy ukryć…

Przykład użycia narzędzia JSFuck

I już. Możemy teraz sprawdzić działanie obfuskowanej funkcji



Teraz spróbujmy zaciemnić funkcję php.

Załóżmy, że napisaliśmy funkcję, przy pomocy której rysujemy trójkąt prostokątny z gwiazdek.
W „ludzkim” języku wygląda ona następująco

```
for($a=5; $a>=0; $a--)
{
    for($j=$a; $j>0; $j--)
        echo " ";
    for($j=$a; $j<5; $j++)
        echo "*";
    echo "<br>";
}
```

```
*
**
***
****
*****
```

Użyjmy teraz dowolnego obfuskatora znalezionego w Internecie, aby ten kod zaciemnić. W tym celu skorzystamy z podstawowego rodzaju kodowania – Base64. Po przepuszczeniu kodu przez nasze narzędzie na wyjściu otrzymamy następujący ciąg znaków:

<?php

eval(base64_decode('Zm9yKCRhPTU7ICRhPj0wOyAkYS0tKQp7CiAgICBmb3IoJGo9JGE7ICRqPjA7ICRqLS0pCiAgICAgICAgZWNobyAiICI7CiAgICBmb3IoJGo9JGE7ICRqPDU7ICRqKyspCiAgICAgICAgZWNobyAiKiI7CiAgICBlY2hvICI8YnI+IjsKfQ=='));

?>

Nasz wynik nijak nie przypomina oryginalnego kodu, prawda?

Zagrożenia

Mamy za sobą odpowiedź na pytanie skąd i po co się to wzięło. Mamy także przykład zastosowania obfuskacji dla prostych funkcji. Wypada teraz powiedzieć co nieco o zagrożeniach związanych z obfuskacją kodu.

Na atak tego typu najczęściej narażona jest nasza skrzynka pocztowa. Co wspólnego ma skrzynka pocztowa z zaciemnionym kodem? Dzięki takiemu zabiegowi możliwe jest

przepuszczenie spamu przez filtry w słabo zabezpieczonej poczcie. W skrajnych przypadkach wystarczy, że w treść będzie wyglądać następująco

Reklama

i filtry w skrzynce pocztowej potraktują spam jako normalną, nie wzbudzającą podejrzeń korespondencję, a sama treść zostanie wyświetlona poprawnie.

