

**PENGEMBANGAN KAKAS *STATIC CODE ANALYSIS* DENGAN  
METODE *TAINT ANALYSIS* UNTUK DETEKSI KERENTANAN  
PADA APLIKASI WEB**

**Laporan Tugas Akhir**

**Disusun sebagai syarat kelulusan tingkat sarjana**

**Oleh**

**ACHMAD FAHRURROZI MASKUR**

**NIM : 13515026**



**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
September 2019**

**PENGEMBANGAN KAKAS *STATIC CODE ANALYSIS*  
DENGAN METODE *TAINT ANALYSIS* UNTUK DETEKSI  
KERENTANAN PADA APLIKASI WEB**

**Laporan Tugas Akhir**

**Oleh**

**ACHMAD FAHRURROZI MASKUR**

**NIM : 13515026**

**Program Studi Teknik Informatika**

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Telah disetujui dan disahkan sebagai Laporan Tugas Akhir  
di Bandung, pada tanggal 27 September 2019

Pembimbing,

Yudistira Dwi Wardhana Asnar S.T., Ph.D.

NIP 19800827 201504 1 002

## **LEMBAR PERNYATAAN**

Dengan ini saya menyatakan bahwa:

1. Pengerjaan dan penulisan Laporan Tugas Akhir ini dilakukan tanpa menggunakan bantuan yang tidak dibenarkan.
2. Segala bentuk kutipan dan acuan terhadap tulisan orang lain yang digunakan di dalam penyusunan laporan tugas akhir ini telah dituliskan dengan baik dan benar.
3. Laporan Tugas Akhir ini belum pernah diajukan pada program pendidikan di perguruan tinggi mana pun.

Jika terbukti melanggar hal-hal di atas, saya bersedia dikenakan sanksi sesuai dengan Peraturan Akademik dan Kemahasiswaan Institut Teknologi Bandung bagian Penegakan Norma Akademik dan Kemahasiswaan khususnya Pasal 2.1 dan Pasal 2.2.

Bandung, 27 September 2019

Achmad Fahrurrozi Maskur

NIM 13515026

## ABSTRAK

# PENGEMBANGAN KAKAS *STATIC CODE ANALYSIS* DENGAN METODE *TAINT ANALYSIS* UNTUK DETEKSI KERENTANAN PADA APLIKASI WEB

Oleh

Achmad Fahrurrozi Maskur

NIM : 13515026

Dewasa ini teknologi informasi dan komunikasi berkembang dengan sangat cepat. Seiring dengan perkembangan yang cepat, maka semakin banyak pula orang yang tertarik untuk menjadi seorang *software engineer*. Namun dari banyaknya *software engineer* tersebut, tidak banyak yang sadar akan pentingnya kualitas dari sebuah *code* khususnya untuk aspek keamanan. Dibutuhkan sebuah solusi yang dapat meminimalisir hal tersebut seperti memanfaatkan teknologi pemindai *code* secara otomatis atau yang sering disebut dengan *static code analysis*. Sehingga pada Tugas Akhir ini dirumuskan sebuah rumusan masalah yaitu bagaimana meminimalisir terjadinya *vulnerability* dengan memanfaatkan teknologi *static code analysis*.

Tujuan dari Tugas Akhir ini yaitu membangun sebuah kakas yang berfungsi untuk melakukan *static code analysis*. *Static code analysis* dilakukan dengan metode *Taint Analysis* yaitu dengan mengidentifikasi *variable-variable* yang dicurigai berbahaya (*tainted*), dikarenakan berasal dari masukan pengguna. Kemudian dilakukan penelusuran terhadap *variable* tersebut terhadap fungsi yang berbahaya yang kemudian disebut sebagai *sink*. Jika *tainted variable* tersebut masuk ke dalam *sink* sebelum dilakukan *filter* atau *sanitize*, maka dianggap sebagai *vulnerability*.

Evaluasi terhadap kakas yang dibangun menunjukkan hasil yang memuaskan dalam menemukan *vulnerability*. Dari 20 *open source project* yang terdaftar pada situs resmi CVE, 12 diantaranya berhasil ditemukan atau sebanyak 60%. Jenis *vulnerability* yang dapat ditemukan oleh metode *taint analysis* yaitu *vulnerability* bertipe *injection*. Selain itu, semua kebutuhan fungsional kakas juga sudah terpenuhi.

Kata kunci: *static code analysis, taint analysis, web vulnerability*.

## KATA PENGANTAR

Alhamdulillah rabbil alamin, segala puji dipanjatkan kepada Allah SWT, karena atas rahmat dan hidayat-Nya penyusunan laporan Tugas Akhir ini dapat penulis selesaikan. Tidak lupa kita panjatkan salam serta shalawat kepada Nabi Muhammad SAW. Selain itu, laporan ini juga dapat diselesaikan berkat bantuan dari banyak pihak, penulis mengucapkan terima kasih kepada:

1. Bapak Yudistira Dwi Wardhana Asnar, S.T., Ph.D. sebagai dosen pembimbing atas ilmu, bimbingan, dan arahan yang diberikan selama proses pengerjaan Tugas Akhir.
2. Bapak Achmad Imam Kistijantoro, S.T., M.Sc., Ph.D. sebagai dosen penguji atas kritik dan saran yang diberikan dalam proses pengerjaan Tugas Akhir.
3. Ibu Dr. Fazat Nur Azizah, S.T., M.T. dan Bapak Nugraha Priya Utama, S.T., M.A, Ph.D. yang telah mengarahkan proses pengerjaan Tugas Akhir dalam kelas kuliah.
4. Seluruh Bapak dan Ibu dosen pengajar program studi Teknik Informatika ITB atas ilmu yang diberikan dalam proses belajar selama kuliah di ITB.
5. Seluruh staff program studi Teknik Informatika ITB atas bantuan selama kuliah serta proses administrasi Tugas Akhir.
6. Bapak Ir. Imam Maskur, Ibu Andi Idayanti, Mas Imanullah Inoviyat Maskur, dan Muhammad Arif serta keluarga yang memberikan motivasi, doa, dukungan, dan kasih sayang.
7. Kevin Jonathan Koswara, Lazuardi Firdaus, Vincent Hendryanto Halim, Aditya Pratama, Richard Matthew, Mahdiar Naufal sebagai teman satu bimbingan dalam mengingatkan pengerjaan Tugas Akhir.
8. Diki Ardian Wirasandi, Radiyya Dwisaputra, David Theosaksomo, Mokhammad Ferdi Ghozali, Akmal Fadlurohman, Muhammad Umar Fariz Tumbuan, Husnulzaki Wibisono Haryadi, Pratamamia Agung Prihatmaja

yang menemani masa perkuliahan di program studi Teknik Informatika ITB.

9. Hisham Lazuardi Yusuf yang memberi bantuan kosan serta menyemangati dan membantu dalam proses pengerjaan Tugas Akhir.
10. Teman-teman Unit Kesenian Sulawesi Selatan, Tim KRSBI Humanoid URO ITB, Sparta HMIF 2016, Arkavida 5.0, Komunitas CTF HMIF, serta teman-teman HMIF ITB yang telah mengisi hari-hari penulis selama berkuliah di ITB
11. Mas dan Mbak divisi Information Security Bukalapak yang memberi bantuan dan semangat dalam pengerjaan Tugas Akhir.
12. Pihak-pihak lain yang tidak bisa disebutkan semua yang telah membantu secara langsung maupun tidak langsung.

Akhir kata, semoga penulisan laporan Tugas Akhir ini dapat berguna bagi bangsa dan negara khususnya pribadi penulis di kemudian hari. Penulis sangat terbuka untuk menerima kritik dan saran yang dapat digunakan dalam pengerjaan lebih lanjut untuk Tugas Akhir ini.

Bandung, 27 September 2019

Penulis

## DAFTAR ISI

<b>ABSTRAK .....</b>	<b>iv</b>
<b>KATA PENGANTAR.....</b>	<b>v</b>
<b>DAFTAR ISI.....</b>	<b>vii</b>
<b>DAFTAR LAMPIRAN .....</b>	<b>ix</b>
<b>DAFTAR GAMBAR.....</b>	<b>x</b>
<b>DAFTAR TABEL .....</b>	<b>xii</b>
<b>BAB I PENDAHULUAN.....</b>	<b>1</b>
I.1    Latar Belakang.....	1
I.2    Rumusan Masalah.....	2
I.3    Tujuan .....	2
I.4    Batasan Masalah .....	2
I.5    Metodologi.....	3
I.6    Sistematika Pembahasan.....	4
<b>BAB II STUDI LITERATUR .....</b>	<b>6</b>
II.1 <i>Web Application Security</i> .....	6
II.1.1 <i>Web Application</i> .....	6
II.1.2    Web Vulnerability .....	8
II.2 <i>Static Code Analysis</i> .....	11
II.2.1    Pendekatan pada <i>Static Code Analysis</i> .....	11
II.2.2 <i>Static Code Analysis Tools</i> .....	13
<b>BAB III ANALISIS DAN RANCANGAN .....</b>	<b>16</b>
III.1    Kemampuan <i>Taint Analysis</i> dalam Menemukan Celah Keamanan .....	16

III.2	Analisis Pemrosesan <i>Source Code</i> .....	19
III.2.1	<i>Preprocessing</i> : Parsing dan Pembuatan <i>Abstract Syntax Tree</i> .....	19
III.2.2	Pemrosesan untuk tiap <i>node</i> pada <i>Abstract Syntax Tree</i> .....	20
III.3	Evaluasi Metode <i>Taint Analysis</i> .....	28
III.4	Analisis Kebutuhan .....	29
III.5	Analisis Arsitektur Kakas.....	31
<b>BAB IV IMPLEMENTASI SOLUSI DAN PENGUJIAN.....</b>		<b>33</b>
IV.1	Implementasi Solusi .....	33
IV.2	Implementasi Kebutuhan.....	37
IV.3	Pengujian .....	42
IV.3.1	Tujuan Pengujian .....	42
IV.3.2	Lingkungan Pengujian .....	42
IV.3.3	Pengujian <i>Taint Analysis</i> .....	43
IV.3.4	Kesimpulan Hasil Pengujian.....	44
<b>BAB V KESIMPULAN DAN SARAN .....</b>		<b>45</b>
V.1	Kesimpulan .....	45
V.2	Saran .....	46
<b>DAFTAR REFERENSI .....</b>		<b>48</b>



## DAFTAR LAMPIRAN

<b>Lampiran A. Daftar <i>Knowledge Base</i> .....</b>	<b>49</b>
A.1 Daftar <i>Sources</i> .....	49
A.2 Daftar Sanitizers .....	51
A.3 Daftar <i>Sinks</i> .....	53
<b>Lampiran B. Daftar <i>Source Code</i> .....</b>	<b>56</b>
B.1 <i>Source code</i> x_xss.php.....	56
B.2 <i>Source code</i> x_switch.php .....	57
B.3 <i>Source code</i> x_sql.php .....	58
B.4 <i>Source code</i> x_os.php .....	59
B.5 <i>Source code</i> x_if_elseif_else.php .....	59
B.6 <i>Source code</i> x_func_funccall.php.....	60
B.7 <i>Source code</i> x_assignment.php.....	63
B.8 <i>Source code</i> x_for_foreach.php .....	64

## DAFTAR GAMBAR

Gambar II.1 <i>Workflow</i> ketika pengguna mengunjungi situs. ....	7
Gambar II.2 Arsitektur Dytan: Dynamic Taint Analysis Framework.....	12
Gambar III.1 Contoh kode yang terdapat <i>vulnerability</i> .....	17
Gambar III.2 Contoh kode yang aman .....	17
Gambar III.3 <i>State</i> diagram <i>taint analysis</i> .....	18
Gambar III.4 Contoh <i>source code</i> dengan tipe <i>node</i> InlineHTML.....	21
Gambar III.5 Contoh <i>source code</i> dengan tipe <i>node</i> Assignment .....	21
Gambar III.6 Contoh Assignment dengan status <i>tainted</i> .....	22
Gambar III.7 Contoh <i>variable</i> yang berbahaya.....	22
Gambar III.8 Contoh Assignment dengan status <i>sanitized</i> .....	23
Gambar III.9 Contoh fungsi-fungsi yang termasuk <i>sanitizers</i> .....	23
Gambar III.10 Contoh FunctionCall yang termasuk <i>sink</i> .....	24
Gambar III.11 Contoh fungsi-fungsi yang termasuk <i>sink</i> .....	24
Gambar III.12 Contoh <i>source code</i> dengan tipe <i>node</i> Include.....	25
Gambar III.13 Contoh <i>source code</i> dengan tipe <i>node</i> Function .....	25
Gambar III.14 Contoh <i>source code</i> dengan tipe <i>node</i> Global.....	26
Gambar III.15 Contoh <i>source code</i> dengan tipe <i>node</i> Return.....	26
Gambar III.16 Contoh <i>source code</i> dengan tipe <i>node</i> If.....	27
Gambar III.17 Contoh <i>source code</i> dengan tipe <i>node</i> Switch .....	27
Gambar III.18 Contoh <i>source code</i> dengan tipe <i>node</i> For .....	28
Gambar III.19 Contoh <i>source code</i> dengan tipe <i>node</i> Foreach.....	28
Gambar III.20 Diagram <i>use case</i> kebutuhan fungsional sistem.....	30

Gambar III.21 Arsitektur kakas <i>static code analysis</i> .....	32
Gambar IV.1 Implementasi arsitektur kakas dan teknologi yang digunakan .....	33
Gambar IV.2 Potongan kode untuk pemrosesan tiap <i>node</i> .....	34
Gambar IV.3 <i>Pseudocode</i> untuk pemrosesan <i>node</i> Assignment .....	35
Gambar IV.4 <i>Pseudocode</i> untuk pemrosesan <i>node</i> FunctionCall.....	35
Gambar IV.5 <i>Pseudocode</i> untuk pemrosesan <i>node</i> Function .....	35
Gambar IV.6 <i>Pseudocode</i> untuk pemrosesan <i>node</i> Include.....	36
Gambar IV.7 <i>Pseudocode</i> untuk pemrosesan <i>node</i> Global.....	36
Gambar IV.8 <i>Pseudocode</i> untuk pemrosesan <i>node</i> Return.....	36
Gambar IV.9 <i>Pseudocode</i> untuk pemrosesan <i>node</i> If .....	36
Gambar IV.10 <i>Pseudocode</i> untuk pemrosesan <i>node</i> Switch .....	37
Gambar IV.11 <i>Pseudocode</i> untuk pemrosesan <i>node</i> For .....	37
Gambar IV.12 <i>Pseudocode</i> untuk pemrosesan <i>node</i> ForEach .....	37
Gambar IV.13 Tampilan antarmuka <i>input source code</i> .....	39
Gambar IV.14 Tampilan antarmuka <i>upload file</i> .....	40
Gambar IV.15 Tampilan antarmuka <i>select 'index' file</i> .....	40
Gambar IV.16 Tampilan antarmuka ketika tidak ditemukan <i>vulnerability</i> .....	41
Gambar IV.17 Tampilan antarmuka ketika ditemukan <i>vulnerability</i> .....	41

## DAFTAR TABEL

Tabel III.1 Tabel kebenaran terjadinya <i>vulnerability</i> .....	18
Tabel III.2 Diagram <i>use case</i> kebutuhan fungsional sistem .....	31
Tabel IV.1 Tabel daftar CVE ID yang diuji.....	43
Tabel IV.2 Tabel hasil pengujian .....	44

# **BAB I**

## **PENDAHULUAN**

Bab ini berisikan penjelasan tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi dan juga sistematika pembahasan dari Tugas Akhir. Hal tersebut dijelaskan pada subbab-subbab berikut.

