JURNAL INFOTEL
Informatika - Telekomunikasi - Elektronika

Website Jurnal : http://ejournal.st3telkom.ac.id/index.php/infotel
ISSN : 2085-3688; e-ISSN : 2460-0997

Basic Static Code Analysis untuk Mendeteksi *Backdoor Shell* pada *Web Server*

Nelly Indriani Widiastuti[1], Muhammad Iqbal[2]
[1,2]Teknik Informatika Universitas Komputer Indonesia
[1,2]Dipati Ukur 112-118, Bandung, Indonesia
Email korespondensi : nelly.indriani@email.unikom.ac.id

Dikirim 14 April 2017, Direvisi 3 Mei 2017, Diterima 13 Mei 2017

Abstrak – Mengakses sistem komputer tanpa ijin merupakan kejahatan yang dilakukan dengan memasuki atau menyusup ke dalam suatu sistem jaringan komputer tanpa sepengetahuan dari pemilik sistem tersebut. Kejahatan tersebut bertujuan untuk mengintai atau mencuri informasi penting dan rahasia. Dalam praktiknya, peretas menyisipkan berkas *backdoor shell* pada lokasi yang sulit ditemukan oleh pemilik sistem. Beberapa perangkat yang sudah ada masih dalam bentuk terminal. Perangkat tersebut melakukan pencarian berkas berdasarkan nama-nama yang telah terdaftar sebelumnya. Akibatnya, pada saat berkas *backdoor shell* jenis baru menginfeksi, *tools* tersebut tidak dapat mendeteksi keberadaannya. Berdasarkan hal tersebut, maka dalam penelitian ini pencarian *backdoor shell* pada web server menggunakan metode *basic static code analysis*. *File* sistem diproses melalui dua tahap utama, yaitu *string matching* dan *taint analysis*. Dalam proses *taint analysis*, sistem menghitung peluang kemungkinan setiap *signature* sebagai *backdoor* untuk mengatasi kamus *backdoor* yang tidak lengkap. Berdasarkan hasil yang didapat dari pengujian yang dilakukan terhadap 3964 berkas diperoleh tingkat akurasi yang lebih besar dibandingkan dengan aplikasi PHP Shell Detector sebesar 75%.

Kata kunci – *Basic Static Code Analysis*, *Backdoor Shell*, Keamanan, *String Matching*, *Web Server*

Abstract - Unauthorized access to computer system is the crime committed by entering or infiltrated into system of computer network without permissions of the system owner. The criminal do it with intention of spying or stealing sensitive and confidential information. In practice the perpetrator install a backdoor shell on web server directory that's hard to found by the system owner. Some of existing tools are still in terminal form and searching backdoor shell based on previous registered list. As a result if the system infected by new kind of backdoor shell, the tools cannot found the existing based on this, then in this study search the backdoor shell on the web server using basic static code analysis method. File system through two main processes of the string matching and taint analysis. In the taint analysis process, the system calculates the probability of each signature as a backdoor to resolve an incomplete backdoor dictionary. Based on the results obtained from the testing on 3964 files, it can be concluded that this application have a better at 75% accuracy compared to the PHP Shell Detector application.

Keywords - Basic Static Code Analysis, Backdoor Shell, Security, String Matching, Web Server

I. PENDAHULUAN

Pada awal munculnya *e-commerce* terdapat cukup banyak kerentanan di server web dan sistem operasi balckhat bisa dengan mudah mengeksploitasi unsur-unsur infrastruktur [1]. Eksploitasi tersebut sangat merugikan sehingga hal ini dinamakan *cybercrime*. Dalam hal ini sistem menjadi sasaran kegiatan kriminal, tempat penyimpanan data tentang kegiatan kriminal dan/atau alat yang digunakan untuk melakukan kejahatan (merencanakan kegiatan criminal) [2].

Backdoor shell merupakan kumpulan kode-kode atau *script* yang pada umumnya berbasis PHP yang dimanfaatkan oleh peretas untuk memperoleh akses pada *file* yang disimpan di *web server* target [3]. Jika pengelola *website* tidak dapat mendeteksi keberadaan *backdoor shell* maka hal itu dapat menimbulkan efek jangka panjang. Peretas dapat dengan mudah mendapatkan data-data rahasia pada sistem secara

berkala. Hal tersebut sebenarnya dapat diatasi dengan melakukan *restore backup file* pada saat pengelola web menyadari ada *file* penyusup di dalam sistem, namun cara ini tidak efektif untuk *file store* yang berkapasitas besar. Selain itu, pengelola tidak mengetahui lama waktu sistem telah terinfeksi oleh *backdoor shell* sehingga kerugian karena kehilangan data pun tidak diketahui.