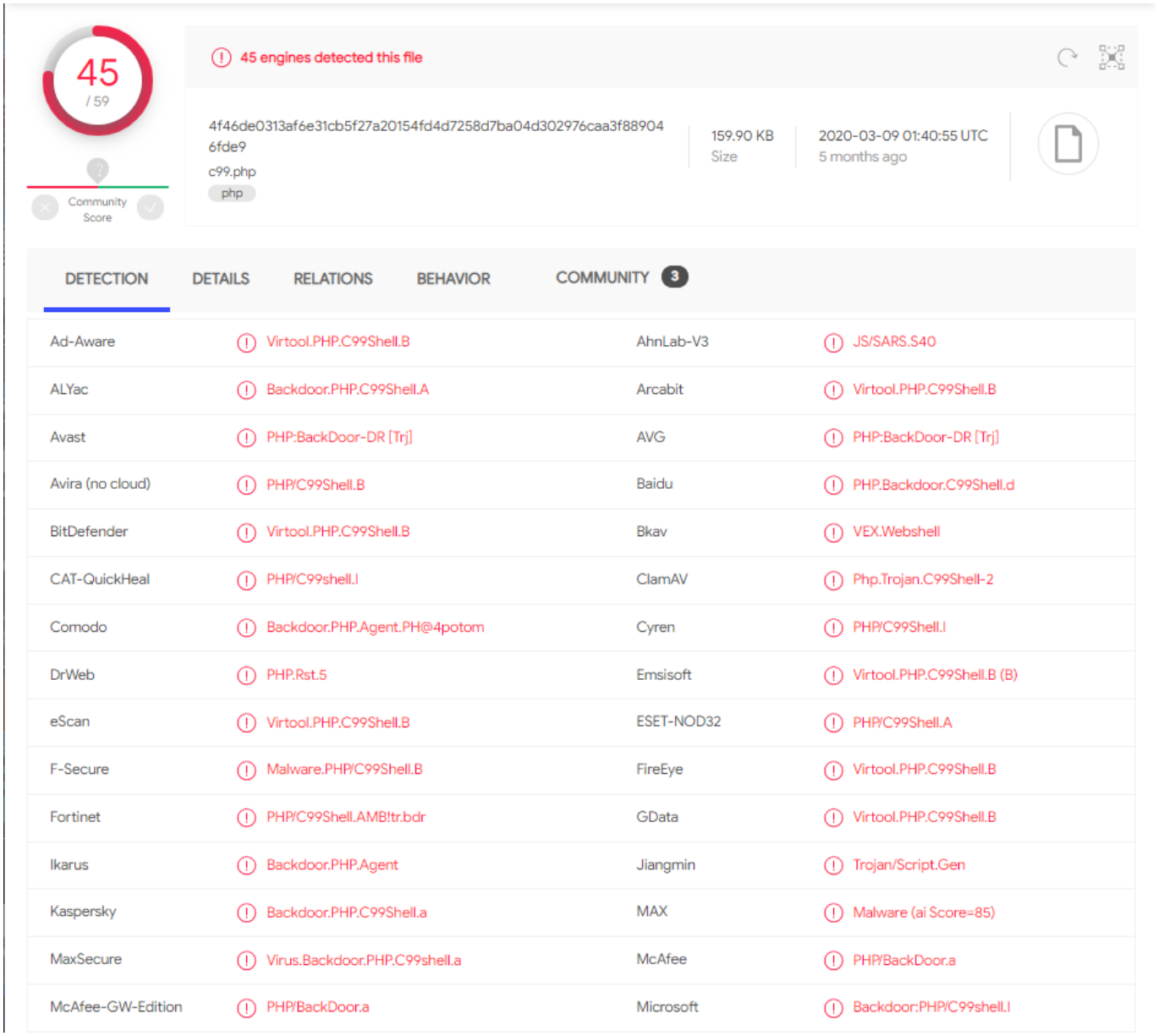
Innym przypadkiem, bardzo podobnym do wspomnianego wyżej jest przeprowadzenie ataku XSS. Atakujący za pomocą zaciemnionego kodu są w stanie osadzić go na stronie internetowej, obchodząc w ten sposób zastosowane w kodzie słabsze filtry, które miały chronić przed tego typu atakiem. A co się stanie później? Wszystko zależy od intencji atakującego. Może wykraść dane sesyjne czy nawet podmienić zawartość strony.

Poza wymienionym wyżej atakiem istnieje bardzo ciekawy sposób wykorzystania zaciemniania kodu, o którym nie każdy z „wtajemniczonych" wie. Zdarza się, że wspomniana metoda używana jest wspólnie z funkcjami php jako backdoor w Web Shell.

> *Backdoor, czyli „tylna furtka" jest celowo tworzoną luką w zabezpieczeniach danego systemu, która w późniejszym czasie może zostać wykorzystana na wiele różnych sposobów, najczęściej w złej intencji. Backdoory mogą zostać tworzone na potrzeby kontroli/śledzenia użytkowników przez twórców aplikacji oraz w skrajnych przypadkach przez rząd (do czego może nie wprost ale jednak przyznali się twórcy po rozkręceniu aferki z WikiLeaks). Tylnych furtek używają także bardzo często hakerzy w wiadomych celach.*

Przykładem takiego backdoora jest c99.php. Do czego służy obfuskowanie w tym przypadku? Odpowiedź jest prostsza niż się wydaje. Zaciemniony kod jest niewykrywalny przez antywirusa.
I tyle. Zdecydowana większość antywirusów bazuje na interpretacji kodu. Wydaje się to dosyć zabawne na swój sposób, ale wystarczy (oczywiście w dużym uproszczeniu) pozamieniać literki miejscami i nagle antywirus staje się ślepy.

