ASEECT
Homepage
https://jurnal.polibatam.ac.id/index.php/ASEECT

Journal of Applied Sciences, Electrical Engineering and Computer Technology

Vol 1 No 3 2020
ISSN Online: 2746-7422

Review Pengujian Keamanan Perangkat Lunak dalam *Software Development Life Cycle* (SDLC)

Lindawani Siregar[1]

[1]*Politeknik Negeri Batam, Jurusan Teknik Elektro, Batam*

E-mail: lindawani@polibatam.ac.id

Received: 23-08-2020 *Accepted: 31-12-2020* *Published: 31-12-2020*

Abstrak

Pengujian keamanan perangkat lunak merupakan sarana penting untuk memastikan keamanan perangkat lunak. Tujuan utama pengujian keamanan adalah untuk memeriksa sejauh mana kelemahan mekanisme keamanan yang diimplementasikan. Hal ini dilakukan untuk menemukan kerentanan (*vulnerabilities*) suatu sistem dan memastikan apakah sistem terlindungi. Perangkat lunak yang keamanannya tidak baik akan berakibat hilangnya informasi dan dimanfaatkan oleh pihak lain yang tidak bertangung jawab. Cara yang lebih baik untuk meningkatkan keamanan perangkat lunak adalah dengan memasukkan pengujian keamanan (*security testing*) dalam proses SDLC (*Software Development Life Cycle*). Tulisan ini me*review* pendekatan pengujian keamanan perangkat lunak dan teknik yang diusulkan pada keamanan perangkat lunak beberapa tahun terakhir. Tulisan ini meninjau dan menyimpulkan teknik atau pendekatan yang digunakan pada pengujian keamanan perangkat lunak dalam beberapa penelitian.

Kata kunci: *security testing; SDLC*

Abstract

Software security testing is an essential means to ensure software security. The main purpose of security testing is to check the weaknesses of implemented security mechanisms. Software security is done for searching the vulnerabilities of a system and ensuring the system is protected. Lack of proper software security will impact to the information losing by a threat source. The better way to improve software security is to include security testing in the SDLC (Software Development Life Cycle) process. The review paper is presented and related to software security testing approaches and techniques proposed in software security in recent years. This paper reviews and concludes the techniques and approaches which used in software security testing in several studies.

Keywords: *security testing; SDLC*

Pendahuluan

Permasalahan utama dalam pengembangan perangkat lunak yaitu muculnya celah keamananan (*vulnerability*) dan kesalahan logika *coding* yang berakibat terjadinya eksploitasi sistem. Dalam mengembangkan perangkat lunak, keamanan merupakan hal yang penting untuk dipertimbangkan. *Vulnerabilties* (kerentanan) merupakan cara untuk menimbulkan *threat* (ancaman) yang menyebabkan *risk* merusak sistem. Namun, segala cacat dalam perangkat lunak dapat ditangani pada setiap tahapan *Software Development Life Cycle* (SDLC). SDLC (*Software Development Life Cycle*) berarti sebuah siklus pengembangan perangkat lunak yang terdiri dari beberapa tahapan dan penting keberadaannya terutama dari segi pengembangannya. Permasalahan saat ini, *developer* mendahului pekerjaan mereka tanpa mempertimbangkan cacat keamanan dan kerentanan. Sebagian besar kerentanan dapat ditelusuri akibat analisis yang buruk, desain yang buruk, dan metode pengembangan yang buruk [1]. Oleh karena itu, cara yang lebih baik untuk meningkatkan keamanan melalui pengujian perangkat lunak ke dalam proses SDLC.