### **I.1 Latar Belakang**

Dewasa ini teknologi informasi dan komunikasi berkembang sangatlah cepat. Kehidupan sehari-hari tidak terlepas dari hal-hal yang berbau teknologi. Mulai dari transportasi, jual beli, hiburan, dan lain-lain. Hampir semua teknologi tersebut dikembangkan dengan teknologi berbasis web. Seiring dengan perkembangan yang cepat tersebut, semakin banyak orang yang tertarik untuk menjadi *Web Software Engineer* atau biasa disebut *web developer*.

Namun dari sekian banyak *developer* tersebut, tidak banyak yang sadar akan pentingnya kualitas dari sebuah *code* khususnya untuk masalah keamanan. Banyak yang hanya melihat dari sisi fungsionalitas-nya saja. Bahkan seorang *web developer* yang sadar pun terkadang lalai dalam memperhatikan aspek selain fungsionalitas. Berdasarkan pada situs resmi CVE Details, terdapat sekitar 14.600 *vulnerability* yang dilaporkan pada tahun 2017, peningkatan drastis jika dibandingkan pada tahun 2016 terdapat 6.447 laporan. Hal ini juga dipengaruhi oleh kompleksnya suatu aplikasi yang memiliki ribuan bahkan lebih *source code* yang menyebabkan kurang efektifnya apabila dilakukan *review* secara manual oleh seorang *developer*. Hal tersebut akan berakhir kepada kualitas *code* yang buruk. *Code* yang buruk menyebabkan banyaknya *bug* dan berakhir kepada munculnya celah keamanan pada sebuah aplikasi.

Tidak sedikit kerugian yang dialami apabila suatu aplikasi yang dikembangkan memiliki kualitas *code* yang buruk. Contohnya yaitu aplikasi berjalan tidak sesuai sebagaimana mestinya, adanya pencurian data-data penting pada suatu individu

ataupun suatu perusahaan. Dan yang sering terjadi yaitu kasus pemerasan akibat kelalaian dari *developer* itu sendiri.

Dengan banyaknya masalah yang dapat terjadi akibat kelalaian dari seorang *web developer* maka dibutuhkan sebuah metode atau kakas yang dapat meminimalisir kelalaian tersebut seperti dengan memanfaatkan teknologi pemindai *source code* secara otomatis atau yang disebut dengan *static code analysis* sehingga para *web developer* mengetahui apabila terdapat suatu *code* yang buruk.

## **I.2 Rumusan Masalah**

Berdasar pada latar belakang yang sudah dijelaskan sebelumnya, dapat diperoleh rumusan masalah yang ada. Rumusan masalah diharapkan mampu terjawab dengan dilaksanakannya Tugas Akhir ini. Rumusan-rumusan masalah tersebut yaitu:

1. Bagaimana mendeteksi kerentanan dengan metode *taint analysis*?
2. Pemodelan *source code* seperti apa sehingga akurasi pendeteksian kerentanan menjadi lebih baik?
3. Jenis celah keamanan apa saja yang dapat dideteksi dengan metode *taint analysis*?
4. Metode evaluasi seperti apa yang tepat untuk mengevaluasi metode ini?

## **I.3 Tujuan**

Berdasarkan rumusan masalah yang sudah dirumuskan pada subbab sebelumnya, diperoleh tujuan dari Tugas Akhir. Tujuan dari pengerjaan Tugas Akhir ini yaitu untuk membuat sebuah kakas *static code analysis* untuk aplikasi web dengan metode *taint analysis* dalam mendeteksi celah keamanan.

## **I.4 Batasan Masalah**

Dalam pengerjaan Tugas Akhir ini, terdapat beberapa batasan lingkup pengerjaan. Batasan-batasan tersebut yaitu:

1. *Static code analysis* yang dilakukan hanya pada *web application* yang dibangun dengan bahasa pemrograman PHP.

2. Objek evaluasi yaitu *Open Source Software PHP project* yang *vulnerability*-nya terdaftar di situs resmi CVE (*Common Vulnerabilities and Exposures*).

## **I.5 Metodologi**

Pengerjaan Tugas Akhir dibagi menjadi beberapa tahapan. Tahapan-tahapan tersebut yaitu:

1. Identifikasi dan analisis masalah

Tahapan pertama yaitu dilakukan pengumpulan data atau teori terkait *static code analysis*. Dari data tersebut kemudian dilakukan identifikasi masalah yang dapat diselesaikan dengan metode maupun pendekatan tertentu. Dari masalah tersebut, kemudian dilakukan analisis terhadap masalah yang akan diselesaikan.

2. Eksplorasi metode *taint analysis*

Pada tahapan ini dilakukan eksplorasi metode yang mungkin digunakan pada tahap selanjutnya, yaitu perancangan dan desain sistem. Dilakukan juga beberapa perbandingan kelebihan dan kekurangan dari metode-metode yang ada. Setelah itu dilakukan perancangan dan desain sistem.

3. Rancang dan desain sistem

Pada tahapan ini dilakukan perancangan dan desain sistem berdasarkan hasil identifikasi dan analisis masalah dan juga eksplorasi metode untuk pengembangan sistem yang paling relevan.

4. Implementasi dan pengembangan sistem

Dari hasil rancang dan desain sistem, kemudian dilakukan implementasi dan pengembangan sistem sehingga kakas dapat digunakan dan dapat diuji sesuai kebutuhan.

## 5. Evaluasi kinerja sistem

Tahapan terakhir yaitu dilakukan evaluasi terhadap sistem yang sudah diimplementasikan sehingga sistem dapat menjawab kebutuhan-kebutuhan yang harus dipenuhi.

## I.6 Sistematika Pembahasan

Sistematika pembahasan pada buku Tugas Akhir ini yaitu:

### 1. Pendahuluan

Bagian ini berisikan penjelasan tentang latar belakang pengerjaan Tugas Akhir, rumusan masalah, tujuan yang ingin dicapai, batasan-batasan dalam menyelesaikan masalah, serta metodologi dalam mengerjakan Tugas Akhir.

### 2. Studi Literatur

Bagian ini berisikan penjelasan tentang konsep dasar yang digunakan sebagai dasar dalam pengerjaan Tugas Akhir. Hal-hal yang dijelaskan yaitu terkait *source code*, *web vulnerability*, *web application*, serta *static code analysis*.

### 3. Analisis Masalah dan Rancangan Solusi

Bagian ini berisikan penjelasan mengenai penggunaan *taint analysis*. Selanjutnya berisi penjelasan terkait analisis permasalahan yang ada dalam membangun *static code analysis*, selanjutnya terdapat evaluasi metode *taint analysis* serta analisis kebutuhan dan alur kerja yang digunakan kakas.

### 4. Implementasi Solusi dan Pengujian

Bagian ini berisikan penjelasan terkait implementasi solusi dari analisis permasalahan Tugas Akhir. Implementasi solusi meliputi arsitektur serta kebutuhan kakas. Selain itu, bagian ini juga berisi penjelasan terkait pengujian kakas. Pengujian kakas meliputi tujuan pengujian, hasil pengujian, serta kesimpulan dari hasil pengujian.



## 5. Kesimpulan dan Saran

Bagian ini berisikan tentang kesimpulan dan saran dari pengerjaan Tugas Akhir. Kesimpulan meliputi hasil yang didapatkan dari pengerjaan Tugas Akhir, serta pelajaran yang didapatkan setelah mengerjakan Tugas Akhir. Saran meliputi hal-hal yang dapat ditingkatkan pada solusi dari permasalahan Tugas Akhir untuk penelitian lebih lanjut.

## **BAB II**

### **STUDI LITERATUR**

Bab ini berisi penjelasan terkait hasil dari studi literatur yang telah dilakukan untuk membantu proses pengerjaan Tugas Akhir. Hasil dari studi literatur digunakan sebagai dasar dalam pengerjaan Tugas Akhir. Hal-hal yang dijelaskan pada bab ini yaitu *source code*, *web vulnerability*, *web application*, dan *static code analysis*.

#### **II.1 *Web Application Security***

Pada subbab ini akan dijelaskan mengenai *web application* dan juga terkait *security* yang ada pada sebuah *web application*. Selain itu juga akan dijelaskan terkait *vulnerability* serta klasifikasi-klasifikasi yang ada.

*Security* merupakan salah satu aspek penting dalam mengembangkan sebuah *web application*. Apabila tingkat keamanan suatu *web application* rendah, maka sangat mungkin untuk terdapat *vulnerability* pada aplikasi *web* tersebut. The Open Web Application Security Project (OWASP) bahkan merilis sebuah dokumen “OWASP Top 10 Most Critical Web Application Security Risks” untuk meningkatkan kesadaran akan pentingnya aspek keamanan dalam mengembangkan sebuah *web application*.

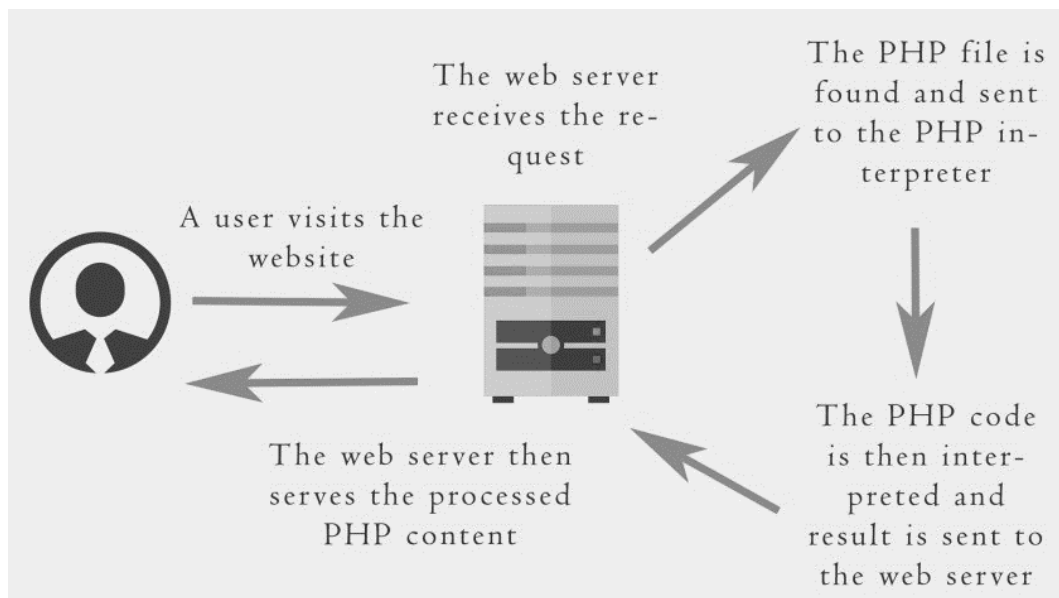
##### **II.1.1 *Web Application***

Menurut (Chaffe, 1999) *web application* merupakan sebuah konsep yang diperkenalkan dengan bahasa pemrograman Java melalui *Servlet Specification ver 2.2*. [2.1?]. Sebuah aplikasi berbasis *web* merupakan program komputer yang terdiri dari *client* dan *server* yang dimana program tersebut berjalan pada sebuah aplikasi *web browser* dengan menggunakan sebuah jaringan internet.

Dewasa ini, aplikasi berbasis *web* sangat populer dan sering sekali ditemui karena kemudahan dalam pengaksesan oleh *client*. Selain itu juga terdapat kemampuan dalam memelihara dan mengembangkan sebuah aplikasi *web* tanpa harus

mendistribusikan ulang (terpusat pada *server*). Hal menarik lainnya dari sebuah aplikasi berbasis *web* yaitu tidak bergantung pada *platform* tertentu sehingga dapat diakses oleh dimana saja.

Bahasa pemrograman yang sering digunakan dalam membuat sebuah *web application* yaitu PHP (PHP Hypertext Preprocessor). Menurut (Woo Huiren, 2015), PHP merupakan bahasa pemrograman yang telah digunakan oleh jutaan *users* di seluruh dunia. Mempelajari PHP sangatlah gampang namun untuk membangun sebuah kode PHP yang “*clean*” sangatlah susah. PHP *code* diinterpretasikan menggunakan PHP *interpreter* yang dipersembahkan oleh Zend Engine. Dibawah ini merupakan gambaran tentang bagaimana PHP bekerja ketika pengguna mengunjungi sebuah situs.



Gambar II.1 Workflow ketika pengguna mengunjungi situs.  
(<https://woohuiren.me/blog/introduction-to-php/>)

Secara garis besar, ketika pengguna mengunjungi sebuah *web server*, *web server* akan mengirimkan request ke sebuah berkas PHP dan kemudian berkas tersebut dikirimkan ke sebuah *interpreter*, *interpreter* tersebut kemudian mengirimkan hasil interpretasi dari berkas PHP tersebut ke *web server* dan kemudian diteruskan ke pengguna.

## II.1.2 Web Vulnerability

Dalam Oxford Living Dictionaries, *vulnerability* dapat diartikan sebagai kualitas atau status yang kemungkinan untuk diserang atau dirugikan, baik secara fisik maupun emosional. Begitupun pada keamanan sistem komputer, yang juga terdapat *vulnerability* atau kerentanan yang menyebabkan suatu sistem dapat diserang orang pihak yang tidak bertanggung jawab sehingga dapat menimbulkan kerugian.

Menurut RFC 2828, *vulnerability* merupakan kelemahan dalam desain, implementasi, atau operasi dan manajemen sistem yang dapat dieksploitasi untuk melanggar kebijakan keamanan sistem.

Merujuk pada Seven Pernicious Kingdoms, sebuah taksonomi yang dibuat oleh Tsipenyuk, Chess, dan McGraw (2005), terdapat tujuh klasifikasi *vulnerability* yaitu:

1. *Input Validation and Representation*
2. *API Abuse*
3. *Security Features*
4. *Time and State*
5. *Error Handling*
6. *Code Quality*
7. *Encapsulation*

Pada Tugas Akhir ini akan difokuskan pada tiga klasifikasi yaitu *Input Validation and Representation*, *Security Features*, dan *Code Quality*. Hal tersebut dikarenakan yang paling berkaitan dengan *static code analysis*.

### II.1.2.1 Input Validation and Representation

Masalah yang terjadi terkait *input validation* merupakan *vulnerability* yang paling berbahaya dan yang paling sering terjadi pada keamanan suatu perangkat lunak. Permasalahan yang terjadi pada *input validation* yaitu disebabkan oleh *input* yang tidak terpercaya yang dimasukkan oleh pengguna, seperti adanya meta-karakter, *encoding*, dan representasi yang lain. Kesalahan dalam memvalidasi *input* dapat menyebabkan sebuah *injection attack* bahkan sampai dengan *memory leakage*.

Oleh karena itu, sebuah aplikasi seharusnya melakukan pengecekan dan validasi terhadap *input* untuk meminimalisir hal-hal yang tidak diinginkan. Merujuk pada OWASP Top 10 - 2017, *vulnerability* yang umum terjadi diantaranya yaitu *Injection*, serta *Cross-Site Scripting* (XSS).