Poniższy przykład przedstawia jak to wygląda w praktyce. Na pierwszym zrzucie ekranu widzimy skan pliku zawierającego kod źródłowy c99.php, który nie jest zaciemniony.



45 / 59

Community Score

45 engines detected this file

4f46de0313af6e31cb5f27a20154fd4d7258d7ba04d302976caa3f889046fde9

c99.php

php

159.90 KB Size

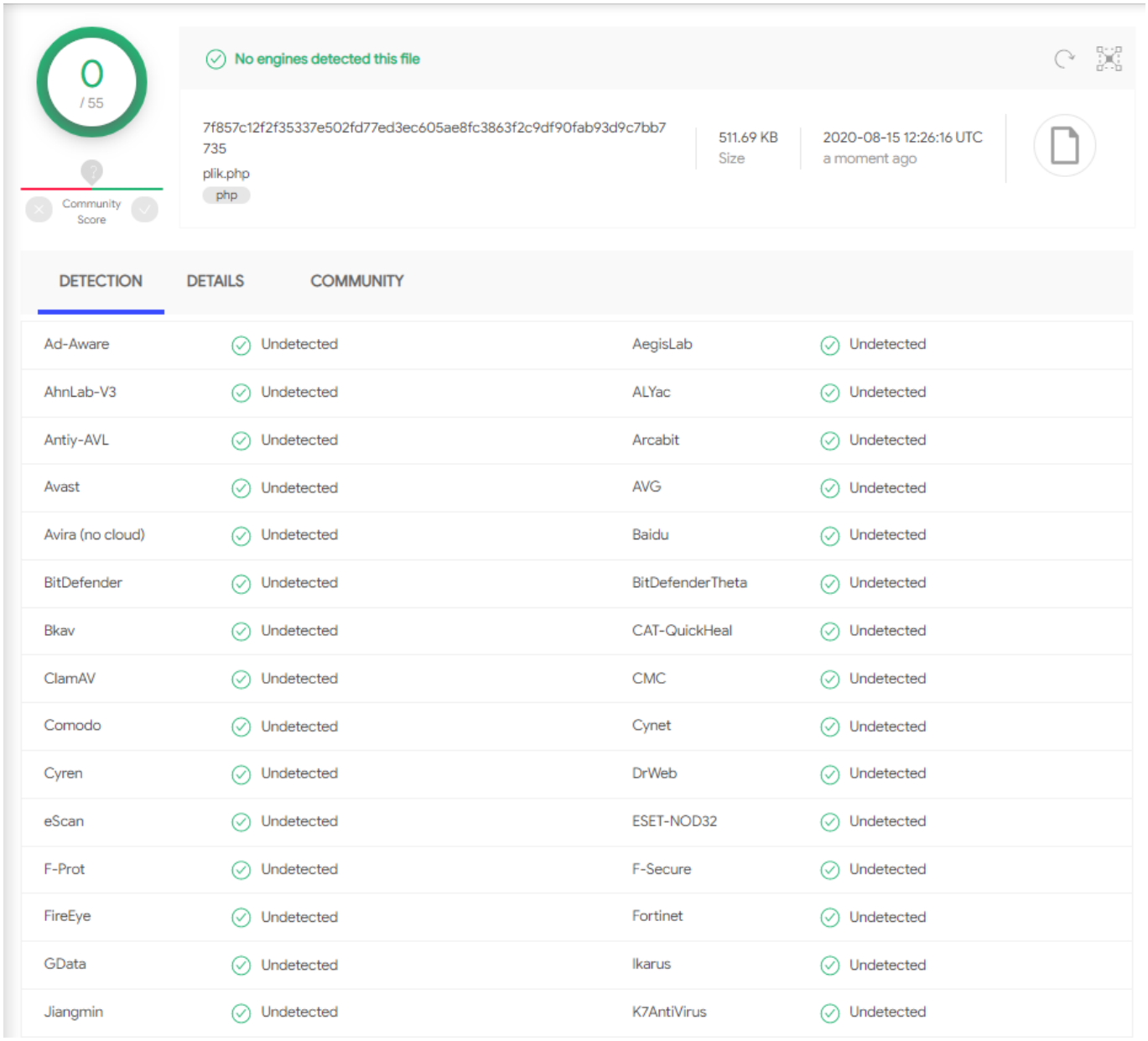
2020-03-09 01:40:55 UTC 5 months ago

DETECTION | DETAILS | RELATIONS | BEHAVIOR | COMMUNITY 3

Ad-Aware	Virtool.PHP.C99Shell.B	AhnLab-V3	JS/SARS.S40
ALYac	Backdoor.PHP.C99Shell.A	Arcabit	Virtool.PHP.C99Shell.B
Avast	PHP:BackDoor-DR [Trj]	AVG	PHP:BackDoor-DR [Trj]
Avira (no cloud)	PHP/C99Shell.B	Baidu	PHP.Backdoor.C99Shell.d
BitDefender	Virtool.PHP.C99Shell.B	Bkav	VEX.Webshell
CAT-QuickHeal	PHP/C99shell.I	ClamAV	Php.Trojan.C99Shell-2
Comodo	Backdoor.PHP.Agent.PH@4potom	Cyren	PHP/C99Shell.I
DrWeb	PHP.Rst.5	Emsisoft	Virtool.PHP.C99Shell.B (B)
eScan	Virtool.PHP.C99Shell.B	ESET-NOD32	PHP/C99Shell.A
F-Secure	Malware.PHP/C99Shell.B	FireEye	Virtool.PHP.C99Shell.B
Fortinet	PHP/C99Shell.AMB!tr.bdr	GData	Virtool.PHP.C99Shell.B
Ikarus	Backdoor.PHP.Agent	Jiangmin	Trojan/Script.Gen
Kaspersky	Backdoor.PHP.C99Shell.a	MAX	Malware (ai Score=85)
MaxSecure	Virus.Backdoor.PHP.C99shell.a	McAfee	PHP/BackDoor.a
McAfee-GW-Edition	PHP/BackDoor.a	Microsoft	Backdoor:PHP/C99shell.I

Wynik analizy mówi sam za siebie.

Poniżej zamieszczamy screen nr 2. W tym przypadku kod c99.php został zaciemniony za pomocą losowej aplikacji online służącej do obfuskacji kodu php.