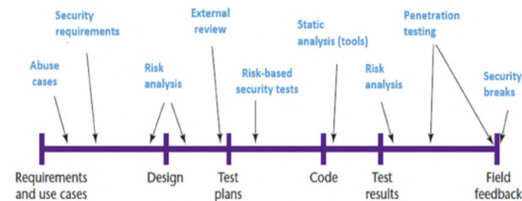
Secure Software Development Life Cycle menekankan suatu keamanan dalam siklus pengembangan perangkat lunak. Perangkat lunak yang aman tidak mudah dihasilkan dimana harus ada perbaikan pada proses pengembangan perangkat lunak untuk meminimalkan jumlah kerentanan dalam mengembangkan perangkat lunak [1]. Proses menganalisis item perangkat lunak dalam mendeteksi perbedaan antara kondisi yang ada dan kondisi yang disyaratkan berupa *defects* (cacat), *bug*, *error,* serta untuk mengevaluasi fitur perangkat lunak yang dikenal dengan nama pengujian (*testing*) [2]. Pengujian adalah salah satu fase dalam SDLC. Saat ini banyak keamanan perangkat lunak yang masih lemah keamanannya. Sehingga *security testing* penting untuk menguji keamanan perangkat lunak. *Security testing* memastikan bahwa perangkat lunak bebas dari segala celah yang dapat menyebabkan kerugian dan permasalahan.

Dalam setiap pengujian keamanan, perangkat lunak akan diperiksa semua faktor keamanan dapat berfungsi dengan baik atau tidak, jika faktor berfungsi dengan baik maka perangkat lunak aman [3]. Tujuan dari *security testing* adalah untuk mengidentifikasi ancaman dalam sistem dan mengukur sejauh mana kerentanan yang ada yang mungkin timbul mengakibatkan hilangnya informasi.Hal ini juga membantu dalam mendeteksi semua risiko keamanan yang mungkin terjadi dalam sistem dan membantu pengembang. Pengujian kemamanan perangkat lunak (*software security testing*) adalah bagian yang tidak terelakkan dari fase SDLC. Oleh karena itu, agar bisa menerapkan pengujian keamanan perangkat lunak dengan benar maka dibutuhkan suatu proses yang sistematis [4].

Pengujian kemamanan perangkat lunak mencakup proses validasi dan verifikasi apakah sistem yang dikembangkan memenuhi persyaratan yang ditentukan oleh pengguna [5]. Pengujian ini nantinya akan menghasilkan perbedaan antara hasil aktual dan yang diharapkan.

A. Tinjauan Software Security Testing-SDLC

Dalam siklus SDLC, keamanan memainkan peran yang sangat penting. Pengujian keamanan perangkat lunak adalah sarana penting untuk mencapai tujuan *secure Software Development Life Cycle* (SDLC). SDLC dianggap sebagai kerangka kerja dalam pengembangan perangkat lunak. Tahapan SDLC dapat dilihat pada gambar 1.



Gambar. 1 *Software Development Life Cycle* [6]

Gambar 1 menunjukkan tahapan dalam siklus SDLC. Ada beberapa tahapan yang harus dilewati dalam pengembangan perangkat lunak. Tahapan tersebut terdiri dari *requirements, design*, *test plan,* implementasi berupa *code, testing* (pengu jian), tahapan terakhir yaitu produk dapat digunakan. Secara umum dekripsi setiap tahapan dirangkum dalam tabel 1 berikut ini:

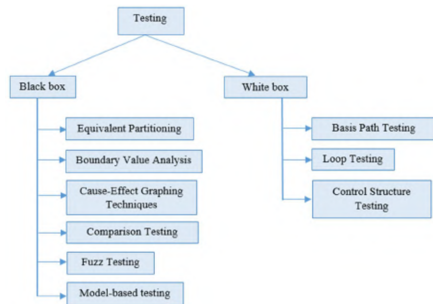
Tabel 1. Deskripsi SDLC

Tahapan SDLC	Deskripsi Proses
Requirement	Seluruh kebutuhan perangkat lunak diperoleh pada fase ini, sehingga didapatkan kegunaan perangkat lunak yang diharapkan oleh pengguna dan batasan yang diinginkan. Proses lainnya yaitu analisis keamanan untuk persyaratan dan kasus penyalahgunaan [7].
Design	Memberi gambaran tentang apa yang harus dilakukan setelah menentukan *requirement*. Kerentanan akan diidentifikasi dan kemudian dianalisis. Selanjutnya akan menganalisis apakah kerentanan tersebut dapat merusak perangkat lunak atau tidak [7].