## A. Injection

Masalah utama yang sering terjadi yaitu *injection*, yang disebabkan oleh tidak di-filternya masukan pengguna. Hal tersebut dapat dilihat pada OWASP Top 10 mulai dari tahun 2010, 2013, hingga 2017 yang dimana *injection* menduduki peringkat pertama dalam *vulnerability* yang paling sering terjadi.

### 1. Command Injection

*Command injection* dapat menyebabkan terjadinya *execute arbitrary code* pada *server* atau yang biasa disebut dengan *Remote Code Execution* (RCE). Sebagai contoh yaitu kode PHP yang berfungsi sebagai kalkulator pada *server side*.

```
$input = $_GET['exp'];  
eval('$answer = `'.$input.`');
```

Apabila pengguna memasukkan parameter *exp* yaitu `10; system('rm -rf /')` maka fungsi *eval* PHP akan mengeksekusi dua *command* dengan *command* kedua yaitu menghapus semua *file* yang ada pada *server*. Oleh karena itu, ada baiknya dilakukan filter atau *sanitize* terhadap masukan pengguna.

### 2. SQL Injection

Sama halnya dengan *command injection*, *SQL injection* disebabkan oleh tidak dilakukannya filter atau *sanitize* terhadap masukan pengguna sehingga pengguna dapat mengubah hasil dari suatu *query* SQL bahkan dapat mengubah basis data itu sendiri. Sebagai contoh, pada kode PHP berikut

```
$query = mysql_query("SELECT * FROM user WHERE
    username = '". $_POST['username'] . "' AND password
    = '". $_POST['password'] . "'");
$count = mysql_num_rows($query);
```

Apabila pengguna memasukkan parameter username yaitu bebas' or 1 = 1 -- maka *query* SQL akan berhenti sampai dengan bagian -- dikarenakan sisanya akan dianggap sebagai *comment* pada *query*.

## B. Cross-site Scripting (XSS)

*Cross-Site Scripting* atau yang sering disebut dengan XSS merupakan *vulnerability* yang terjadi ketika pengguna dapat memasukkan sebuah *script* pada *input* yang kemudian *input* tersebut ditampilkan pada halaman yang di-generate oleh sebuah *web application* (Mitchell, 2010). XSS terbagi dua tipe, yaitu Reflected XSS dan juga Stored XSS. Reflected XSS terjadi ketika masukan pengguna hanya ditampilkan sesaat seperti pada hasil *query* pencarian. Sedangkan Stored XSS terjadi ketika masukan pengguna disimpan pada *server* seperti *databases*, *comment* pada forum, dll (OWASP, 2018).

### II.1.2.2 Security Features

*Security Features* atau fitur keamanan pada sistem perangkat lunak harus diimplementasikan dengan benar untuk menghindari adanya *vulnerability* pada sistem tersebut. Contoh kesalahan dengan tidak memperhatikan sebuah fitur keamanan yaitu menyimpan *plain text* sebuah *password* pada *source code*. Hal tersebut dapat dihindari dengan menggunakan sebuah fitur keamanan berupa *cryptography*. Selain itu, fitur-fitur keamanan yang dapat digunakan yaitu berupa *authentication*, *access control*, dan lain-lain.

### II.1.2.3 Code Quality

Kualitas dari sebuah *code* merupakan hal yang penting dalam meminimalisir adanya *vulnerability*. Kualitas yang tidak baik akan menyebabkan perilaku yang tidak dapat diprediksi. Dengan memanfaatkan sebuah *null pointer* atau sebuah *infinite loop* dapat menyebabkan terjadinya *Denial of Service attack*. Namun bukan

cuman itu, banyak hal lain yang dapat dilakukan oleh *attacker* dengan memanfaatkan kualitas *code* yang buruk.

## **II.2 *Static Code Analysis***

Untuk menghindari terjadinya kesalahan ataupun *vulnerability* yang sudah disebutkan, maka perlu dilakukan metode untuk menemukan kesalahan pada program mulai dari awal pengembangan hingga tahap *production*. Terdapat beberapa cara, diantaranya dilakukan *code review*, *static code analysis*, *unit test*, *vulnerability scan*, dll.

Salah satu keunggulan dari *static code analysis* dibandingkan metode yang lain dalam menemukan kesalahan pada program yaitu *vulnerability* dapat ditemukan tanpa dilakukan eksekusi terhadap program. *Vulnerability* ditemukan hanya dengan membaca *source code* dari program tersebut dengan berbagai pendekatan.

### **II.2.1 Pendekatan pada *Static Code Analysis***

Pada subbab ini dijelaskan pendekatan-pendekatan yang ada pada *static code analysis*. Dalam Tugas Akhir ini, pendekatan yang digunakan yaitu *lexical analysis* serta *taint analysis*.

#### **II.2.1.1 *Lexical Analysis***

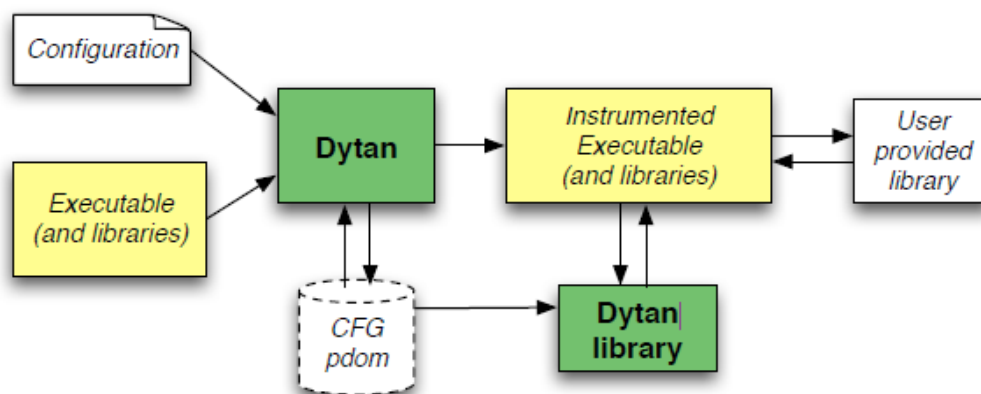
*Static code analysis* dengan pendekatan *lexical analysis*, dilakukan dengan terlebih dahulu *source code* direpresentasikan sebagai ‘token’ agar lebih mudah untuk dimanipulasi (Sotirov, 2005). Sebagai contoh kode PHP yang belum diubah menjadi token `<?php $name = "Rozi"; ?>` kemudian setelah diubah menjadi token, akan menjadi seperti berikut.

```
T_OPEN_TAG
T_VARIABLE
=
T_CONSTANT_ENCAPSED_STRING
;
T_CLOSE_TAG
```

### II.2.1.2 Taint Analysis

Berdasarkan (Edward dkk., 2010), *taint analysis* dilakukan dengan terlebih dahulu menandai masukan pengguna sebagai *tainted*, setelah itu semua *variable* ataupun fungsi yang dilewati oleh *tainted* akan ditandai sebagai *tainted* juga. Namun terdapat dua policy yang mendukung *taint analysis*. Pertama yaitu *taint* hanya bergantung terhadap *memory cell*. Kedua yaitu apabila *address of memory* terkena *tainted*, maka nilai dari *address* tersebut juga *tainted*.

Dytan (Clause dkk., 2007), merupakan sebuah *generic dynamic taint analysis framework* yang menyediakan beberapa kelebihan dengan *dynamic tainting*. Terdapat tiga karakteristik pada Dytan, *taint sources*, *propagation policy*, dan *taint sinks*. *Taint sources* merupakan deskripsi dari *memory location* yang harus diinisialisasikan dengan *taint marking*. *Propagation policy* menjelaskan bagaimana *taint markings* harus dipropagasikan selama eksekusi berlangsung. *Taint sinks* merupakan lokasi dimana *code* dari pengguna dilakukan pengecekan dengan *taint markings* dari beberapa *memory location*. Selanjutnya dilakukan *Control-flow Based Taint Propagation* yang bertujuan untuk mendeteksi adanya *vulnerability*.



Gambar II.2 Arsitektur Dytan: Dynamic Taint Analysis Framework.  
(Clause, 2007)

*Taint Analysis* dilakukan dengan mengidentifikasi *variable* yang dicurigai berbahaya (*tainted*), dikarenakan berasal dari masukan pengguna. Kemudian dilakukan penelusuran terhadap *variable* tersebut terhadap fungsi yang juga



dicurigai terdapat *vulnerable* di dalamnya. Jika *tainted variable* tersebut masuk ke dalam *sink* sebelum dilakukan filter atau *sanitize*, maka ditandai sebagai *vulnerability*. Secara garis besar, *taint analysis* terdiri dari tiga komponen yang penting, yaitu:

1. *Sources*

*Sources* merupakan semua sumber yang dapat digunakan oleh penyerang untuk menyisipkan *malicious code* yang berbahaya. Pada sebuah aplikasi PHP berbasis web, beberapa yang menjadi *sources* diantaranya parameter HTTP *request*, *session*, *file* yang diunggah, dan lain-lain

2. *Sinks*

*Sinks* merupakan tempat atau fungsi akhir yang menjalankan atau mengeksekusi aksi yang rentan terhadap serangan. Pada sebuah aplikasi PHP berbasis web, beberapa yang menjadi *sinks* diantaranya yaitu fungsi eksekusi SQL *query*, fungsi *include* file lain, fungsi yang berhubungan dengan *operating system*, dan lain-lain.

3. *Sanitizers*

*Sanitizers* merupakan apapun yang dapat melindungi atau melakukan filter terhadap *sources* yang diberikan. Pada sebuah aplikasi PHP berbasis web, beberapa yang menjadi *sanitizers* diantaranya yaitu fungsi escape tag html, fungsi escape SQL *query*, dan lain-lain.

Apabila terdapat suatu *sources* yang melewati sebuah *sinks* tanpa melewati sebuah *sanitizers* terlebih dahulu, maka dapat dikatakan aplikasi web tersebut mungkin rentan terhadap sebuah serangan.

## **II.2.2 Static Code Analysis Tools**

Terdapat beberapa *tools* yang digunakan untuk melakukan *static code analysis*. Terkhusus dalam pengembangan sebuah *web application* terdapat sebuah *tools* untuk melakukan pengecekan terhadap *source code* PHP yang umum digunakan untuk membangun sebuah *web application*. *Tools* tersebut yaitu RIPS dan juga FindBugs.

### II.2.2.1 RIPS

RIPS (*Re-Inforce PHP Security*) merupakan *tools* yang populer untuk melakukan *static code analysis* khususnya pada bahasa pemrograman PHP. RIPS dikembangkan pertama kali oleh Johannes Dahse sebagai *software open-source* dengan menggunakan pendekatan *lexical analysis*. Versi pertama dari RIPS berhasil mendeteksi beberapa *vulnerability* seperti SQL Injection, Cross-site Scripting, Local File Inclusion, dll. Kelemahan dari versi pertama ini yaitu banyaknya *false positive*. Namun pada tahun 2016, muncul versi terbaru dari RIPS yang dirilis oleh RIPS Technologies yang menggunakan teknik yang lebih *advance* sehingga dapat meminimalisir terjadinya *false positive*. Akan tetapi, masih terdapat beberapa kelemahan yang dimiliki oleh RIPS, yaitu RIPS masih banyak membuat asumsi, terdapat limit pada *source code* yang memiliki model *Object Oriented Programming*, juga pada tipe data, serta *flow data*.

Berdasarkan pada situs RIPS Technologies, RIPS bekerja dengan cara terlebih dahulu dilakukan transformasi PHP *code* menjadi representasi *graph*. *Code* tersebut diubah menjadi *token*, *abstract syntax tree*, sehingga membentuk sebuah *control flow graph*. Dengan bantuan *taint analysis*, masukan pengguna dapat dideteksi apabila terdapat masukan yang tidak difilter terlebih dahulu.

### II.2.2.2 FindBugs

Menurut (Grindstaff, 2004), FindBugs merupakan sebuah kakas *open source* yang melakukan *static analysis* pada Java *code* yang dikembangkan oleh Bill Pugh dan David Hovemeyer. *Vulnerability* diklasifikasikan menjadi empat, *scaries*, *scary*, *troubling*, dan juga *of concern*. Beberapa kelemahan dari FindBugs yaitu, masih terdapat *false positive* sehingga developer perlu mengalokasikan *resource* tambahan berupa waktu untuk melakukan *review* secara manual. Selain itu FindBugs juga biasanya hanya menemukan *subset* dari *issue* yang sebenarnya.

Hal terpenting dalam *taint analysis* pada FindBugs yaitu dengan membuat *taint state* yang direpresentasikan dengan beberapa nilai yaitu *TAINTED*, *UNKNOWN*, *SAFE*, dan *NULL*. *TAINTED* apabila terdapat instruksi dengan pemanggilan metode

yang *tainted*, *SAFE* disimpan untuk *load constant instruction*, *NULL* untuk *aconst\_null*, dan *UNKNOWN* adalah *default value*. Penggabungan *taint state* didefinisikan sehingga dapat dibandingkan  $TAINTED > UNKNOWN > SAFE > NULL$ , gabungan diambil dari yang terbesar, misalnya *TAINTED* dan *SAFE*, maka menjadi *TAINTED*.

## **BAB III**

### **ANALISIS DAN RANCANGAN**

Bab ini berisi penjelasan mengenai penggunaan *taint analysis* dalam menemukan celah keamanan pada aplikasi web. Selanjutnya berisi penjelasan terkait analisis permasalahan yang ada dalam membangun *static code analysis*, yaitu analisis pemrosesan *source code*, selanjutnya terdapat evaluasi metode *taint analysis* serta analisis kebutuhan dan alur kerja yang digunakan kakas. Hal tersebut akan dijelaskan pada subbab-subbab berikut.

#### **III.1 Kemampuan *Taint Analysis* dalam Menemukan Celah Keamanan**

Masalah yang paling utama yaitu kurangnya kesadaran dari sebuah *web developer* dalam mengembangkan sebuah *web application* dengan bahasa pemrograman PHP pada aspek *security*. Rata-rata *developer* mengembangkan aplikasi hanya melihat dari sisi fungsionalitasnya saja. Tidak sedikit *developer* yang mengembangkan sebuah perangkat lunak yang tidak dibantu dengan *tutorial* yang ada di *internet*. Namun tidak sedikit juga *web tutorial* yang sadar terhadap *security* dari sebuah *code* tersebut. Hasil akhir dari kelalaian ini yaitu banyak terjadi *vulnerability* dalam perangkat lunak yang dibangun. Selain itu, aplikasi yang dibangun biasanya cukup kompleks dan memiliki ribuan bahkan lebih baris *source code*, sehingga sulit dan tidak efektif untuk dilakukan *code review* secara manual. Oleh karena itu, terdapat beberapa alternatif solusi berupa *tools* dalam melakukan *static code analysis* yang berfungsi untuk mendeteksi kualitas dari *source code* yang buruk baik berupa *bug* maupun *vulnerability* yang ada secara otomatis.