Saat ini ada beberapa alat yang dapat digunakan pengelola *web server* untuk melakukan pencarian *backdoor shell* seperti NeoPI [4] dan PHP Shell Detector [5]. NeoPi merupakan *unix command* unruk mencari *file backdoor shell*. Hal ini membuat pengelola web yang tidak familiar dengan sistem operasi unix/linux mengalami beberapa kesulitan dalam menggunakannya, yaitu hasil dari pencarian masih berupa daftar pada terminal sehingga pengguna harus melakukan pencarian manual pada *web server* sesuai alamat yang telah ditemukan, membutuhkan waktu dalam pengecekannya, dan *list* yang ditampilkan tidak hanya *file backdoor shell*, tetapi juga *file* legal yang memiliki fungsi sama. Perangkat lainnya, yaitu PHP Shell Detector, merupakan *script* berbasis PHP yang sudah memiliki GUI untuk mempermudah pengguna. Keterbatasan PHP Shell Detector adalah pencarian *file* berdasarkan nama-nama telah terdaftar sebelumnya sehingga saat jenis *file backdoor shell* baru menginfeksi, alat ini tidak dapat mendeteksi keberadaan *backdoor shell* tersebut.

Static source code analysis adalah metode terukur untuk penemuan kesalahan perangkat lunak dan kerentanan keamanan. Teknik untuk *static source code analysis* telah banyak digunkan dalam decade terakhir [6], termasuk untuk keamanan reaktor nuklir [7] dan deteksi *wireless network* [8], meskipun masih dalam tahap pengembangan. Beberapa penelitian yang berkaitan dengan *backdoor* dan *static code analysis* berhasil mengurangi serangan ke dalam sistem autentifikasi. Sebuah penelitian mendeteksi *backdoor* dengan cara meng-*generate verification signature* dan kemudian dibandingkan antara pengujian dan respon. Sistem ini berjalan hanya untuk *native code C/C++* [9]. Penelitian tentang *static code* dilakukan oleh Prabhu Seshagiri, dkk. Mereka mengembangkan Amrita Malware Analyzer (AMA). *Framework* ini mampu mendeteksi kehadiran kode berbahaya melalui *static code analysis* pada halaman web [10]. Penelitian yang lain juga menyajikan *static source code analysis* yang disebut Pistachio yang memeriksa kode C terhadap spesifikasi berbasis aturan. Hasil analisis menunjukkan tidak sepenuhnya berhasil selama beberapa perkiraan *heuristic* terjadi, namun memiliki tingkat *false negative* yang rendah [11]. *Static analysis* juga dilakukan terhadap *Embedded Real-Time Concurrent Software* dengan prioritas yang berubah-ubah [12].

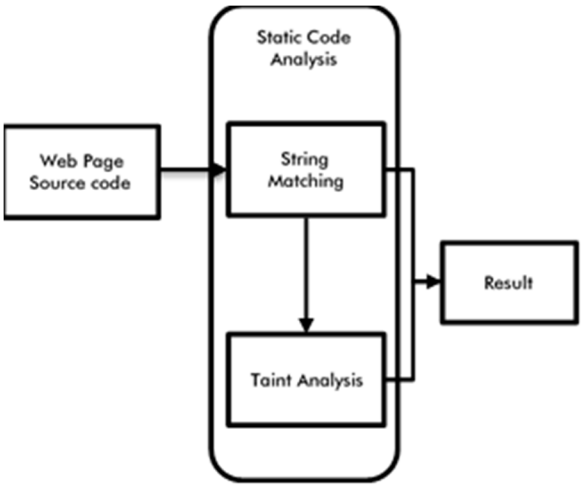
Dalam penelitian ini, *backdoor* akan diatasi dengan menggunakan Teknik *Basic Static Code Analysis* untuk mengoptimalkan identifikasi *file backdoor*. Dalam mengatasi kamus *backdoor* yang tidak lengkap, sistem melakukan perhitungan terhadap peluang kemungkinan setiap *signature* sebagai *backdoor*.

II. METODE PENELITIAN

Metode penelitian yang digunakan dalam penelitian ini adalah analisis deskriptif. Analisis dilakukan terhadap metode yang digunakan sehingga menggambarkan proses sistem untuk memperoleh hasil kerja *basic static code* dalam mendeteksi *backdoor* jika dibandingkan dengan perangkat lain, yaitu PHP Shell Detector. Teknik dan ilustrasi kerja *basic static code analysis* yang digunakan dalam penelitian ini dapat dilihat pada Gambar 1. Proses utama terbagi dua, yaitu [13]:

1. *String Matching* yang akan digunakan untuk menganalisis kata kunci pada *source code* yang memiliki indikasi sebagai *file* illegal seperti *backdoor sheel signature*.
2. *Taint Analysis* yang digunakan untuk menganalisis *source code* yang mengandung fungsi-fungsi yang diidentifikasi sebagai *backdoor shell*.

Dalam sistem ini *webpage source code* akan dicari ciri-ciri yang menunjukkan *backdoor shell*. Ciri tersebut biasa disebut *signature backdoor*. Secara umum, proses pencariannya dilakukan dengan metode *string matching*.