Tahapan SDLC	Deskripsi Proses
Tahapan SDLC	Deskripsi Proses
Code	Ini adalah tahap dimana pengembang mengimplementasikan kode pada perangkat lunak [7]. Berdasarkan desain yang dibuat pada tahap sebelumnya kode program akan diimplementasikan. Pada tahap ini, kerentanan seperti *buffer overflow* cenderung masuk ke dalam sistem perangkat lunak. Sehingga disarankan agar saat kode selesai, cek secara manual sehingga kesalahan kecil atau *bug* terdeteksi pada tahap ini.
Testing	Pengujian perangkat lunak adalah tahap dimana jumlah kesalahan dan *bug* maksimum harus diidentifikasi. Secara umum *security testing* berfungsi dalam proses validasi, verifikasi dan *error detection*. Tujuannya yaitu: (a) untuk memeriksa apakah perangkat lunak berperilaku sebagaimana mestinya, (b) mengidentifikasi *bug*, (c) mengautentikasi sistem yang diinginkan dengan kebutuhan asli pengguna [7]

1. Pengujian Kemanan Perangkat Lunak

Perangkat lunak dapat diuji dengan pengujian dinamis. Teknik pengujian yaitu *black box testing* dan *white box testing* [8] . Teknik pengujian yang utama ditunjukkan seperti pada gambar 2. Pengujian *white box* secara signifikan efektif karena metode pengujian ini tidak hanya menguji fungsionalitas dari perangkat lunak tapi juga menguji struktur internal perangkat lunak. Pengujian ini menggunakan stuktur kontrol desain posedural untuk memperoleh *test case*. Saat merancang *test case* untuk melakukan pengujian *white box*, keah- lian pemrograman sangat diperlukan.



Gambar 2. Klasifikasi umum teknik pengujian [8]

Pengujian ini dapat diterapkan ke semua tingkatan termasuk pengujian unit, integrasi atau sistem. Jenis pengujian ini juga digunakan sebagai pengujian keamanan untuk menentukan apakah sistem terlindungi [9]. Namun, pengujian *white box* menjadi proses pengujian yang kompleks karena mengharuskan keterampilan pemrograman. Pengujian *black box* adalah pengujian yang dilakukan hanya berdasarkan *requirement* keluaran tanpa mengetahui seputar struktur internal dalam sistem [9]. Pengujian *black box*, dapat dikatakan hanya mengevaluasi tampilan luar secara fungsional tanpa mengetahui apa sesungguhnya yang terjadi dalam proses secara detailnya. Pengujian ini untuk mengetahui apakah fungsi-fungsi, inputan, dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan. Pengujian *black box* merupakan paling sederhana dibandingkan *white box*.

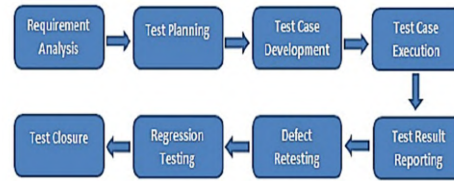
Tabel 2. Deskripsi jenis dari teknik pengujian *white box* dan *black box*

Teknik Pengujian	Jenis	Deskripsi
Pengujian *black box* [8]	*Equivalent partitioning*	Disebut kelas kesetaraan. Teknik ini membagi domain input dari sebuah program ke kelas-kelas sehingga *test case* dapat diperoleh. Pengujian ini merupakan *test case* yang ideal mengungkapkan kelas kesalahan, karena pada teknik ini berusaha mengungkapkan kelas-kelas kesalahan.
	Boundary Value Analysis	BVA merupakan desain teknik *test case* yang melengkapi *equivalent partitioning*. Dari pada memfokuskan hanya pada kondisi input, BVA juga menghasilkan *test case* dari domain *output*. *Boundary value* fokus pada suatu batasan nilai dimana kemungkinan yang terjadi yaitu terdapat cacat tersembunyi.
	Cause-Effect Graphing Techniques	Teknik ini ketika seseorang ingin menerjemahkan sebuah prosedur yang ditentukan ke dalam bahasa perangkat lunak. Setiap kondisi input dialokasikan dalam bentuk grafik hingga terbentuk grafik sebab-akibat. Grafik ini akan dijadikan dalam bentuk tabel keputusan untuk digunakan dalam *test case*.
	Comparison Testing	Para pengembang perangkat lunak menghasilkan versi independen dari sebuah aplikasi. Masing-masing versi diuji dengan data uji yang sama untuk memastikan bahwa seluruhnya menyediakan output yang sama. *Test case* yang dirancang untuk satu versi perangkat lunak dijadikan masukan pada pengujian versi perangkat lunak lainnya, dan diharapkan hasil kedua versi perangkat lunak harus sa-