Jeżeli nie wierzycie nam na słowo, możecie przeprowadzić taki test sami. Wystarczy pobrać kod źródłowy c99.php z githuba i przetestować go chociażby w serwisie virustotal.com. Tak na marginesie, jeżeli korzystacie z Windows Defendera to w zasadzie wystarczy przygotować dwa pliki z kodem (jeden zwykły, drugi obfuskowany) i zobaczycie, że wykryje tylko jeden z nich. Zgadnijcie sami który…

Podsumowanie

Obfuskowanie kodu ma swoje zalety i wady. Jest na pewno ciekawym i przede wszystkim prostym narzędziem do zapobiegania kradzieży oraz ochrony własności intelektualnej. Jak wszystko, w niewłaściwych rękach może narobić szkód jednak nie są one aż tak inwazyjne jak w przypadku innych podatności. Uważamy, że warto zainteresować się zaciemnianiem kodu. Zdecydowanie wiedza w tym zakresie nam nie zaszkodzi, a dobrze wykorzystane obfuskowanie może nam tylko pomóc.

Poprzednie numery

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 1 10/19

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 2 11/19

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 3 12/19

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 4 01/20

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 5 02/20

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 6 03/20

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 7 04/20

Z firewall’a wzięte czyli biuletyn bezpieczeństwa komputerowego S.M.S. Nr 8 05/20

Post powstał we współpracy z Fudo Security. Nie jest to materiał sponsorowany, a wszystkie opinie zawarte w biuletynie należą do S.M.S. i są jedynie naszymi spostrzeżeniami. Serdecznie dziękujemy kolegom i koleżankom z Fudo Security za umożliwienie nam testowania rozwiązania Fudo PAM.