Permasalahan yang sering terjadi yaitu tidak adanya *filter* atau *sanitize* terhadap masukan dari sebuah pengguna. Hal tersebut menyebabkan pengguna dapat memasukkan sebuah *malicious code* yang dapat menyebabkan terjadinya sebuah *vulnerability* yaitu tipe *injection* yang dapat menyebabkan terjadinya pencurian data-data penting hingga merusak sistem yang sudah dibangun. Oleh karena itu,

salah satu solusi dalam meminimalisir kejadian tersebut yaitu dengan menggunakan *taint analysis*.

Dalam melakukan *code analysis*, terdapat dua metode yaitu *dynamic* dan *static*. Berbeda dengan *dynamic taint analysis* yang dimana harus melakukan eksekusi terhadap *source code*, *static code analysis* dilakukan hanya dengan membaca *source code* tanpa dilakukan eksekusi. Beberapa komponen penting dalam melakukan *static code analysis* yaitu *input (source code)*, dan *rules* yang mengatur *variable* dan *fungsi* yang dicurigai berbahaya.

Dengan *taint analysis*, *variable* yang merupakan masukan dari pengguna akan ditandai sebagai *tainted* dan akan ditelusuri setiap fungsi-fungsi yang akan dilewati, dan apabila fungsi tersebut merupakan fungsi berbahaya atau *sinks* maka dapat dikatakan sebagai *vulnerability*.

```
<?php
1
2 $name = $_GET["name"]; # Input dari user, maka variable tainted
3 echo "Hello, ". $name; # Sink dengan tainted parameter
4                               # Maka merupakan vulnerability
5
```

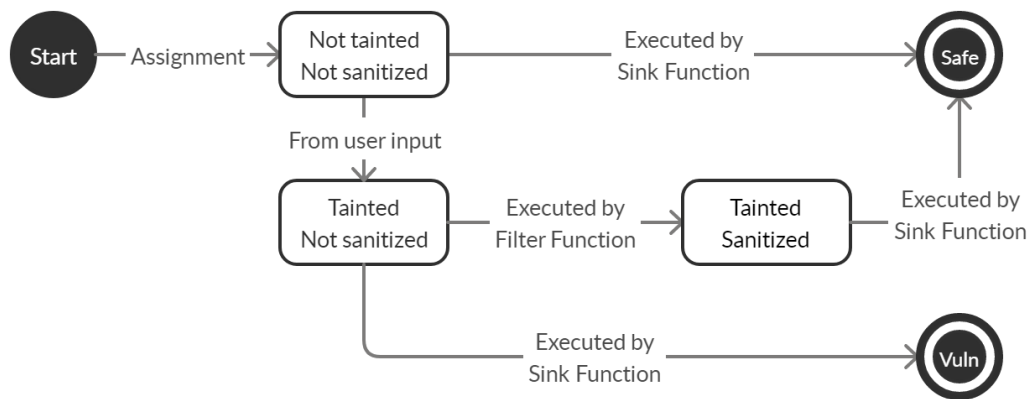
Gambar III.1 Contoh kode yang terdapat *vulnerability*

Contoh Gambar III.1 diatas merupakan contoh kode yang dapat menyebabkan terjadinya *vulnerability*, yaitu *Cross-site Scripting*. Namun, apabila masukan dari pengguna tersebut terlebih dahulu melewati fungsi *sanitize*, maka input pengguna dianggap aman dari bahaya *injection*. Contoh kode yang aman dapat dilihat pada Gambar III.2

```
<?php
1
2 $name = $_GET["name"]; # Input dari user, maka variable tainted
3 $name = htmlentities($name) # Tainted melewati fungsi sanitize
4 echo "Hello, ". $name; # Sink dengan sanitized parameter
5                               # Maka bukan merupakan vulnerability
6
```

Gambar III.2 Contoh kode yang aman

Dengan *taint analysis*, *vulnerability* yang dapat dideteksi merupakan *vulnerability* tipe *injection* dikarenakan hanya dilakukan pengecekan pada fungsi atau *sink* yang berbahaya, juga pada masukan atau *input* dari pengguna. Sederhananya, suatu program dikatakan terdapat *vulnerability* jika dan hanya jika terdapat sebuah *variable* yang *tainted*, tidak *sanitized* dan kemudian dieksekusi oleh fungsi berbahaya atau *sink*. Agar lebih mudah dipahami, berikut visualisasi dalam bentuk *state* diagram yang dapat dilihat pada Gambar III.3.



Gambar III.3 *State* diagram *taint analysis*

Selain *state* diagram, dapat dilihat pada tabel kebenaran ketika sesuatu dikatakan *vulnerable*. Dari delapan kombinasi, hanya terdapat satu kemungkinan terjadinya *vulnerability*, yaitu ketika *tainted*, *not sanitized*, dan *executed*. Tabel kebenaran dapat dilihat pada Tabel III.1 dibawah ini.

Tabel III.1 Tabel kebenaran terjadinya *vulnerability*

Tainted	Sanitized	Executed by Sink	Vuln
T	T	T	F
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Ketiga komponen *sources*, *sanitizers*, serta *sinks* nantinya akan didefinisikan menggunakan *knowledge base*. *Knowledge base* yang digunakan berupa file *config* yang bertujuan untuk mendaftarkan semua *knowledge* yang dimiliki. *Knowledge* tersebut diantaranya yaitu *variable* apa saja yang termasuk dalam *sources*, serta fungsi apa saja yang termasuk dalam *sanitizers* atau *sinks*. *Knowledge base* yang digunakan pada proses pengerjaan Tugas Akhir ini dapat dilihat pada Lampiran A. Dalam melakukan pengecekan terhadap *source code* yang diberikan, terlebih dahulu dilakukan *preprocessing* berupa *parsing* terhadap *source code* yang kemudian menghasilkan sebuah *abstract syntax tree* untuk diproses lebih lanjut. Penjelasan lebih lanjut akan dibahas pada subbab III.2

### **III.2 Analisis Pemrosesan Source Code**

Berdasarkan subbab III.1 dapat disimpulkan untuk menemukan *vulnerability* bertipe *injection* dapat dilakukan *static code analysis* dengan metode *taint analysis*. Hal ini dapat dilakukan dengan melakukan *parsing* terhadap *source code* dan menghasilkan sebuah *abstract syntax tree*, kemudian untuk tiap *node* pada *tree* dilakukan proses sesuai dengan jenis token masing-masing *node*. Pada subbab selanjutnya dijelaskan lebih lanjut mengenai permasalahan yang ada.

#### **III.2.1 Preprocessing: Parsing dan Pembuatan Abstract Syntax Tree**

*Preprocessing* dilakukan dengan melakukan *parsing* dan pembuatan suatu *abstract syntax tree*. Tujuan dilakukannya *preprocessing* yaitu untuk memudahkan dalam melakukan *taint analysis*, khususnya dalam pemrosesan tiap *node* pada *abstract syntax tree* yang dihasilkan.

*Preprocessing* dilakukan dengan bantuan library *open source* yaitu *phply*, *phply* merupakan sebuah *parser* bahasa pemrograman PHP dengan menggunakan bantuan *PLY* (Python Lex-Yacc), yang mengubah *source code* dalam bentuk *lexical* atau token. Contohnya untuk *source code* sederhana seperti pada gambar III.1 akan menghasilkan *abstract syntax tree* sebagai berikut.

```

('Assignment',
 {'expr': ('ArrayOffset',
          {'expr': u'name',
           'lineno': 2,
           'node': ('Variable', {'lineno': 2, 'name': u'$_GET'})}),
  'is_ref': False,
  'lineno': 2,
  'node': ('Variable', {'lineno': 2, 'name': u'$name'})})
('Echo',
 {'lineno': 3,
  'nodes': [('BinaryOp',
              {'left': u'Hello, ',
               'lineno': 3,
               'op': u'.',
               'right': ('Variable', {'lineno': 3, 'name': u'$name'})})]})

```

### III.2.2 Pemrosesan untuk tiap *node* pada *Abstract Syntax Tree*

*Abstract syntax tree* yang dihasilkan terdiri dari beberapa *node* sesuai dengan urutan pada *source code* yang diberikan. Tiap *node* terdiri dari tipe serta ekspresi dari *code* itu sendiri. Dalam pemrosesan tiap *node*, terlebih dahulu dilakukan pengecekan tipe tiap *node*. Perlakuan yang dilakukan untuk tiap tipe tentunya berbeda-beda, Tipe *node* yang berpengaruh terhadap *taint analysis* yaitu hal-hal yang berhubungan dengan *sources*, *sinks*, serta *sanitizers*.

*Sources* berpengaruh dikarenakan merupakan asal dari munculnya status *tainted*. Dalam *abstract syntax tree* yang diberikan, *sources* banyak ditemukan pada tipe *node* Assignment. *Sinks* berpengaruh dikarenakan merupakan asal dari munculnya status *executed*. *Sinks* banyak ditemukan pada tipe *node* FunctionCall. *Sanitizers* berpengaruh dikarenakan merupakan asal dari munculnya status *sanitized*. *Sanitizers* banyak ditemukan pada tipe *node* Assignment-FunctionCall. Penjelasan lebih lanjut untuk tiap *node* dijelaskan pada paragraf selanjutnya.



**InlineHTML**, merupakan tipe *node* berupa kode HTML biasa, sehingga bukan merupakan kode PHP. Oleh karena itu, tipe *node* ini tidak diproses. Contoh *source code* dengan tipe *node* InlineHTML dapat dilihat pada Gambar III.4.

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>My first PHP page</h1>
6
```

Gambar III.4 Contoh *source code* dengan tipe *node* InlineHTML

**Assignment**, merupakan salah satu tipe *node* yang berperan penting pada proses *taint analysis*. Hal ini disebabkan karena masukan dari pengguna akan disimpan kedalam sebuah daftar *variable*. Untuk tiap *variable*, parameter yang disimpan yaitu *value(object)*, *line number(int)*, *tainted(bool)*, dan *sanitized(bool)*. *Assignment* sendiri terdiri dari beberapa jenis *assignment*, diantaranya yaitu Assignment-Variable, Assignment-Array, Assignment-Constant, Assignment-FunctionCall, dll. Perlakuan untuk tiap jenis *assignment* pun berbeda-beda. Contoh jenis-jenis *source code* dengan tipe *node* Assignment dapat dilihat pada Gambar III.5.

```
<?php
1
2 # Variable Assignment
3 $myVariable = $anotherVar;
4
5 # Array Assignment
6 $myArray = array(1, 2, 3);
7
8 # Constant Assignment
9 $myConst = true;
10
11
```

Gambar III.5 Contoh *source code* dengan tipe *node* Assignment

Suatu *variable* dikatakan *tainted* ketika *variable* tersebut di-assign oleh masukan pengguna yang sudah didefinisikan sebelumnya melalui *knowledge base*. Contoh *source code* yang melakukan assignment dengan status *tainted* dapat dilihat pada

Gambar III.6. Sedangkan contoh *variable* yang berbahaya dapat dilihat pada Gambar III.7.

```
<?php
1
2 # Variable name Tainted
3 $name = $_GET["name"];
4 |
```

Gambar III.6 Contoh Assignment dengan status *tainted*

```
1
2 ▾ USER_INPUT = [
3     "$_GET",
4     "$_POST",
5     "$_COOKIE",
6     "$_REQUEST",
7     "$_FILES",
8     "$_SERVER",
9     "$HTTP_GET_VARS",
10    "$HTTP_POST_VARS",
11    "$HTTP_COOKIE_VARS",
12    "$HTTP_REQUEST_VARS",
13    "$HTTP_POST_FILES",
14    "$HTTP_SERVER_VARS",
15    "$HTTP_RAW_POST_DATA",
16    "$argc",
17    "$argv"
18 ]
19
```

Gambar III.7 Contoh *variable* yang berbahaya

Selain status *tainted*, status *sanitized* juga didapatkan pada tipe *node* Assignment, yaitu ketika suatu *variable* di-assign oleh fungsi yang termasuk kedalam *sanitizers*. Fungsi-fungsi yang termasuk ke dalam *sanitizers* juga terlebih dahulu didefinisikan melalui *knowledge* yang dimiliki. Contoh *source code* yang melakukan *assignment* dengan status *sanitized* dapat dilihat pada Gambar III.8. Sedangkan contoh fungsi-fungsi yang termasuk ke dalam *sanitizers* dapat dilihat pada Gambar III.9.

```

<?php
1
2 # Variable name Tainted
3 $name = $_GET["name"];
4
5 # Variable name Tainted-Sanitized
6 $name = htmlentities($name);
7

```

Gambar III.8 Contoh Assignment dengan status *sanitized*

```

1  XSS = [
2      "htmlentities",
3      "htmlspecialchars",
4      "highlight_string",
5  ]
6
7  SQL = [
8      "addslashes",
9      "dbx_escape_string",
10     "db2_escape_string",
11     "ingres_escape_string",
12     "maxdb_escape_string",
13     "maxdb_real_escape_string",
14     "mysql_escape_string",
15     "mysql_real_escape_string",
16     "mysqli_escape_string",

```

Gambar III.9 Contoh fungsi-fungsi yang termasuk *sanitizers*

**FunctionCall**, juga merupakan salah satu tipe *node* yang berperan penting pada proses *taint analysis*. FunctionCall dapat berupa fungsi yang sudah didaftarkan ataupun fungsi bawaan yang merupakan *sinks*. Fungsi bawaan yang merupakan *sinks* ditentukan dari *knowledge* yang sudah ditentukan sebelumnya. Contoh *source code* yang memanggil sebuah fungsi yang termasuk *sinks* dapat dilihat pada Gambar III.10. Sedangkan contoh fungsi-fungsi yang termasuk ke dalam *sinks* dapat dilihat pada Gambar III.11.

```

<?php
1
2 $name = $_GET["name"];
3
4 # FunctionCall Sinks
5 print("Hello, " + $name);
6
7 |

```

Gambar III.10 Contoh FunctionCall yang termasuk *sink*

```

1  ▾ XSS = [
2    "Echo",
3    "Print",
4    "print_r",
5    "exit",
6    "die",
7    "printf",
8    "vprintf",
9    "trigger_error",
10   "user_error",
11   "odbc_result_all",
12   "ovrimos_result_all",
13   "ifx_htmltbl_result",
14   ]
15
16  ▾ OS = [
17    "assert",
18    "create_function",
19    "eval",
20    "mb_ereg_replace",

```

Gambar III.11 Contoh fungsi-fungsi yang termasuk *sink*

Selain tipe *node* yang berpengaruh terhadap *taint analysis*, juga terdapat tipe *node* yang berpengaruh terhadap *control flow* dari *source code* yang diberikan. Beberapa contoh tipe *node* yang berpengaruh terhadap *control flow* yang mengatur jalannya suatu program, tipe *node* tersebut yaitu Include, If, Switch, For, Foreach, dll. Penjelasan lebih lanjut untuk tiap *control flow node* dijelaskan pada paragraf selanjutnya.

**Include**, dan/atau Require merupakan tipe *node* yang melakukan *load file* dari sumber yang lain, ketika ditemukan *node* dengan tipe Include, maka akan dilakukan

*taint analysis* terlebih dahulu terhadap *file* yang akan di-*include* sehingga dilakukan penggabungan *abstract syntax tree*. Contoh *source code* dengan tipe *node* Include dapat dilihat pada Gambar III.12.