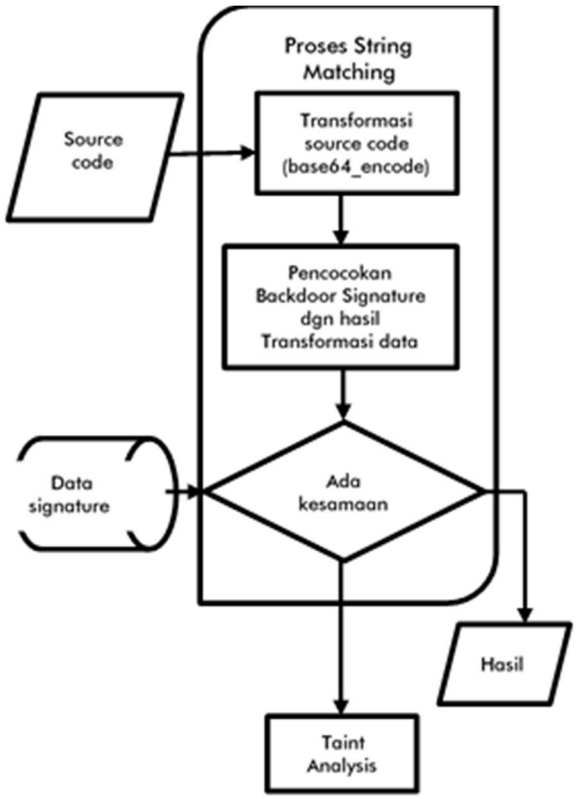
Gambar 1. Ilustrasi Kerja *Basic Static Code Analysis*

A. String Matching

Tahap *string matching* merupakan tahapan awal dalam menganalisis *source code*. Dalam modul ini, *file* sistem akan diproses untuk mencari *keyword* yang memiliki indikasi sebagai *backdoor shell*idengan cara pencocokan *string* pada *source code* dengan kamus data *backdoor signature* yang terdapat pada *database*. Gambaran umum proses *string matching* dapat dilihat pada Gambar 2.

Proses *string matching* dapat diterapkan pada jenis *backdoor encrypted* dan *non-encrypted* [14]. Apabila *file* yang dianalisis tidak menunjukkan adanya indikasi

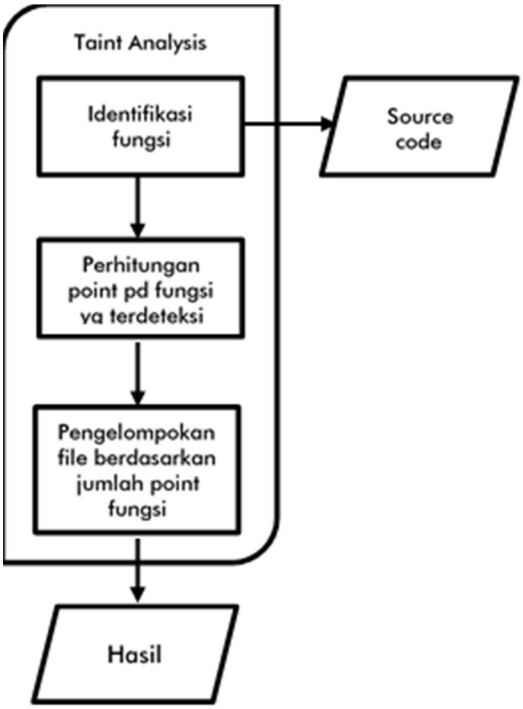
sebagai *backdoor shell* maka selanjutnya *file* akan diproses ke tahap *taint analysis*.



Gambar 2. Proses *String Matching*

B. *Taint Analysis*

Taint analysis merupakan tahapan analisis *source code* berdasarkan fungsi-fungsi yang dimiliki oleh *source code* tersebut. Berikut pada Gambar 3 adalah gambaran umum proses *taint analysis*.



Gambar 3. Proses *Taint Analysis*

Proses *taint analysis* secara umum adalah mengidentifikasi *source code* menggunakan *function matching*, jika terdapat fungsi yang berpotensi sebagai *backdoor shell* maka *file* tersebut akan ditandai sebagai *file tainted* [15]. Pada penelitian ini, *taint analysis* diproses dengan menghitung kemungkinan munculnya fungsi-fungsi (*sinks*) terhadap 40 *file backdoor shell*.

Hasil proses *taint analysis* merupakan keluaran yang menunjukkan *file* yang dideteksi merupakan *backdoor shell* atau bukan.

III. HASIL PENELITIAN

Dalam pengujian sistem, yang ingin diperoleh adalah seberapa akurat penggunaan *basic static code analysis* dalam mendeteksi *backdoor shell*. Adapun pengujian dilakukan pada sistem yang memiliki *user interface* dibagi menjadi dua bagian, yaitu:

1. Skenario pengujian deteksi menggunakan sistem yang dibangun
2. Skenario pengujian deteksi menggunakan PHP Shell Detector

Pengujian akurasi dalam mendeteksi *backdoor shell* pada *web server* dilakukan terhadap 12 sampel *backdoor shell* dengan total *file* yang dianalisis 3964 *file*. Pengujian dilakukan dengan menghitung nilai *precision* dan *recall* [16] berdasarkan total *backdoor shell*.