Teknik Pengujian	Jenis	Deskripsi
		ma.
	Fuzz Testing	Disebut random input, teknik ini memberi masukan acak (inputan yang tidak diharapkan) ke dalam aplikasi. Karakteristik utama pengujian *fuzz testing* adalah: (a) inputnya acak; (b) jika aplikasi *hang*, pengujiannya gagal.
	Model-based testing	Teknik ini menggu nakan pendekatan model berdasarkan *requirement system* dan fungsi yang telah ditentukan. *Test case* diturunkan dari model yang menggambarkan aspek yang diuji. Biasanya disebut sebagai uji abstrak.
	Basis Path Testing	Teknik ini meng evaluasi kompleksi tas kode program dan pendefinisian alur yang akan dieksekusi. *Test case* mendapatkan ukuran yang kompleks dari perancangan prosedural. Selanjutnya digunakan ukuran ini untuk mendefinisikan basis dari jalur pengerjaan.
Pengujian *white box* [8]	*Loop Testing*	Teknik ini melakukan pengujian beberapa kali di bawah kontrol program. Aspek yang paling penting dari pengujian ini adalah memastikan bahwa *loop* kontrol dijalankan berkali-kali dan berhasil keluar saat kondisi tertentu terpenuhi.
	Control Structure Testing	Terdapat dua komponen utama dalam pengujian ini yaitu *condition testing* dan *data flow testing*. *Condition testing,* setiap kondisi logis dalam suatu program diuji. *Data flow testing,* pengujian ini berdasarkan penggunaan variabel yang telah ditentukan.

B. Software Testing Life Cycle

Software Testing Life Cycle mendefinisikan langkah-langkah atau tahap dalam pengujian perangkat lunak. Gambar 3 membahas langkah-langkah STLC yang dilakukan perangkat lunak selama proses pengujian.



Gambar. 3 *Software Testing Life Cycle* [9]

Tahap pertama dalam STLC adalah *requirement analysis*. Dalam langkah ini *Quality Assurance* (QA) memahami persyaratan yang akan diuji. *Test planning* merupakan fase terpenting dalam STLC karena karena ini adalah tahap dimana semua strategi pengujian didefinisikan [9]. Tahap ini berkaitan dengan persiapan rencana uji. *Test case development* merupakan fase dimana tim QA menuliskan *test case*. Yang terpenting pada fase ini adalah *test case* dan *test script*. *Test execution* adalah fase dimana penguji akan menjalankan *test case* yang telah dipersiapkan sebelumnya. Jika ditemukan *bug* atau *error*, penguji akan membuat dalam bentuk laporan [9]. *Test result reporting* merupakan pelaporan hasil yang dihasilkan setelah pengujian dimana laporan *bug* tadi diteruskan ke tim *developer* untuk diperbaiki [9]. *Defect retesting* merupakan tahapan dimana penguji melakukan pengujian ulang terhadap kode yang telah diubah yang diberikan pengembang untuk memeriksa apakah *defect* (cacat) tersebut diperbaiki atau tidak. *Test closure* adalah tahapan akhir dari STLC. Penguji akan menganalisa hasil laporan serta menentukan strategi yang tepat untuk perbaikan aplikasi berikutnya.