```
<?php
1
2 # Include/Require dari file lain
3 include("settings.php");
4 require("myconst.php");
5
6 ...
7
```

Gambar III.12 Contoh *source code* dengan tipe *node* Include

**Function**, merupakan tipe *node* yang melakukan suatu pendaftaran fungsi. Mirip dengan tipe Assignment, nantinya setiap fungsi yang didaftarkan dicatat pada suatu daftar fungsi. Tujuan dari pendaftaran fungsi ini yaitu jika ditemukan sebuah *node* FunctionCall yang memanggil fungsi yang sudah didaftarkan, maka dilakukan analisis terhadap *node* di dalam fungsi tersebut. Contoh *source code* dengan tipe *node* Function dapat dilihat pada Gambar III.13.

```
<?php
1
2 # Tipe node Function
3 function greeting($name) {
4     echo "Hello, ". $name;
5 }
6
7 # FunctionCall dengan fungsi-
8 # yang sudah didaftarkan
9 greeting("Rozi");
10
```

Gambar III.13 Contoh *source code* dengan tipe *node* Function

**Global**, merupakan tipe *node* yang melakukan pemanggilan *variable* global ke dalam sebuah fungsi. Hal ini dilakukan agar *variable* global dapat diakses di dalam *scope* suatu fungsi. Contoh *source code* dengan tipe *node* Global dapat dilihat pada Gambar III.14.

```

<?php
1
2 function greeting() {
3     # Tipe node Global
4     global $name;
5     echo "Hello, ". $name;
6 }
7
8 # Deklarasi variable global
9 $name = "Rozi";
10 greeting();
11

```

Gambar III.14 Contoh *source code* dengan tipe *node* Global

**Return**, merupakan tipe *node* yang mengembalikan suatu nilai pada sebuah fungsi. Hal ini dilakukan ketika terdapat *node* Assignment-FunctionCall, sehingga *return value* dari fungsi tersebut dapat disimpan ke dalam daftar *variable*. Contoh *source code* dengan tipe *node* Return dapat dilihat pada Gambar III.15.

```

<?php
1
2 function add($a, $b) {
3     $ret = $a + $b;
4     # Tipe node Return
5     return $ret;
6 }
7
8 $c = add(1, 2);
9

```

Gambar III.15 Contoh *source code* dengan tipe *node* Return

**If**, merupakan tipe *node* yang melakukan percabangan, salah satu tantangan pada *taint analysis* yaitu ketika suatu *event* hanya terjadi pada salah satu cabang, sehingga harus dipastikan suatu *event* terjadi jika dan hanya jika *event* tersebut berada di semua cabang. Selain itu, semua cabang akan ditelusuri walaupun terdapat kasus yang tidak mungkin. Contoh *source code* dengan tipe *node* If dapat dilihat pada Gambar III.16.

```

<?php
1
2 $name = $_GET["name"];
3 if ($name){
4     echo $name;
5 } else {
6     echo "I don't know u";
7 }
8

```

Gambar III.16 Contoh *source code* dengan tipe *node* If

**Switch**, merupakan tipe *node* yang mirip seperti tipe *node* If, sehingga perlakuan tipe *node* Switch mirip seperti tipe If. Perbedaannya yaitu pada tipe *node* Switch, terdapat sebuah *variable* tertentu yang mengatur *flow* dari program. Contoh *source code* dengan tipe *node* Switch dapat dilihat pada Gambar III.17.

```

<?php
1 $favcolor = "blue";
2 switch ($favcolor) {
3     case "red":
4         echo "Your favcolor is red!";
5         break;
6     case "blue":
7         echo "Your favcolor is blue!";
8         break;
9     default:
10        echo "I dont know ur favcolor";
11 }
12

```

Gambar III.17 Contoh *source code* dengan tipe *node* Switch

**For**, merupakan tipe *node* yang melakukan perulangan, karena kakas melakukan pemrosesan secara statis, maka hanya akan dilakukan pemeriksaan sebanyak satu kali dengan nilai *loop variable* yaitu nilai awal dilakukannya *assignment*. Contoh *source code* dengan tipe *node* For dapat dilihat pada Gambar III.18.

```

<?php
1
2 for ($i=0; $i < 10; $i++) {
3     echo $i;
4 }
5

```

Gambar III.18 Contoh *source code* dengan tipe *node* For

**Foreach**, merupakan tipe *node* yang mirip dengan tipe *node* For. Perbedaan tipe *node* Foreach dan For yaitu pada Foreach, untuk setiap nilai akan dilakukan iterasi, sehingga dilakukan pengecekan untuk tiap isi array. Contoh *source code* dengan tipe *node* Foreach dapat dilihat pada Gambar III.19.

```

<?php
1
2 $numbers = array(1, 2, 3);
3 foreach ($numbers as $number) {
4     echo $number;
5 }
6

```

Gambar III.19 Contoh *source code* dengan tipe *node* Foreach

### III.3 Evaluasi Metode *Taint Analysis*

Evaluasi dilakukan secara bertahap dimulai dengan aplikasi yang memiliki *source code* sederhana terlebih dahulu hingga aplikasi yang cukup kompleks. *Source code* dari aplikasi sederhana yang dirancang dapat dilihat pada Lampiran B. Selain itu, evaluasi juga dilakukan dengan menggunakan aplikasi *open source* yang tersebar di Internet. Aplikasi yang digunakan merupakan aplikasi yang memiliki CWE yang berhubungan dengan *injection*. Dari CWE ID tersebut dapat ditemukan CVE ID yang sesuai yang terdaftar pada situs resmi CVE. Hal tersebut dilakukan agar memastikan aplikasi yang diuji benar terdapat *vulnerability* di dalamnya. Proses ini dilakukan untuk mengetahui apakah kakas yang dibangun telah mencapai tujuan dalam mendeteksi *vulnerability*. Selain itu, juga akan dilakukan perbandingan terhadap beberapa kakas serupa yang sudah dijelaskan pada Bab II.



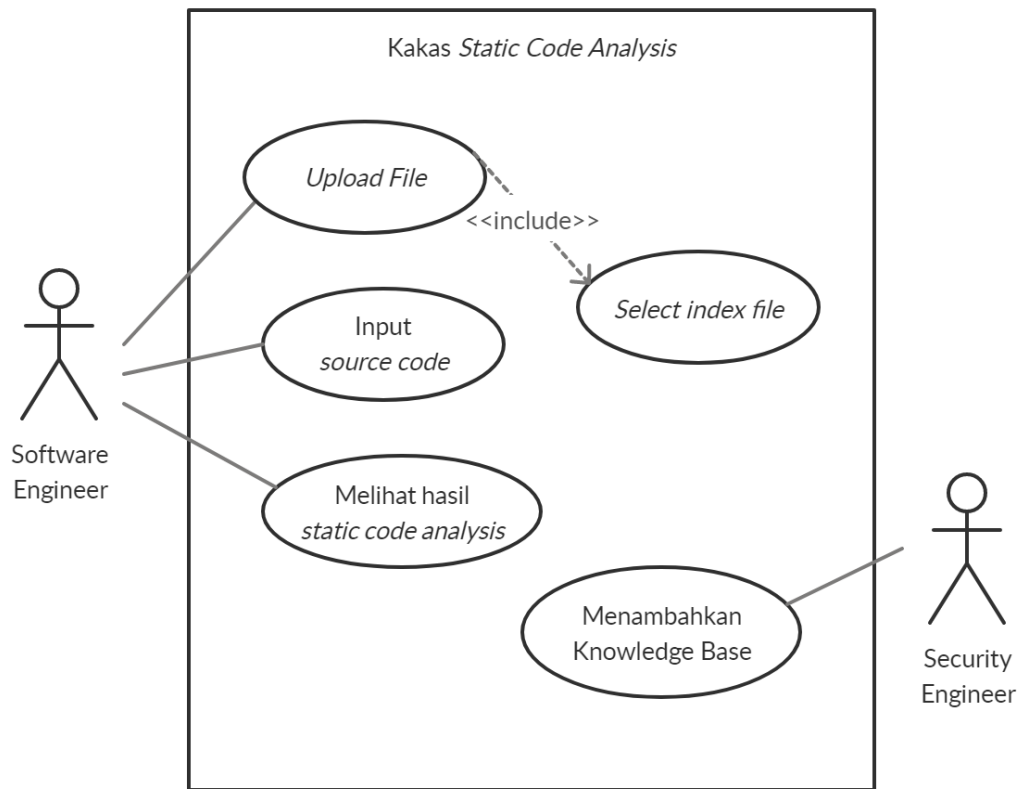
Masih terdapat banyak kelemahan pada kakas *static code analysis* yang ada saat ini, salah satu kelemahan yang sering ditemui yaitu banyaknya *false positive* dan/atau *false negative* yang menyebabkan perlunya diadakan pengecekan secara manual. Tentunya perlu dibutuhkan *resources* tambahan berupa alokasi tenaga maupun waktu untuk dilakukan pengecekan tambahan tersebut. Banyak hal yang menyebabkan kemunculan *false positive* tersebut. Diantaranya disebabkan oleh banyaknya fungsi bawaan pada bahasa PHP yang rentan terhadap *malicious code* serta bahasa pemrograman PHP yang sangat *dynamic* sehingga dapat terjadi kekeliruan ketika dilakukan analisis terhadap *source code* yang diberikan. Contohnya yaitu ketika terdapat sebuah deklarasi *variable* yang dilakukan oleh sebuah fungsi *base64* yang menyebabkan *static code analysis* tidak dapat mendeteksi *vulnerability* yang ada pada *source code* tersebut.

```
$a = base64_decode('c3lzdGVt');  
$a($_GET['input']);
```

Pada contoh potongan kode diatas, fungsi *system* tidak terdeteksi karena fungsi tersebut terlebih dahulu di-*encode* dengan *base64* sehingga kakas tidak mendeteksi adanya fungsi yang berbahaya.

### III.4 Analisis Kebutuhan

Berdasarkan analisis pada subbab III.2, dapat diambil beberapa kebutuhan yang harus dimiliki oleh kakas *static code analysis* yang akan dibangun. Kebutuhan ini berupa fungsi minimal yang dimiliki oleh kakas. Fungsionalitas ini digambarkan dengan diagram *use case* yang dapat dilihat pada Gambar III.20.



Gambar III.20 Diagram *use case* kebutuhan fungsional sistem

Berdasarkan diagram *use case* di atas, terdapat dua jenis aktor yang berhubungan dengan kakas ini, yaitu *software engineer* dan *security engineer*. *Software engineer* merupakan pengguna kakas pada umumnya yang dapat melakukan *static code analysis* dengan cara melakukan input *source code* atau melakukan *upload file*. *Software engineer* tentunya juga dapat melihat hasil dari *static code analysis* yang dilakukan. *Security engineer* merupakan aktor yang sudah berpengalaman terhadap *vulnerability* yang ada, sehingga *security engineer* merupakan pengguna kakas yang dapat menambahkan *knowledge base* yang dimiliki oleh kakas untuk memperkaya *knowledge* pada kakas.

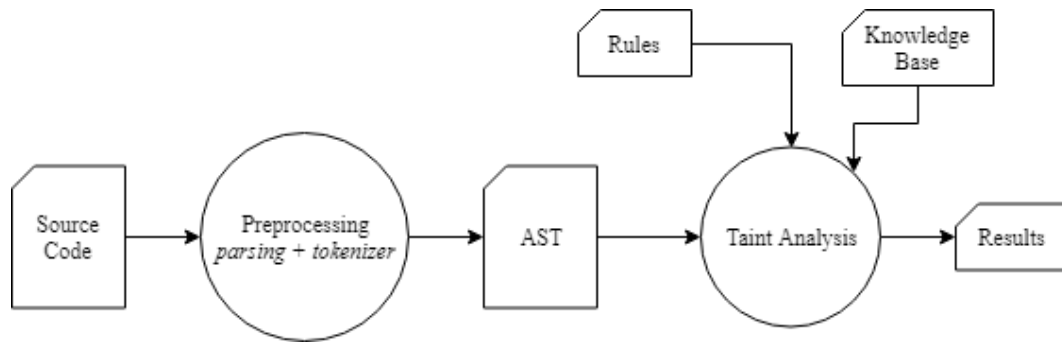
Kebutuhan-kebutuhan fungsionalitas tersebut dapat dilihat secara umum pada Tabel III.2.

Tabel III.2 Diagram *use case* kebutuhan fungsional sistem

No	Komponen	Deskripsi Kebutuhan
1	Kakas mampu melakukan <i>static code analysis</i>	Fungsionalitas ini merupakan kebutuhan utama dari kakas <i>static code analysis</i> yang dibangun. Kakas diharapkan dapat mendeteksi kerentanan jika terdapat <i>vulnerability</i> pada <i>source code</i> yang diberikan
2	Kakas memiliki fitur <i>input source code</i>	Pengguna dapat mengetikkan atau menempelkan <i>source code</i> pada kakas untuk selanjutnya dilakukan <i>static code analysis</i>
3	Kakas memiliki fitur <i>upload file</i>	Pengguna dapat melakukan <i>upload file</i> dengan tipe ekstensi tertentu (php dan zip). Dengan fungsi <i>upload file</i> , pengguna juga dapat melakukan <i>upload multiple file</i> . Ketika pengguna menggunakan fitur <i>upload file</i> dengan tipe file zip, maka pengguna dapat memilih <i>index file</i> .
4	Kakas dapat menampilkan hasil <i>static code analysis</i>	Pengguna dapat melihat hasil dari <i>static code analysis</i> . Terdapat dua kemungkinan yaitu tidak ada <i>vulnerability</i> yang terdeteksi, atau <i>vulnerability</i> terdeteksi.
5	<i>Security Engineer</i> dapat menambahkan <i>knowledge base</i>	Pengguna ( <i>security engineer</i> ) dapat menambahkan <i>knowledge</i> yang ada pada kakas dengan merubah <i>file config</i> pada kakas

### III.5 Analisis Alur Kerja Kakas

Berdasarkan pada subbab-subbab sebelumnya, gambaran umum alur kerja dari kakas yang dibangun akan diilustrasikan dalam bentuk *flowchart*, ilustrasi tersebut dapat dilihat pada Gambar III.21.



Gambar III.21 Alur kerja kakas *static code analysis*

Pada pengerjaan Tugas Akhir ini, yang menjadi fokus utama yaitu pada proses AST ke Taint Analysis. Hal tersebut dikarenakan pada proses *preprocessing* dilakukan dengan bantuan kakas *open source* PHPLY. Selain itu, *taint analysis* juga menerima masukan dari *rules* serta *knowledge base* yang sudah dijelaskan pada subbab-subbab sebelumnya. Dari ketiga masukan tersebut, diharapkan kakas dapat menghasilkan hasil pendeteksian *vulnerability* yang tepat.

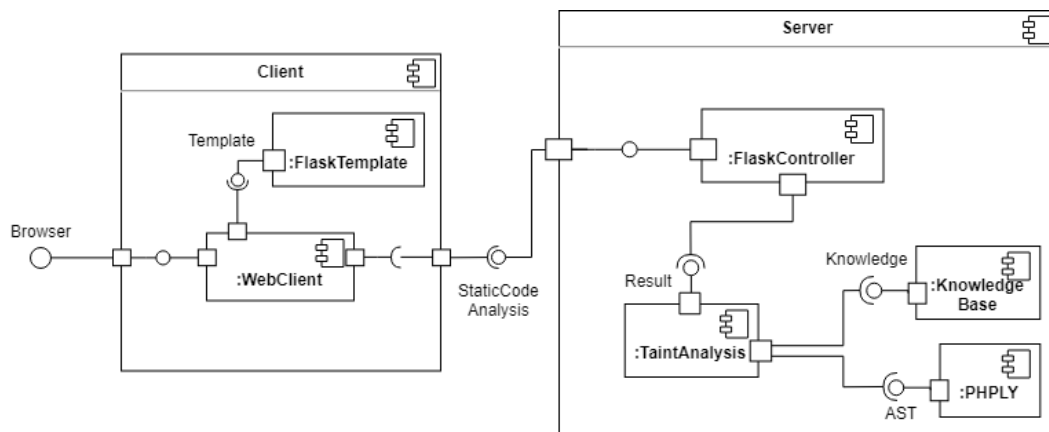
## BAB IV

### IMPLEMENTASI SOLUSI DAN PENGUJIAN

Bab ini berisi penjelasan terkait implementasi solusi dari analisis permasalahan Tugas Akhir. Implementasi solusi meliputi arsitektur serta kebutuhan kakas berdasarkan penjelasan pada Bab III. Selain itu, bab ini juga berisi penjelasan terkait pengujian kakas. Pengujian kakas meliputi tujuan pengujian, hasil pengujian, serta kesimpulan dari hasil pengujian.

#### IV.1 Implementasi Solusi

Implementasi arsitektur dilakukan sesuai dengan rancangan analisis alur kerja kakas yang telah dijelaskan pada subbab III.5 seperti pada Gambar III.21. Seperti yang telah dijelaskan sebelumnya, fokus pengerjaan Tugas Akhir ini berada ada komponen Taint Analysis saja, *preprocessing* dilakukan dengan bantuan kakas *open source* PHPLY. Berikut implementasi arsitektur kakas berkaitan dengan teknologi yang digunakan yang digambarkan dengan menggunakan komponen diagram.



Gambar IV.1 Implementasi arsitektur kakas dan teknologi yang digunakan

Berdasarkan pada Gambar IV.1, hasil implementasi arsitektur kakas berupa sebuah aplikasi web yang menggunakan prinsip *client-side* dan *server-side* dengan

menggunakan teknologi Flask pada sisi *client* dan *server*. Pada *client-side*, digunakan *library flask-materialize* untuk memudahkan dalam melakukan implementasi *front end*. Pada *server-side* bagian *controller*, terdapat dua fitur berbeda yaitu *input source code*, dan *upload file*. Kemudian dari *source code* atau *file* yang diberikan, dilakukan *taint analysis* yang akan menghasilkan hasil berupa ada atau tidaknya *vulnerability* yang dideteksi. Selain itu, *taint analysis* juga dilakukan dengan bantuan PHPLY dan *knowledge base* yang dimiliki.

**Taint analysis** diimplementasikan dengan bahasa pemrograman python. Dari *abstract syntax tree* yang diberikan, pertama-tama program melakukan iterasi untuk tiap *node* sesuai dengan urutan baris pada *source code*. Pada tiap iterasi, terlebih dahulu dilakukan pengecekan tipe *node* sesuai dengan yang telah dijelaskan pada Bab III. Potongan kode untuk proses ini disajikan pada Gambar IV.2.