A. *Hasil Pengujian Deteksi Menggunakan Aplikasi yang Dibangun*

Pengujian akurasi dilakukan pada *web server* yang memiliki total *file* 3964 *file* dengan kondisi 12 sampel *backdoor shell* telah disisipkan di dalam *web server*. Hasil pendeteksian 12 *file* yang telah disisipkan tersebut dapat dilihat pada Tabel 1/

Tabel 1. Hasil Pendeteksian Oleh Sistem

No	Backdoor Shell	Hasil Deteksi
1	BoffMax	Dicurigai
2	C102	Positif
3	CYBERDARK	Positif
4	Dhanush	Positif
5	EgY_SpIdEr	Positif
6	G00nshell	Dicurigai
7	K2ll33d	Positif
8	K3c4	Positif
9	Fi73 M4n4g3r	Negatif
10	Madspotshellscript	Positif
11	SpyShell	Positif
12	Storm7Shell	Positif

Dari hasil pengujian diperoleh data sebagai berikut: total *file* yang ditemukan sebanyak 12 *file*. *File* yang positif teridentifikasi sebagai *backdoor shell* sebanyak 9 *file*. *File* yang dicurigai sebagai *backdoor shell* sejumlah 2 *file*. *File* yang negatif atau tidak

terdeteksi sebagai *backdoor* terdapat 1 *file*. Berdasarkan Tabel 1, perhitungan akurasi dalam mengidentifikasi *file backdoor shell* dapat dilihat pada rumus (1) sebagai berikut.

$$Akurasi = \frac{Total\ file\ backdoor\ terdeteksi}{Total\ file\ backdoor\ yang\ disisipkan} \quad (1)$$

$$Akurasi = \frac{9}{12} \times 100\% = 75\%$$

Selain akurasi, pengujian juga dilakukan untuk menghitung *precision* dan *recall*. Pengujian *precision* merupakan cara untuk mengetahui proporsi kasus *file* yang positif *backdoor* yang diprediksi yang benar-benar *backdoor*. Pengujian *recall* mencerminkan berapa banyak kasus yang relevan dengan *file* yang positif *backdoor*. Adapun hasil pengujian dapat dilihat pada Tabel 2 berikut.

Tabel 2. Pengujian *Precision* dan *Recall*

N : 3964		***Predicted Backdoor***	
		No	***Yes***
Actual Backdoor	***No***	3946	6
	Yes	1	11

Pada Tabel 2, hasil pengujian menunjukkan bahwa dari total 3964 *file*, terdapat 6 *file* yang sebenarnya (*Actual No*) bukan *backdoor* tetapi terdeteksi sebagai *backdoor* (*Predicted Yes*). Dari 12 *file backdoor* (*Actual Yes*), 1 *file* tidak dikenali sebagai *backdoor* (*Predicted No*) dan 11 sisanya dideteksi sebagai *backdoor* (*Predicted Yes*). Berdasarkan Tabel 2, nilai *precision* dan *recall* dapat dilihat pada rumus (2) adalah sebagai berikut.

$$Precision = \frac{TP}{(TP + FP)} \quad (2)$$

$$Precision = \frac{11}{11 + 6} = \frac{11}{17} = 0{,}64$$

$$Recall = \frac{TP}{(TP + FN)} \quad (3)$$

$$Recall = \frac{11}{11 + 1} = \frac{11}{12} = 0{,}91$$

Keterangan:

TP : *True Prositive*

FN : *False Negative*

FP : *False Positif*

B. Hasil Pengujian Deteksi Backdoor Menggunakan PHP Shell Detector

Pengujian sistem juga dilakukan dengan membandingkan hasil pendeteksian *backdoor* menggunakan PHP Shell Detector. Total *file* dalam sistem berjumlah 3964 dan 12 *file backdoor* yang disisipkan. Berdasarkan hasil pengujian diperoleh data sebagai berikut: total *file* yang ditemukan sebanyak 12 *file*. *File* yang positif teridentifikasi sebagai *backdoor shell* sejumlah 4 *file*. *File* yang dicurigai sebagai *backdoor shell* terdapat 6 *file*. *File* yang negatif atau tidak terdeteksi sejumlah 2 *file*. Hasil pengujian sistem dengan menggunakan PHP Shell Detector dapat dilihat pada Tabel 3 berikut.

Tabel 3. Hasil Pengujian Pada PHP *Shell Detector*

No	Backdoor Shell	Hasil Deteksi
1	BoffMax	Dicurigai
2	C102	Dicurigai
3	CYBERDARK	Dicurigai
4	Dhanush	Positif
5	EgY_SpIdEr	Positif
6	G00nshell	Dicurigai
7	K2ll33d	Negatif
8	K3c4	Positif
9	Fi73 M4n4g3r	Negatif
10	Madspotshellscript	Dicurigai
11	SpyShell	Dicurigai
12	Storm7Shell	Positif

Berdasarkan Tabel 3, akurasi dalam mengidentifikasi *file backdoor shell* dengan PHP Shell Detector menggunakan rumus (1) dapat dilihat sebagai berikut.

$$Akurasi = \frac{4}{12} \times 100\% = 33{,}3\%$$

Hasil pengujian *precision* dan *recall* pada sistem yang diuji menggunakan PHP Shell Detector dapat dilihat pada Tabel 4 berikut ini.