```
1 # Check node type
2 def processNode(node):
3     node_types = node[0]
4     node_fields = node[1]
5     if node_types != "InlineHTML":
6         if node_types == "Assignment":
7             processAssignment(node_fields)
8         elif node_types == "FunctionCall":
9             processFuncCall(node_fields)
10        elif node_types == "Function":
11            processFunction(node_fields)
12        # ...dst
13
```

Gambar IV.2 Potongan kode untuk pemrosesan tiap *node*

Selanjutnya, tiap *node* akan diproses oleh masing-masing fungsi yang bersesuaian. Implementasi dilakukan sesuai dengan apa yang telah dianalisis pada Bab III. Untuk tiap tipe *node* akan disajikan penjelasan proses masing-masing fungsi dalam bentuk *pseudocode*. *Pseudocode* dapat dilihat pada Gambar IV.3 sampai dengan Gambar IV.12 di bawah.

```

1 ▾ def processAssignment(expr):
2     # I.S. deklarasi variable dengan default value
3     # F.S. variable dengan parameter yang sesuai
4     #
5     # variable = [
6     #     tainted: False,
7     #     sanitized: False
8     # ]
9     # IF tipe dari assignment IN [variable, array, dll..]
10    # IF dari user input
11    #     variable["tainted"] = True
12    # ELSE IF assignment oleh fungsi sanitizers
13    #     variable["sanitized"] = True
14    # RETURN variable
15

```

Gambar IV.3 *Pseudocode* untuk pemrosesan *node* Assignment

```

1 ▾ def processFuncCall(expr):
2     # I.S nama fungsi dan sinks terdefinisi
3     # F.S fungsi di-scan,
4     #     tambahkan vuln jika terdeteksi
5     #
6     # IF nama fungsi ada pada daftar fungsi
7     #     FOREACH node
8     #         process node
9     # ELSEIF nama fungsi IN sinks
10    #     IF parameter fungsi tainted AND not sanitized
11    #         add vulnerability
12

```

Gambar IV.4 *Pseudocode* untuk pemrosesan *node* FunctionCall

```

1 ▾ def processFunction(expr):
2     # I.S node fungsi terdefinisi
3     # F.S node fungsi ditambahkan ke list fungsi
4     #
5     # RETURN function node (untuk ditambahkan ke daftar fungsi)
6

```

Gambar IV.5 *Pseudocode* untuk pemrosesan *node* Function

```

1 ▾ def processInclude(expr):
2     # I.S. filename terdefinisi
3     # F.S. file diproses, jika ada.
4     #
5     # IF filename EXIST
6     #   read file
7     #   bangun AST pada file yang diinclude
8     #   proses tiap node pada AST yang baru
9     .

```

Gambar IV.6 *Pseudocode* untuk pemrosesan *node* Include

```

1 ▾ def processGlobal(expr):
2     # I.S. variable global terdefinisi
3     # F.S. variable lokal = variable global
4     #
5     # panggil variable global ke dalam variable fungsi
6     .

```

Gambar IV.7 *Pseudocode* untuk pemrosesan *node* Global

```

1 ▾ def processReturn(expr):
2     # I.S. parameter return terdefinisi
3     # F.S. pemanggil fungsi mendapatkan hasil
4     #
5     # lakukan processAssignment dengan variable yang di-return
6     .

```

Gambar IV.8 *Pseudocode* untuk pemrosesan *node* Return

```

1 ▾ def processIf(expr):
2     # I.S. tidak ada
3     # F.S. semua node pada cabang IF diproses
4     #
5     # proses node pada kasus if terpenuhi
6     # proses node pada kasus elseif terpenuhi (jika ada)
7     # proses node pada kasus else terpenuhi (jika ada)
8     .

```

Gambar IV.9 *Pseudocode* untuk pemrosesan *node* If



```

1 ▾ def processSwitch(expr):
2     # I.S. tidak ada
3     # F.S. semua node diproses sesuai dengan kondisi
4     #
5     # IF variable_switch in user input
6     #   process node pada semua kasus switch
7     # ELSEIF ada nilai dari variable switch yang terpenuhi
8     #   process node pada kasus yang terpenuhi
9     # ELSE
10    #   process node pada kasus default switch
11

```

Gambar IV.10 *Pseudocode* untuk pemrosesan *node* Switch

```

1 ▾ def processFor(expr):
2     # I.S. tidak ada
3     # F.S. semua node pada FOR diproses
4     #
5     # ambil nilai awal loop
6     # process tiap node pada for dengan..
7     # ..nilai iterasi adalah nilai awal loop
8

```

Gambar IV.11 *Pseudocode* untuk pemrosesan *node* For