Tabel 4. Pengujian *Precision* dan *Recall*

N : 3964		***Predicted Backdoor***	
		No	***Yes***
Actual Backdoor	***No***	3842	110
	Yes	2	10

Tabel 4 menunjukkan bahwa terdapat 110 *file* bukan *backdoor* (*Actual No*) tetapi terdeteksi sebagai *backdoor* (*Predicted Yes*). Dari 12 *file backdoor* (*Actual Yes*), 2 *file* tidak dikenali sebagai *backdoor* (*Predicted No*) dan 10 sisanya dideteksi sebagai *backdoor* (*Predicted Yes*). Nilai *precision* dan *recall* berdasarkan Tabel 4 menggunakan rumus (2) dan (3) adalah sebagai berikut.

$$Precision = \frac{10}{10 + 110} = \frac{10}{120} = 0{,}083$$

$$Recall = \frac{10}{10 + 12} = \frac{10}{22} = 0{,}83$$

Berdasarkan pengujian akurasi yang telah dilakukan terhadap kedua aplikasi, dapat disimpulkan bahwa aplikasi yang dibangun memiliki akurasi yang lebih baik dibandingkan dengan menggunakan aplikasi PHP Shell Detector. Selain itu, jumlah hasil deteksi yang dicurigai dari sistem yang dibangun lebih sedikit dibandingkan dengan aplikasi PHP Shell Detector sehingga lebih efisien jika *administrator website* atau pemilik *website* ingin menganalisis lebih mendalam.

IV. PEMBAHASAN

Pada bagian ini menggambarkan proses yang dilalui sebuah sampel data sehingga dapat diketahui apakah *source code* tersebut termasuk *backdoor* atau bukan. Tahap awal pada proses ini, sistem mengambil *source code* pada *web server* kemudian dilakukan transformasi *source code* secara keseluruhan. Contoh berikut adalah *file worse.php* yang akan dianalisis dapat dilihat pada Gambar 4.

```
print "<form method=post
enctype=\"multipart/form-data\"><table>";
print "<tr><td><b>Execute
command:</b></td><td><input size=100
name=\"_cmd\"
value=\"".$currentCMD."\"></td>";
print "<td><input type=submit name=_act
value=\"Execute!\"></td></tr>";
print "<tr><td><b>Change
directory:</b></td><td><input size=100
name=\"_cwd\" value=\"".$currentWD."\"></td>";
print "<td><input type=submit name=_act
value=\"List files!\"></td></tr>";
```

Gambar 4. Kutipan *Source Code* PHP

Pada *source code* tersebut dilakukan transformasi data menggunakan fungsi *base64_encode*. Tujuan transformasi ini adalah untuk mengubah data sehingga data tersebut dapat digunakan oleh sistem lainnya secara benar dan aman, misalkan dapat dikirim via *e-mail* ataupun ditampilkan di halaman *website* [15]. Gambar 5 adalah contoh kutipan *file worse.php* yang telah ditransformasikan.

```
cHJpbnQgIjxmb3JtIG1ldGhvZD1wb3N0IGVuY3R5c
GU9XCJtdWx0aXBhcnQvZm9ybS1kYXRhXCI+PHRhYm
xlPiI7CnByaW50ICI8dHI+PHRkPjxiPkV4ZWN1dGU
gY29tbWFuZDo8L2I+PC90ZD48dGQ+PGluCHV0IHNp
emU9MTAwIG5hbWU9XCJfY21kXCIgdmFsdWU9XCIiL
iRjdXJyZW50Q01ELiJcIj48L3RkPiI7CnByaW50IC
I8dGQ+PGluCHV0IHR5cGU9c3VibWl0IG5hbWU9X2F
jdCB2YWx1ZT1cIkV4ZWN1dGUhXCI+PC90ZD48L3Ry
PiI7CnByaW50ICI8dHI+PHRkPjxiPkNoYW5nZSBka
XJlY3RvcnK6PC9iPjwvdGQ+PHRkPjxpbnB1dCBzaX
plPTEwMCBuYW1lPVwiX2N3ZFwiIHZhbHV1PVwiIi4
kY3VycmVudFdELiJcIj48L3RkPiI7CnByaW50ICI8
dGQ+PGluCHV0IHR5cGU9c3VibWl0IG5hbWU9X2Fjd
CB2YWx1ZT1cIkxpc3QgZmlsZXMhXCI+PC90ZD48L3
RyPiI7Cg
```

Gambar 5. Kutipan Hasil Transformasi *Worse.Php*

Proses selanjutnya adalah sistem akan melakukan *request signature backdoor* pada *database* yang berupa *string* hasil dari *encode* dengan fungsi *base64_encode*. Berikut adalah Gambar 6 yang merupakan contoh *signature backdoor*. Fungsi tersebut merupakan *signature* milik *backdoor shell* "*worse*". Setiap data *signature* yang disimpan dalam *database* memiliki jumlah karakter sebanyak 64 digit.

```
IG5hbWU9XCJfY21kXCIgdmFsdWU9XCIiLiRjdXJyZ
W50Q01ELiJcIj48L3RkPiI7
```