```

1 ▾ def processForEach(expr):
2     # I.S. tidak ada
3     # F.S. semua node pada FOREACH diproses
4     #
5     # mapping tiap nilai yang ada di array..
6     # ..dengan key-variable yang diberikan
7     # process tiap node pada foreach dengan..
8     # ..variable yang sudah dimapping
9

```

Gambar IV.12 *Pseudocode* untuk pemrosesan *node* ForEach

## IV.2 Implementasi Kebutuhan

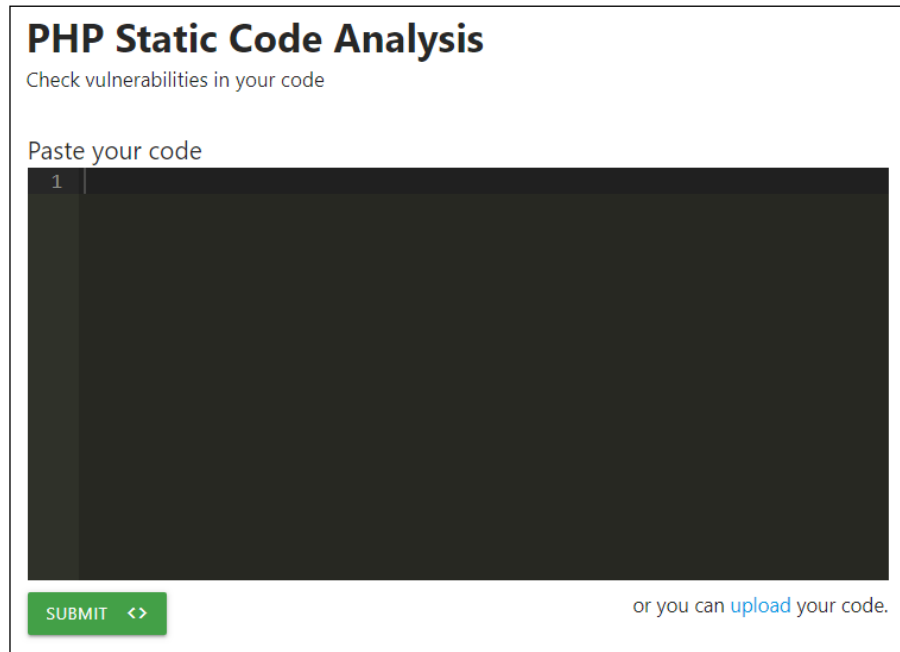
Berdasarkan pada Bab III, kebutuhan kakas terdiri dari kebutuhan fungsional. Implementasi yang dilakukan berupa fitur atau aturan yang diterapkan pada aplikasi kakas. Satu kebutuhan bisa menjadi beberapa fitur terpisah, dan sebaliknya, satu fitur dapat mencakup beberapa kebutuhan sekaligus. Kebutuhan fungsional yang telah dilakukan yaitu:

1. Kakas mampu melakukan *static code analysis*

Pada kebutuhan fungsionalitas yang pertama, implementasi yang dilakukan yaitu membangun sebuah aplikasi web secara umum dengan bantuan *framework* Flask pada sisi *client-side* serta *server-side*. Aplikasi dapat menerima masukan dari pengguna berupa *source code* ataupun *file* dengan ekstensi php atau zip. Setelah itu, kakas melakukan *static code analysis* dengan metode *taint analysis* yang sudah dijelaskan pada Bab sebelumnya. Terakhir, aplikasi web dapat menampilkan hasil dari *static code analysis*.

2. Kakas memiliki fitur *input source code*

Implementasi pada kebutuhan fungsionalitas berikutnya yaitu dengan menambahkan suatu *field textarea* yang dapat melakukan *highlight* pada *source code* dengan ekstensi PHP. Selain itu, apabila terdapat *syntax error*, program dapat menampilkan simbol silang yang menandakan *error*. Fitur ini berada pada halaman utama kakas, tujuan dari *input source code* yaitu jika pengguna ingin melakukan *static code analysis* pada *single file*. Selanjutnya ketika pengguna menekan tombol *submit*, kakas melakukan *taint analysis*. Tampilan antarmuka pada fitur ini dapat dilihat pada Gambar IV.13.

The image shows a web application titled "PHP Static Code Analysis" with the subtitle "Check vulnerabilities in your code". Below the title is a text input area labeled "Paste your code" with a dark background and a light-colored cursor. A green "SUBMIT" button with a code icon is located at the bottom left of the input area. To the right of the button, there is a link that says "or you can upload your code.".

Gambar IV.13 Tampilan antarmuka *input source code*

3. Kakas memiliki fitur *upload file*

Implementasi pada kebutuhan fungsionalitas berikutnya yaitu dengan menambahkan suatu *field input* bertipe *file* yang berfungsi untuk melakukan *upload file*. *File* yang dapat diunggah adalah *file* dengan ekstensi php atau zip. Apabila pengguna mengunggah ekstensi selain php atau zip maka program akan menolak untuk diproses lebih lanjut. Fitur ini berada pada halaman *upload*. Tujuan dari fitur *upload file* yaitu jika pengguna ingin melakukan *static code analysis* dengan *multiple file*. *Static code analysis* dengan *multiple file* dapat dilakukan dengan mengunggah *file* dengan ekstensi zip. Selanjutnya ketika pengguna menekan tombol *submit*, kakas terlebih dahulu memberikan daftar *file* yang sudah di-*extract* lalu memberikan pilihan kepada pengguna untuk dipilih sebagai "*index*" *file*. Setelah itu kemudian dilakukan *taint analysis*. Tampilan antarmuka pada fitur ini dapat dilihat pada Gambar IV.14 dan Gambar IV.15.

**PHP Static Code Analysis**  
Check vulnerabilities in your code

Select, or Drag n Drop your code

or you can [paste](#) your code.

SUBMIT <>

Gambar IV.14 Tampilan antarmuka *upload file*

**PHP Static Code Analysis**  
Check vulnerabilities in your code

Select 'index' file:

☐ another\_require.php

☒ example\_.php

☐ includedfile.php

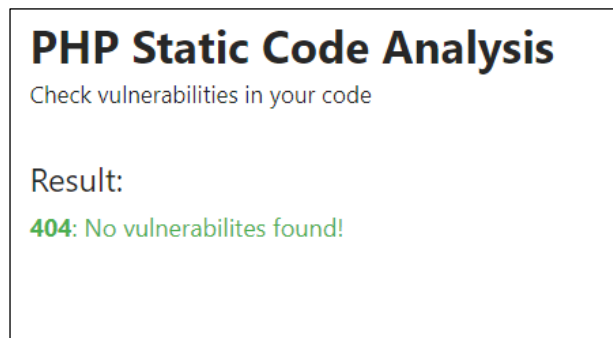
SUBMIT <>

Gambar IV.15 Tampilan antarmuka *select 'index' file*

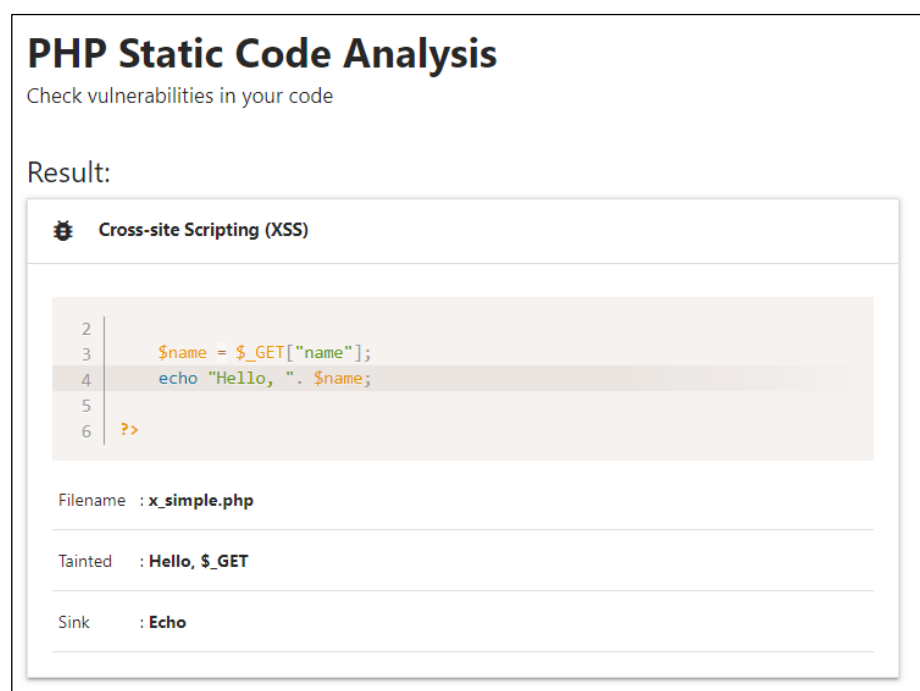
4. Kakas dapat menampilkan hasil *static code analysis*

Implementasi pada kebutuhan fungsionalitas berikutnya yaitu dengan menampilkan hasil *taint analysis* dari *source code* yang telah dimasukan. Terdapat dua tampilan untuk fitur ini. Tampilan pertama yaitu ketika tidak ditemukan adanya *vulnerability*, maka akan ditampilkan tulisan warna hijau yang berartikan aman. Selanjutnya yaitu tampilan ketika ditemukan

*vulnerability*, maka ditampilkan beberapa informasi terkait *vulnerability* yang ditemukan. Informasi-informasi yang ditampilkan meliputi jenis *vulnerability*, nomor baris kode, nama file (jika menggunakan fitur upload file), *tainted variable*, serta *sink function*. Tampilan antarmuka pada fitur ini dapat dilihat pada Gambar IV.16 dan Gambar IV.17.



Gambar IV.16 Tampilan antarmuka ketika tidak ditemukan *vulnerability*



Gambar IV.17 Tampilan antarmuka ketika ditemukan *vulnerability*

5. *Security Engineer* dapat menambahkan *knowledge base*

Implementasi pada fitur ini yaitu dengan adanya *file config* yang terpisah dari *file* utama yang melakukan *taint analysis*. *Config* terdiri dari tiga *file* yang berbeda, yaitu *Config-Securing* merupakan daftar fungsi yang termasuk *sanitizers*, *Config-Sink* merupakan daftar fungsi yang termasuk *sinks*, serta *Config-Source* yang merupakan daftar *variable* bawaan yang dapat menyebabkan sebuah *variable* menjadi *tainted*.

### IV.3 Pengujian

Berdasarkan pada hasil implementasi pada subbab sebelumnya, dilakukan pengujian untuk memastikan keberhasilan kakas yang dibangun dalam mendeteksi adanya kerentanan. Selanjutnya dijelaskan terkait pengujian yang dilakukan sesuai dengan yang telah dijelaskan pada Bab III.3. Pada subbab ini juga dijelaskan terkait tujuan dari pengujian dan kesimpulan dari hasil pengujian.

#### IV.3.1 Tujuan Pengujian

Setelah dilakukan implementasi solusi, kakas *static code analysis* diuji untuk memastikan keberhasilan kakas. Tujuan utama dalam pengujian yaitu implementasi kakas berhasil mendeteksi adanya kerentanan dengan metode *taint analysis*.

#### IV.3.2 Lingkungan Pengujian

Lingkungan pengujian kakas *static code analysis* menggunakan perangkat keras sebagai berikut.

1. *Processor* Intel Core i7-7500 @ 2.70Ghz (4 CPUs)
2. *Memory* DDR3 8 GB
3. *Harddisk* 1 TB

Selain perangkat keras, lingkungan pengujian kakas *static code analysis* menggunakan perangkat lunak sebagai berikut.

1. Sistem operasi Windows 10 64-bit
2. Web browser Google Chrome versi 76.0.3809.100

### IV.3.3 Pengujian *Taint Analysis*

Sebelum melakukan pengujian pada aplikasi yang kompleks, terlebih dahulu dilakukan rancangan *source code* sederhana yang sudah didesain terdapat *vulnerability* di dalamnya. Terdapat tiga komponen penting dalam pengujian kakas. Ketiga komponen tersebut adalah kebenaran daftar *variable*, kebenaran *control flow* program, dan kebenaran pendeteksian *vulnerability*. Pada pengujian dengan *source code* sederhana yang sudah didesain seperti yang dapat dilihat pada Lampiran B, kakas telah berhasil menguji ketiga komponen yang sudah disebutkan. Selanjutnya dilakukan pengujian dengan *open source* PHP project yang memiliki CWE (*Common Weakness Enumeration*) yang berhubungan dengan Injection dan Cross-site Scripting. Dari CWE ID yang ada, dilakukan pendeteksian pada CVE ID yang bersesuaian. Daftar aplikasi yang diuji dapat dilihat pada Tabel IV.1.

Tabel IV.1 Tabel daftar CVE ID yang diuji.

No	CWE ID	CVE ID	Nama Aplikasi
1	79	CVE-2019-1010287	Timesheet Next Gen
2	79	CVE-2019-13186	MiniCMS
3	79	CVE-2019-13339	MiniCMS
4	79	CVE-2019-13340	MiniCMS
5	79	CVE-2019-13341	MiniCMS
6	79	CVE-2019-11359	I, Librarian
7	79	CVE-2019-11428	I, Librarian
8	79	CVE-2019-11449	I, Librarian
9	79	CVE-2019-9839	VFront
10	77	CVE-2019-12585	Pfsense
11	78	CVE-2018-1000019	OpenEMR
12	78	CVE-2017-1000235	I, Librarian
13	78	CVE-2016-10709	Pfsense
14	89	CVE-2019-1010153	Zzcms
15	89	CVE-2019-1010104	TechyTalk Quick Chat
16	89	CVE-2019-13578	GiveWP Give plugin
17	89	CVE-2019-13575	WPEverest Everest Forms
18	89	CVE-2019-13086	CSZ CMS
19	89	CVE-2019-14529	OpenEMR
20	89	CVE-2019-14313	10Web Photo Gallery

#### IV.3.4 Kesimpulan Hasil Pengujian

Dari daftar CVE pada subbab sebelumnya, diperoleh hasil yang cukup memuaskan. 12 dari 20 *vulnerability* berhasil dideteksi oleh kakas. Selain itu, dari delapan yang tidak ditemukan, tiga diantaranya membutuhkan teknik khusus, yaitu melakukan *bypass* terhadap fungsi *sanitizers*. Dua membutuhkan Object Oriented Programming. Dan sisanya dikarenakan fungsi yang tidak diketahui oleh kakas. Untuk lebih jelasnya, dapat dilihat pada Tabel IV.2 Hasil pengujian.

Tabel IV.2 Tabel hasil pengujian

No	CVE ID	Found	Penjelasan
1	CVE-2019-1010287	V	<i>Vulnerability</i> terdeteksi
2	CVE-2019-13186	V	<i>Vulnerability</i> terdeteksi
3	CVE-2019-13339	V	<i>Vulnerability</i> terdeteksi
4	CVE-2019-13340	V	<i>Vulnerability</i> terdeteksi
5	CVE-2019-13341	V	<i>Vulnerability</i> terdeteksi
6	CVE-2019-11359	V	<i>Vulnerability</i> terdeteksi
7	CVE-2019-11428	V	<i>Vulnerability</i> terdeteksi
8	CVE-2019-11449	V	<i>Vulnerability</i> terdeteksi
9	CVE-2019-9839	X	Parameter <i>tainted</i> pada fungsi yang tidak diketahui
10	CVE-2019-12585	V	<i>Vulnerability</i> terdeteksi
11	CVE-2018-1000019	V	<i>Vulnerability</i> terdeteksi
12	CVE-2017-1000235	X	Diperlukan teknik khusus ( <i>insert payload "  &lt;os-commands&gt;  "</i> )
13	CVE-2016-10709	X	Diperlukan teknik khusus ( <i>bypass via ' ' character</i> )
14	CVE-2019-1010153	V	<i>Vulnerability</i> terdeteksi
15	CVE-2019-1010104	X	Diperlukan teknik khusus ( <i>bypass fungsi like_escape</i> )
16	CVE-2019-13578	X	Tidak didukung oleh kakas (Object Oriented Programming)
17	CVE-2019-13575	X	Tidak didukung oleh kakas (Object Oriented Programming)
18	CVE-2019-13086	V	<i>Vulnerability</i> terdeteksi
19	CVE-2019-14529	X	SyntaxError (Recursive ArrayOffset)
20	CVE-2019-14313	X	Tidak didukung oleh kakas ( <i>database controller</i> )



## BAB V

### KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan dan saran dari pengerjaan Tugas Akhir. Kesimpulan meliputi hasil yang didapatkan dari pengerjaan Tugas Akhir, serta pelajaran yang didapatkan setelah mengerjakan Tugas Akhir. Saran meliputi hal-hal yang dapat ditingkatkan pada solusi dari permasalahan Tugas Akhir untuk penelitian lebih lanjut.

#### V.1 Kesimpulan

Pada Tugas Akhir ini telah dibuat sebuah kakas *static code analysis* dengan metode *taint analysis* dalam mendeteksi *vulnerability* pada web. Kakas dibangun pada platform web. Berdasarkan implementasi serta pengujian yang dilakukan dapat diambil beberapa kesimpulan, diantaranya:

1. *Vulnerability* yang dapat dideteksi oleh *taint analysis* merupakan *vulnerability* bertipe injection yang terdaftar pada OWASP Top Ten 2017 yaitu A1 Injection serta A7 Cross-Site Scripting (XSS).
2. *Sources*, *sanitizers*, dan *sinks* merupakan *knowledge base* yang dapat dimodifikasi. Hal ini dapat dilakukan oleh *security engineer* yang sudah berpengalaman.
3. Kakas belum mendukung Object Oriented Programming (OOP) sehingga masih terdapat *False Negative*. Namun, kakas tetap berhasil mendeteksi *vulnerability* pada file yang menggunakan fitur OOP dikarenakan *vulnerability* yang ditemukan ada pada bagian yang tidak menggunakan fitur OOP.
4. Pendekatan dengan *static code analysis* terbatas pada pembacaan *source code* saja sehingga apabila terdapat *sinks* yang disembunyikan maka tidak dapat dideteksi oleh kakas. Hal tersebut membutuhkan eksekusi *source code*

yang dimana hal tersebut sudah termasuk kedalam metode *dynamic analysis*.

5. Kakas berhasil mendeteksi celah keamanan apabila pada *source code* sudah jelas ada *tainted variable* yang dieksekusi oleh *sinks* tanpa melalui fungsi *sanitizer* terlebih dahulu. Sejumlah fungsi *sanitizer* tidak efektif dalam “membersihkan” masukan yang berbahaya, yang menyebabkan terdapat beberapa kasus *vulnerability* tetap terjadi, sehingga masih terdapat *False Negative*.

## V.2 Saran

Berdasarkan hasil pengujian kakas *static code analysis*, terdapat beberapa hal yang dapat ditingkatkan serta analisis lebih lanjut. Beberapa hal yang dapat dilakukan dalam meningkatkan performa maupun hasil dari kakas yaitu:

1. Kakas dapat mendukung Object Oriented Programming (OOP) sehingga penemuan *vulnerability* pada *source code* dapat lebih akurat. Saat ini kakas yang dibangun belum mendukung OOP sehingga masih belum maksimal dalam menemukan *vulnerability*.
2. Kakas dapat mendukung *framework* yang banyak digunakan, seperti misalnya CodeIgniter, Wordpress, dan lain-lain.
3. Menambahkan *knowledge base* yang mungkin dalam menemukan *vulnerability*. Salah satu aspek terpenting dalam menemukan *vulnerability* yaitu berdasarkan *knowledge* yang dimiliki oleh kakas.
4. Menambahkan tipe *vulnerability* yang dapat dideteksi oleh kakas. Saat ini, tipe yang dapat dideteksi oleh kakas yaitu Cross-Site Scripting, SQL Injection, OS Command Injection. Diharapkan kedepannya kakas dapat mendeteksi lebih banyak tipe *vulnerability*.
5. Optimisasi dalam melakukan pendeteksian *vulnerability*, khususnya pada segi waktu. Apabila *source code* yang diupload berukuran besar, maka akan memakan waktu yang cukup lama.

6. Kakas dapat diimplementasikan dalam bentuk ekstension di salah satu code editor (Visual Studio Code), sehingga *developer* dapat mengetahui secara *real time* jika terdapat *vulnerability* pada source code yang sedang dibangun.

## DAFTAR REFERENSI

- Jurásek, P. (2018). *PHPWander: A Static Vulnerability Analysis Tool for PHP*. Thesis dissertation, University of Oslo, Faculty of Mathematics and Natural Sciences, Department of Informatics.
- Chess, M., & West, J. (2007). *Secure Programming with Static Analysis*. Pearson Education, Inc.
- OWASP (2017). *The Ten Most Critical Web Application Security Risks*. The OWASP Foundation.
- Dahse, J. (2010). *RIPS – A static source code analyser for vulnerabilities in PHP scripts*. Seminar Work at Chair for Network and Data Security, Horst Görtz Institute & Ruhr-University Bochum.
- OWASP (updated 2018). Cross-site Scripting (XSS) [online]. [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)) (diakses 21 Oktober 2018).
- The Linux Information Project (updated 2006). *Source code definition* [online]. [http://www.linfo.org/source\\_code.html](http://www.linfo.org/source_code.html) (diakses 21 Oktober 2018).
- Shirey, R. (2000). RFC 2828 – Internet Security Glossary [online]. <https://tools.ietf.org/html/rfc2828> (diakses 21 Oktober 2018).
- Dahse, J. (2016). *Introducing the RIPS analysis engine* [online]. <https://blog.ripstech.com/2016/introducing-the-rips-analysis-engine/> (diakses 21 Oktober 2018).
- Schwartz, E., & Brumley, D. (2010). *All You Ever Wanted to Know About Dynamic Taint Analysis & Forward Symbolic Execution (but might have been afraid to ask)*. Carnegie Mellon University.
- Clause, J., dkk. (2007). *Dytan: A Generic Dynamic Taint Analysis Framework*. College of Computing, Georgia Institute of Technology.
- Dahsem J. (2016). *Static Code Analysis of Complex PHP Application Vulnerabilities*. AppSecEU 2016. Rome, Italy.
- Shannon M. (2018). *Is your code tainted? Finding security vulnerabilities using taint tracking*. EuroPython Conference 2018. Edinburgh, United Kingdom.

## Lampiran A. Daftar *Knowledge Base*

Lampiran ini berisikan daftar *knowledge base* yang digunakan dalam pengimplementasian kakas *static code analysis*. *Knowledge* ini digunakan sebagai dasar penentuan *sources*, *sinks*, dan *sanitizers* pada *source code* yang diberikan. Berikut merupakan *knowledge base* untuk setiap komponen.

### A.1 Daftar *Sources*