Gambar 6. *Backdoor Shell Signature*

Setelah data *backdoor signature* diambil maka akan dilakukan pencocokan *string backdoor signature* terhadap *source code* yang telah ditransformasikan sebelumnya. Gambar 7 berikut adalah contoh hasil temuan.

```
cHJpbnQgIjxmb3JtIG1ldGhvZD1wb3N0IGVuY3R5cGU9X
CJtdWx0aXBhcnQvZm9ybS1kYXRhXCI+PHRhYmxlPiI7Cn
ByaW50ICI8dHI+PHRkPjxiPkV4ZWN1dGUgY29tbWFuZDo
8L2I+PC90ZD48dGQ+PGluCHV0IHNpemU9MTAwIG5hbWU9
XCJfY21kXCIgdmFsdWU9XCIiLiRjdXJyZW50Q01ELiJcI
j48L3RkPiI7CnByaW50ICI8dGQ+PGluCHV0IHR5cGU9c3
VibWl0IG5hbWU9X2FjdCB2YWx1ZT1cIkV4ZWN1dGUhXCI
+PC90ZD48L3RyPiI7CnByaW50ICI8dHI+PHRkPjxiPkNo
YW5nZSBkaXJlY3RvcnK6PC9iPjwvdGQ+PHRkPjxpbnB1d
CBzaXplPTEwMCBuYW1lPVwiX2N3ZFwiIHZhbHV1PVwiIi
4kY3VycmVudFdELiJcIj48L3RkPiI7CnByaW50ICI8dGQ
+PGluCHV0IHR5cGU9c3VibWl0IG5hbWU9X2FjdCB2YWx1
ZT1cIkxpc3QgZmlsZXMhXCI+PC90ZD48L3RyPiI7Cg
```

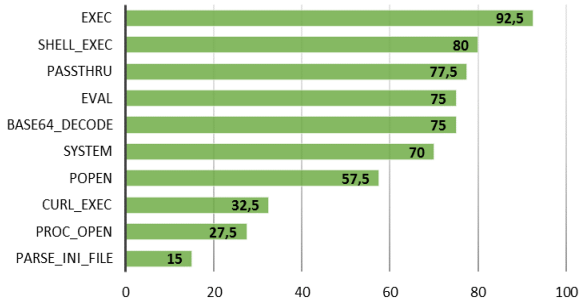
Gambar 7. Ditemukan Hasil *String Matching*

Dari hasil pencarian ditemukan kesamaan identic pada *string* terhadap *backdoor signature* pada *file worse.php* pada bagian yang diberi *highlight*. Berdasarkan hal tersebut, maka dengan demikian *file worse.php* positif sebagai *backdoor* dan akan diberikan *flag* merah pada hasil pencarian. Apabila hasil *string matching* tidak ditemukan *backdoor signature* maka *script* diberi *flag* hijau yang kemudian dilanjutkan dengan proses *taint analysis*.

Sebelum proses *taint analysis* dilakukan, ada beberapa hal yang dilakukan terlebih dahulu. Dalam penelitian ini dilakukan analisis terhadap beberapa sampel *backdoor shell* yang mana hasil analisis tersebut diterapkan dalam perhitungan dalam proses *taint analysis*.

Analisis dilakukan terhadap 40 *file backdoor shell* kemudian dilakukan perhitungan probabilistas kemunculan terhadap fungsi-fungsi seperti: *exec*, *system*, *shell exec*, *passthru*, *eval*, *base64 decode*, *proc_open*, *popen*, *curl_exec*, dan *parse_ini_file*. Hasil perhitungan tersebut menentukan fungsi-fungsi yang identic sebagai *backdoor shell* dan juga sebagai parameter untuk melakukan penyaringan terhadap *file-file* yang bukan *backdoor shell* berdasarkan kombinasi pada fungsi-fungsi tersebut.

Perhitungan peluang kemunculan fungsi-fungsi tersebut diperoleh hasil dalam bentuk persentase pada Gambar 8. Berdasarkan dari hasil perhitungan tersebut, kemudian ditentukan fungsi yang memiliki kemungkinan muncul lebih atau sama dengan 75%, yaitu sebanyak 5 fungsi sebagai fungsi yang identik sebagai *backdoor shell*. Dengan demikian, poin yang diberikan kepada 10 fungsi tersebut seperti dalam Tabel 5.



Gambar 8. Peluang Kemunculan Fungsi

Tabel 5. Poin Pada Fungsi

No	Fungsi	Poin
1	Eval	2
2	base64_decode	2
3	shell_exec	2
4	exec	2
5	passthru	2
6	proc_open	1
7	popen	1
8	system	1
9	curl_exec	1
10	Parse_ini_file	1

Poin pada Tabel 5 akan diakumulasikan berdasarkan kombinasi fungsi yang dimiliki oleh *file* yang sedang dianalisis. Dari hasil akumulasi poin, diperoleh data kombinasi poin terendah, yaitu sebesar 4 poin, untuk nilai kombinasi rata-rata poin 10,25, dan untuk kombinasi poin tertinggi, yaitu 14 poin. Berdasarkan hasil analisis tersebut, dapat dijadikan Batasan dalam menentukan kombinasi poin pada *file* yang positif sebagai *backdoor shell* dan yang tidak maka diasumsikan pada Tabel 6 berikut.

Tabel 6. Pembagian Jenis *File*

Total Poin	Flag	Ket
< 4	Putih	Bersih
4 – 10	Kuning	Hati-Hati
> 10	Merah	Berbahaya

Berikut ini adalah keterangan dari 3 jenis *file* pada Tabel 6:

- *File* dengan *flag* putih merupakan *file* yang bebas atau tidak terdeteksi sebagai *backdoor shell*.
- *File* dengan *flag* kuning merupakan jenis *file* yang masih memiliki kemungkinan sebagai *backdoor shell* atau tidak.
- *File* dengan *flag* merah merupakan jenis *file* yang positif diidentifikasi sebagai *backdoor shell*.

Dalam proses *taint analysis*, dilakukan identifikasi terhadap 10 fungsi yang telah dipaparkan sebelumnya. Setelah sistem menemukan fungsi yang terdapat pada *source code* kemudian dilakukan akumulasi poin dengan aturan poin pada Tabel 5. Ketika sistem telah melakukan kalkulasi poin fungsi maka tahap selanjutnya akan dilakukan pengelompokan *file* yang dimana aturan pengeleompokan *file* mengikuti Tabel 6 tentang pembagian jenis *file*. Setelah dilakukan pengelompokan jenis *file* maka sistem dapat menampilkan hasil pencarian kepada *user* dengan menampilkan data *file* yang memiliki *flag* merah dan kuning.

Pada tahap awal, sistem kan melakukan *scanning* fungsi pada *source code file*. Proses ini bertujuan untuk memperoleh hasil dari *scanning* terhadap 10 fungsi yang telah dijelaskan sebelumnya. Dalam sebuah *source code* terdeteksi beberapa fungsi, seperti pada Gambar 9 adalah contoh hasil deteksi fungsi *eval*().

```
case "GZINFLATE":
$res_t=base64_encode(gzdeflate(trim(stri
pslashes($c_ntent.''),'<?php, ?>'),9));
$res_t="<?php /* Encoded in INDRAJITH
SHELL PROJECT */
eval(gzinflate(base64_decode("$res_t")))
; ?>";
break;
```

Gambar 9. Contoh Fungsi *eval*()

Contoh fungsi *base64_decode*() terdeteksi di beberapa *line*, salah satunya pada 1110 dapat dilihat pada Gambar 10.

```
switch($typ_d)
{
    case "GZINFLATE":
    $res_t=gzinflate(base64_decode($c_nt
ent));
```

Gambar 10. Contoh Fungsi *base64_decode*()

Pada Gambar 11 adalah contoh fungsi *system*() yang terdeteksi pada *line* 728.

```
else if(function_exists('system')){
 ob_start();
 system($cmd);
 $res = ob_get_contents();
 ob_end_clean();     }
```

Gambar 11. Contoh Fungsi *system*()

Contoh fungsi *shell_exec*() terdeteksi di *line* 718. Dapat dilihat Gambar 12 berikut.

```
if(function_exists('shell_exec')) {
 $res=shell_exec($cmd);}
```

Gambar 12. Contoh Fungsi *shell_exec*()

Fungsi *exec*() terdeteksi di *line* 722 dapat dilihat pada Gambar 13 berikut.

```
else if(function_exists('exec')) {
 exec($cmd,$res);
 $res=join("n",$res); }
```

Gambar 13. Contoh Fungsi *exec*()

Contoh fungsi *passthru*() terdeteksi di *line* 735 pada Gambar 14.

```
elseif(function_exists('passthru')){
        ob_start();
        passthru($cmd);
        $res=ob_get_contents();
        ob_end_clean();  }
```

Gambar 14. Contoh Fungsi *passthru*()

Pada Gambar 15 adalah contoh fungsi *proc_open*() terdeteksi di *line* 742.

```
else if(function_exists('proc_open')){
     $descriptorspec = array(0 =>
array("pipe", "r"),1 =>array("pipe",
     "w"),  2 => array("pipe", "w"));
     $handle = proc_open($cmd
,$descriptorspec , $pipes);
        if(is_resource($handle))
```

Gambar 15. Contoh Fungsi *proc_open*()

Contoh fungsi *popen*() terdeteksi di *line* 765 dapat dilihat pada Gambar 16 berikut.

```
else if(function_exists('popen')){
        $handle = popen($cmd , "r");
        if(is_resource($handle)){
            if(function_exists('fread')
&& function_exists('feof'))
```

Gambar 16. Contoh Fungsi *popen*()

Contoh fungsi *curl_exec*() terdeteksi di *line* 778 dapat dilihat pada Gambar 17 berikut.

```
$ch = curl_init($urlzzx);
   curl_setopt($ch, CURLOPT_NOBODY,
true);
   curl_exec($ch);
   $status_code=curl_getinfo($ch,
CURLINFO_HTTP_CODE);
```

Gambar 17. Contoh Fungsi *curl_exec*()

Setelah sistem melakukan *scanning* pada seluruh baris *source code*, hasilnya ternyata dapat dideteksi *file* tersebut memiliki 9 fungsi *backdoor*, yaitu *eval*(), *base64_decode*(), *system*(), *shell_exec*(), *exec*(), *passthru*(), *proc_open*(), *popen*(), dan *curl_exec*(). Hasil *scanning* tersebut kemudian dihitung poin yang dimiliki *file* tersebut berdasarkan Tabel 5. Masing-masing nilai fungsi dijumlahkan kemudian diklasifikasi berdasarkan ketentuan pada Tabel 6. Total nilai untuk *file* tersebut dapat dilihat pada Tabel 7.