```
#!/usr/bin/env python

USER_INPUT = [
    "$_GET",
    "$_POST",
    "$_COOKIE",
    "$_REQUEST",
    "$_FILES",
    "$_SERVER",
    "$HTTP_GET_VARS",
    "$HTTP_POST_VARS",
    "$HTTP_COOKIE_VARS",
    "$HTTP_REQUEST_VARS",
    "$HTTP_POST_FILES",
    "$HTTP_SERVER_VARS",
    "$HTTP_RAW_POST_DATA",
    "$argc",
    "$argv"
]

SERVER_PARAM = [
    "HTTP_ACCEPT",
    "HTTP_ACCEPT_LANGUAGE",
    "HTTP_ACCEPT_ENCODING",
    "HTTP_ACCEPT_CHARSET",
    "HTTP_CONNECTION",
    "HTTP_HOST",
    "HTTP_KEEP_ALIVE",
    "HTTP_REFERER",
    "HTTP_USER_AGENT",
    "HTTP_X_FORWARDED_FOR",
    "PHP_AUTH_DIGEST",
    "PHP_AUTH_USER",
    "PHP_AUTH_PW",
    "AUTH_TYPE",

```

```
"QUERY_STRING",
"REQUEST_METHOD",
"REQUEST_URI",
"PATH_INFO",
"ORIG_PATH_INFO",
"PATH_TRANSLATED",
"REMOTE_HOSTNAME",
"PHP_SELF"
]
```

```
FILE = [
    "bzip2_read",
    "dio_read",
    "exif_imagetype",
    "exif_read_data",
    "exif_thumbnail",
    "fgets",
    "fgetss",
    "file",
    "file_get_contents",
    "fread",
    "get_meta_tags",
    "glob",
    "gzread",
    "readdir",
    "read_exif_data",
    "scandir",
    "zip_read",
]
```

```
DB = [
    "mysql_fetch_array",
    "mysql_fetch_assoc",
    "mysql_fetch_field",
    "mysql_fetch_object",
    "mysql_fetch_row",
    "pg_fetch_all",
    "pg_fetch_array",
    "pg_fetch_assoc",
    "pg_fetch_object",
    "pg_fetch_result",
    "pg_fetch_row",
    "sqlite_fetch_all",
    "sqlite_fetch_array",
    "sqlite_fetch_object",
    "sqlite_fetch_single",
]
```

```
"sqlite_fetch_string",  
]  
  
ALL = USER_INPUT + SERVER_PARAM + FILE + DB
```

## A.2 Daftar Sanitizers

```
#!/usr/bin/env python  
  
XSS = [  
    "htmlentities",  
    "htmlspecialchars",  
    "highlight_string",  
]  
  
SQL = [  
    "addslashes",  
    "dbx_escape_string",  
    "db2_escape_string",  
    "ingres_escape_string",  
    "maxdb_escape_string",  
    "maxdb_real_escape_string",  
    "mysql_escape_string",  
    "mysql_real_escape_string",  
    "mysqli_escape_string",  
    "mysqli_real_escape_string",  
    "pg_escape_string",  
    "pg_escape_bytea",  
    "sqlite_escape_string",  
    "sqlite_udf_encode_binary",  
    "cubrid_real_escape_string",  
]  
  
OS = [  
    "escapeshellarg",  
    "escapeshellcmd"  
]  
  
STRING = [  
    "intval",  
    "floatval",  
    "doubleval",  
    "filter_input",  
    "urlencode",  
    "rawurlencode",  
    "round",
```

```
"floor",
"strlen",
"strrpos",
"strpos",
"strftime",
"strptime",
"md5",
"md5_file",
"sha1",
"sha1_file",
"crypt",
"crc32",
"hash",
"mhash",
"hash_hmac",
"password_hash",
"mcrypt_encrypt",
"mcrypt_generic",
"base64_encode",
"ord",
"sizeof",
"count",
"bin2hex",
"levenshtein",
"abs",
"bindec",
"decbin",
"dechex",
"decoct",
"hexdec",
"rand",
"max",
"min",
"metaphone",
"tempnam",
"soundex",
"money_format",
"number_format",
"date_format",
"filetype",
"nl_langinfo",
"bzcompress",
"convert_uuencode",
"gzdeflate",
"gzencode",
"gzcompress",
```



```
"http_build_query",
"lzf_compress",
"zlib_encode",
"imap_binary",
"iconv_mime_encode",
"bson_encode",
"sqlite_udf_encode_binary",
"session_name",
"readlink",
"getservbyport",
"getprotobynumber",
"gethostname",
"gethostbyname1",
"gethostbyname",
]
```

ALL = XSS + SQL + STRING + OS

### A.3 Daftar Sinks

```
#!/usr/bin/env python
```

```
XSS = [
    "Echo",
    "Print",
    "print_r",
    "exit",
    "die",
    "printf",
    "vprintf",
    "trigger_error",
    "user_error",
    "odbc_result_all",
    "ovrimos_result_all",
    "ifx_htmltbl_result",
]
```

```
OS = [
    "assert",
    "create_function",
    "eval",
    "mb_ereg_replace",
    "mb_eregi_replace",
    "preg_filter",
    "preg_replace",
    "preg_replace_callback",
]
```

```
"backticks",
"exec",
"expect_popen",
"passthru",
"pcntl_exec",
"popen",
"proc_open",
"shell_exec",
"system",
"mail",
"mb_send_mail",
"w32api_invoke_function",
"w32api_register_function"
]
```

```
SQL = [
    "dba_open",
    "dba_popen",
    "dba_insert",
    "dba_fetch",
    "dba_delete",
    "dbx_query",
    "odbc_do",
    "odbc_exec",
    "odbc_execute",
    "db2_exec",
    "db2_execute",
    "fbsql_db_query",
    "fbsql_query",
    "ibase_query",
    "ibase_execute",
    "ifx_query",
    "ifx_do",
    "ingres_query",
    "ingres_execute",
    "ingres_unbuffered_query",
    "msql_db_query",
    "msql_query",
    "msql",
    "mssql_query",
    "mssql_execute",
    "mysql_db_query",
    "mysql_query",
    "mysql_unbuffered_query",
    "mysqli_stmt_execute",
    "mysqli_query",

```

```
"mysqli_real_query",
"mysqli_master_query",
"oci_execute",
"ociexecute",
"ovrimos_exec",
"ovrimos_execute",
"ora_do",
"ora_exec",
"pg_query",
"pg_send_query",
"pg_send_query_params",
"pg_send_prepare",
"pg_prepare",
"sqlite_open",
"sqlite_popen",
"sqlite_array_query",
"arrayQuery",
"singleQuery",
"sqlite_query",
"sqlite_exec",
"sqlite_single_query",
"sqlite_unbuffered_query",
"sybase_query",
"sybase_unbuffered_query",
"query",
]
```

ALL = XSS + SQL + OS

## Lampiran B. Daftar *Source Code*

Lampiran ini berisikan daftar *source code* sederhana yang dirancang dan digunakan dalam pengujian kakas *static code analysis*. *Source code* dirancang sedemikian rupa sehingga terdapat *vulnerability*. Berikut merupakan *source code* yang dirancang.

### B.1 *Source code x\_xss.php*

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php

echo $_GET["lol"]; # vuln
echo htmlentities($_GET["lol"]); # safe

echo "a" + $_GET["lol"]; # vuln
echo "a" + htmlentities($_GET["lol"]); # safe

echo $_GET["lol"] + "a"; # vuln
echo htmlentities($_GET["lol"]) + "a"; # safe

echo htmlentities($_GET["lol"]) + $_GET["hehe"]; # vuln
echo ($_GET["lol"]) + htmlentities($_GET["hehe"]); # vuln

$user = $_GET["user"];
print $user; # vuln
print md5($user); # safe

$usersafe = htmlentities($user);
echo $usersafe + "ea";

$cookiee = htmlentities($_COOKIE["enak"]);
echo $cookiee;

$arr1 = ['ea'=>$_GET["hehe"], "hehe"];
echo $arr1['ea']; # vuln

$arr2 = array(0, 1, 2, $_GET["hehe"]);
echo $arr2[3]; # vuln

print_r($_GET["lol"]); # vuln
```

?>

<?=\$\_GET['hehe'];?><!-- vuln -->

## B.2 Source code x\_switch.php

```
<?php

# TC 1 - Vuln
$favcolor = "blue";
switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is ".$_GET["haha"];
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}

# TC 2 - Safe
$favcolor = "yellow";
switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is ".$_GET["haha"];
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}

# TC 3 - Safe
$fav = ["red", "green", "blue"];
switch ($fav[0]) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is ".$_GET["haha"];
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
```

```

}

# TC 4 - Vuln
$fav = ["red", "green", "blue"];
switch ($fav[2]) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is ".$_GET["haha"];
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}

# TC 5 - Vuln
switch ($_GET["hehe"]) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is ".$_GET["haha"];
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}

?>

```

### B.3 Source code x\_sql.php

```

<?php

mysql_connect('localhost','root','root');
mysql_select_db('hp');

$user = $_POST['user'];
$pass = $_POST['password'];
$query = "select * from zend_adminlist where user_name = '$user' and
password = '$pass'";
$re = mysql_query($query);

?>

```

#### B.4 Source code x\_os.php

```
<?php

# Vuln
exec("ping -c 1 " . $_GET['host'], $output);
print_r($output);

# Safe
exec("ping -c 1 " . escapeshellarg($_GET['host']), $output);
print_r($output);

?>
```

#### B.5 Source code x\_if\_elseif\_else.php

```
<?php

# TC 1 - All Safe
$satu = $_GET["test"];
$satu = htmlentities($satu);
if ($_GET["id"]){
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
    $satu = htmlentities($satu);
} elseif (true) {
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
    $satu = htmlentities($satu);
} else {
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
    $satu = htmlentities($satu);
}
echo $dua;
echo $satu;

# TC 2 - vuln cause by if
if ($_GET["id"]){
    $dua = $_GET["id"];
} elseif (true) {
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
} else {
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
}
```

```

}
echo $dua; # vuln

# TC 3 - vuln cause by elseif
if ($_GET["id"]){
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
} elseif (true) {
    $dua = $_GET["id"];
} else {
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
}
echo $dua; # vuln

# TC 4 - vuln cause by else
if ($_GET["id"]){
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
} elseif (true) {
    $dua = $_GET["id"];
    $dua = htmlentities($dua);
} else {
    $dua = $_GET["id"];
}
echo $dua; # vuln

# TC 5 - vuln cause $satu not filtered in if
$satu = $_GET["test"];
if ($_GET["id"]){
    // do something else
} elseif (true) {
    $satu = htmlentities($satu);
} else {
    $satu = htmlentities($satu);
}
echo $satu; # vuln
?>

```

## B.6 Source code x\_func\_funccall.php

```

<?php

// # TC 1 - Safe
function func1() {

```



```

    echo $a;
}
$a = $_GET["hehe"];
func1();

// # TC 2 - Vuln
function func2() {
    global $a;
    echo $a;
}
$a = $_GET["hehe"];
func2();

// # TC 3 - Safe
function func3() {
    global $a;
    $a = htmlentities($a);
    echo $a;
}
$a = $_GET["hehe"];
func3();

// # TC 4 - Safe
function func4() {
    global $a;
    echo $a;
}
$a = $_GET["hehe"];
$a = htmlentities($a);
func4();

// # TC 5 - Safe
function func5() {
    global $a;
    $a = htmlentities($a);
}
$a = $_GET["hehe"];
func5();
echo $a;

# TC 6 - Vuln
function func6() {
    $a = htmlentities($a);
    global $a;
    echo $a;
}

```

```

$a = $_GET["hehe"];
func6();

# TC 7 - Safe
function func7() {
    $a = $_GET["hehe"];
    global $a;
    $a = htmlentities($a);
    echo $a;
}
$a = $_GET["hehe"];
func7();
echo $a;

# TC 8 - Vuln
function func8($param1) {
    echo $param1;
}
$a = $_GET["hehe"];
func8($a);

# TC 9 - Vuln
function func9($param1) {
    echo $param1[0];
}
$a = [$_GET["hehe"], 1];
func9($a);

# TC 10 - Safe
function func10($param1) {
    echo htmlentities($param1[1]);
}
$a = [$_GET["hehe"], 1];
func10($a);

# TC 11 - Vuln
function func11() {
    return $_GET["hehe"];
}
$a = func11();
echo $a;

# TC 12 - Vuln
function func12() {
    $a = $_GET["hehe"];
    return $a;
}

```

```

}
$a = func12();
echo $a;

# TC 13 - Safe
function func13() {
    $a = $_GET["hehe"];
    return htmlentities($a);
}
$a = func13();
echo $a;

# TC 14 - Safe
function func14() {
    return $_GET["hehe"];
}
$a = htmlentities(func14() . 'hehe');
echo $a;

# TC 15 - Vuln
function func15() {
    return $_GET["hehe"];
}
$a = func15() . 'hehe';
echo $a;

?>

```

## B.7 Source code x\_assignment.php

```

<?php

// Normal
$a0 = 31;

// Variable
$b0 = $a0;

// UnaryOp
$c0 = -31;
$c1 = ~31;

// BinaryOp
$d0 = 1 + 2;
$d1 = 3 + 4 * 2;

```

```

$d2 = '1' + '2';
$d3 = (true or false);

// TernaryOp
$um = isset($_GET["um"]) ? $_GET["um"] : "";

// FunctionCall
$e0 = $_GET["lol"];
$e1 = htmlentities($d0);
$e2 = htmlentities($_GET["um"]);

// Normal Array
$a = '1';
$f0 = [1, 2];
$f0[0] = 3;
$f0[1] = $_GET["ahaha"];
$f0[$i] = $i;
$f1 = ["a", "b"];
$f2 = ["a", $a];
$f3 = ["a", $_GET["hehe"]];
$f4 = ["a", htmlentities($a)];
$f5 = ["a", htmlentities($_GET["ha"])];
$f6 = ["a", '1' + $_GET["lol"]];
$f7 = ["a", '1' + htmlentities($_GET["lol"])];

// Array with Key
$g0 = ["b" => 1];
$g0["b"] = 2;
$g1 = ["a" => 1, "b" => 2];
$g2 = [$_GET["um"], 2];
$g3 = [1, "ea" => htmlentities($_GET["um"])];
$g4 = $g3["ea"];
$g5 = ["b", "c"];
$g6 = ["d", "e" => 0];
$g7 = $g0[$g5[$g6["e"]]];

// Nested Array
$a = 0;
$h0 = ["a" => [0, "b" => [1, $_GET["um"]]], "b" => [3]];
$h1 = $h0["a"]["b"][1];
echo $h1; # Vuln

?>

```

## B.8 Source code x\_for\_foreach.php

```
<?php

# For
for ($i=0; $i < 10; $i+=2) {
    echo $i + $_GET["hehe"];
}

# Foreach 1
$a = 3;
$asd = array('a' => 1, $_GET["lol"], 'b' => $a, 4);
foreach ($asd as $k) {
    echo $k; # Vuln
}

# Foreach 2
$asd = array('x' => 1, 'xx' => 2, 'xxx' => 3, 'xxxx' => $_GET["lol"]);
foreach ($asd as $key => $value) {
    echo $key;
    echo $value; # Vuln
}

?>
```