Berdasarkan akumulasi poin di Tabel 7, maka *file* pada contoh ditandai dengan *flag* merah dan dikategorikan sebagai *backdoor* karena nilai akumulasi lebih dari 10.

Tabel 7. Hasil Perhitungan Fungsi

Fungsi	Poin
Eval	2
Base64_decode	2
System	1
Shell_exec	2
Exec	2
Passthru	2
Proc_open	1
Popen	1
Curl_exec	1
Total	**14**

V. PENUTUP

A. Kesimpulan

Hasil yang didapat dari pengujian yang dilakukan berdasarkan proses dalam penelitian ini serta mengacu pada tujuan penelitian, maka dapat disimpulkan bahwa:

1. Sistem yang dibangun memiliki akurasi yang lebih baik dibandingkan dengan menggunakan aplikasi PHP Shell Detector. Hasil pengujian menunjukkan pendeteksian *backdoor* menggunakan PHP Shell Detector memiliki akurasi 33%, sedangkan pendeteksian dengan sistem yang dibuat menunjukkan akurasi sebesar 75%.
2. Jumlah hasil deteksi yang dicurigai dari sistem yang dibangun lebih sedikit, yaitu 12 *file* dibandingkan dengan aplikasi PHP Shell Detector sejumlah 120 *file* sehingga lebih efisien jika *administrator website* atau pemilik *website* ingin menganalisis lebih mendalam.

B. Saran

Berdasarkan hasil yang telah dicapai saat ini, bahwa sistem pendeteksi *backdoor shell* pada *web server* ini mempunyai beberapa hal yang disarankan untuk menambahkan hal-hal yang dapat melengkapi sistem ini di waktu yang akan dating, yaitu:

1. Perlunya ditambah fitur untuk mengelola hasil pendeteksi, seperti mengeksekusi *file* langsung dan melihat *source code* dari aplikasi sehingga mempercepat proses pengecekan oleh *administrator website*.
2. Penambahan jenis *script* yang dapat dianalisis dengan mengunakan *taint analysis*.

DAFTAR PUSTAKA

[1] D. Watson, "Web Application Attacks," *Netw. Secur.*, vol. 2007, no. 10, pp. 10–14.

[2] M. Chawki, A. Darwish, M. A. Khan, and S. Tyagi, "Cybercrime, Digital Forensics and Jurisdiction," in *Studies in Computational Intelligence*, vol. 593, 2015, pp. 97–111.

[3] E. Zam, *Buku Sakti Hacker*. Jakarta: Mediakita, 2011.

[4] S. Agisilaos, "Detecting malicious code in a web server "," University of Piraeus, 2016.

[5] E. Setiyawan, J. Atmaji, and B. M. Susanto, "Monitoring Keamanan Jaringan Komputer Menggunakan Network Intrussion Detection System (NIDS)," pp. 118–122, 2016.

[6] Popstojanova, Katerina Goseva and A. Perhinschi, "On the capability of static code analysis to detect security vulnerabilities," *Inf. Softw. Technol.*, vol. 68, pp. 18–33, 2015.

[7] J. Lee, E. Kim, J. Yoo, and J. Lee, *A Preliminary Report on Static Analysis of C Code for Nuclear Reactor Protection System*, vol. 61131, no. 9. IFAC, 2013.

[8] A. Ouadjaout, A. Miné, N. Lasla, and N. Badache, "Static analysis by abstract interpretation of functional properties of device drivers in TinyOS," *J. Syst. Softw.*, vol. 120, pp. 114–132, 2016.

[9] J. P. Jyotiyana and A. Mishra, "Secure Authentication: Eliminating Possible Backdoors in Client-Server Endorsement," *Procedia Comput. Sci.*, vol. 85, no. Cms, pp. 606–615, 2016.

[10] P. Seshagiri, A. Vazhayil, and P. Sriram, "AMA: Static Code Analysis of Web Page for the Detection of Malicious Scripts," *Procedia Comput. Sci.*, vol. 93, no. September, pp. 768–773, 2016.

[11] O. Udrea, C. Lumezanu, and J. S. Foster, "Rule-based static analysis of network protocol implementations," *Inf. Comput.*, vol. 206, no. 2–4, pp. 130–157, 2008.

[12] A. Miné, "Static Analysis of Embedded Real-Time Concurrent Software with Dynamic Priorities," *Electron. Notes Theor. Comput. Sci.*, vol. 331, pp. 3–39, 2017.

[13] D. Oktavianto, *Ethical Hacking and Computer Security*. Bandung: Training Nixtrain, 2015.

[14] T. Dinh, C. Guang, G. Xiaojun, and W. Pan, "'Webshell detection Technique in web application,'" in *Computing, Communication and Networking Technologies (ICCCNT)*, 2014.

[15] P. Luczko, "PHP Shell Detector." [Online]. Available: https://github.com/emposa/PHP-Shell-Detector. [Accessed: 28-Aug-2015].

[16] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation," Adelaide, 2007.