C. Software Release Life Cycle

Software Release Life Cycle adalah tahapan-tahapan dalam pengembangan perangkat lunak sampai perangkat lunak tersebut dapat dirilis, *software* yang telah dilakukan perbaikan *bug* atau *error* atau berupa pengembangan *software*. Tahapan ini muncul setelah tahapan STLC dan mencakup tahapan pengujian lebih lanjut yaitu *alpha testing* dan *beta testing*. *Alpha testing* adalah jenis pengujian dilakukan untuk mengidentifikasi semua kemungkinan masalah atau *bug* sebelum produk diluncurkan ke *user* [10]. *Alpha testing* bisa dilakukan menggunakan teknik *white box* atau *black box* [9]. *Alpha testing* dilakukan di lingkungan laboratorium dan biasanya penguji adalah yang memang ahli di bidangnya [11].

Beta testing dilakukan setelah pengujian *alpha*, dan dapat dianggap sebagai *user acceptance testing*. *Beta testing* sepenuhnya berhubungan dengan *end-user* tanpa ada hubungannya dengan *developer* [9]. Ini adalah pengujian terakhir sebelum mengirimkan produk ke

pengguna. *Feedback* langsung dari pelanggan merupakan keuntungan utama *beta testing*. Kedua pengujian ini dilakukan untuk memungkinkan pengguna untuk memvalidasi seluruh kebutuhan. Pengguna menemukan kesalahan yang lebih rinci dan membiasakan untuk memahami perangkat lunak yang telah dibuat.

1. Pengujian Non-fungsional

Pengujian non-fungsional mencakup aspek perangkat lunak yang mungkin tidak berkaitan dengan fungsi tertentu. Hal ini pada dasarnya berkaitan dengan persyaratan non-fungsional dan dimodelkan untuk menilai kesiapan sistem sesuai dengan kriteria yang tidak tercakup dalam pengujian fungsional [2]. Beberapa contoh pengujian non-fungsional dapat dilihat pada gambar 3.

Gambar 4. Contoh teknik pengujian non-fungsional [2]

Gambar 4 menunjukkan beberapa contoh teknik pengujian non-fungsional seperti berikut:

a. Performance testing

Teknik ini menilai seluruh kinerja sistem. Ini digunakan untuk mengevaluasi kinerja sistem di bawah beban kerja tertentu [2].

b. Load testing

Uji beban dilakukan untuk memastikan kapasitas pengambilan beban pada sistem. Pengujian beban dilakukan untuk mengetahui perilaku sistem pada kondisi beban normal maupun beban puncak [2].

c. Endurance testing

Ini adalah teknik pengujian yang dilakukan untuk menentukan perilaku sistem setelah waktu tertentu. Misalnya sebuah sistem bekerja dengan baik pada awal jam pertama namun kinerjanya menurun setelah tiga jam eksekusi [2].

d. Stress testing

Cara dilakukan untuk mengetahui kinerja sistem di luar kapasitas operasionl. Hal ini terkait dengan kapasitas muatan sistem [2].

e. Security testing

Pengujian keamanan dilakukan untuk menilai bahwa sistem aman atau tidak. Ini adalah proses yang berkaitan dengan fakta bahwa data dapat terlindungi dan memelihara fungsionalitas sistem sebagaimana mestinya [2].

f. Recovery testing

Dilakukan untuk memeriksa pemulihan sistem setelah terjadi kegagalan pada perangkat keras. Akibatnya perangkat lunak terpaksa gagal, hingga pada akhirnya yang akan diuji adalah sistem yang telah di*recovery* [2].

g. Compatibility testing

Pengujian kompatibilitas berkaitan dengan pengecekan kompatibilitas sistem dengan lingkungan lainnya. Cara ini memeriksa kompatibilitas sistem terhadap komponen lainnya seperti perangkat keras, perangkat lunak lain, DBMS dan sistem operasi [2].

Review Penelitian

Beberapa penelitian di*review* terkait dengan pengujian keamanan perangkat lunak. Sesuai dengan tahapan SDLC, penulis me*review security testing* dalam dua tahapan yaitu secara teknik maupun metodologi yang digunakan peneliti.

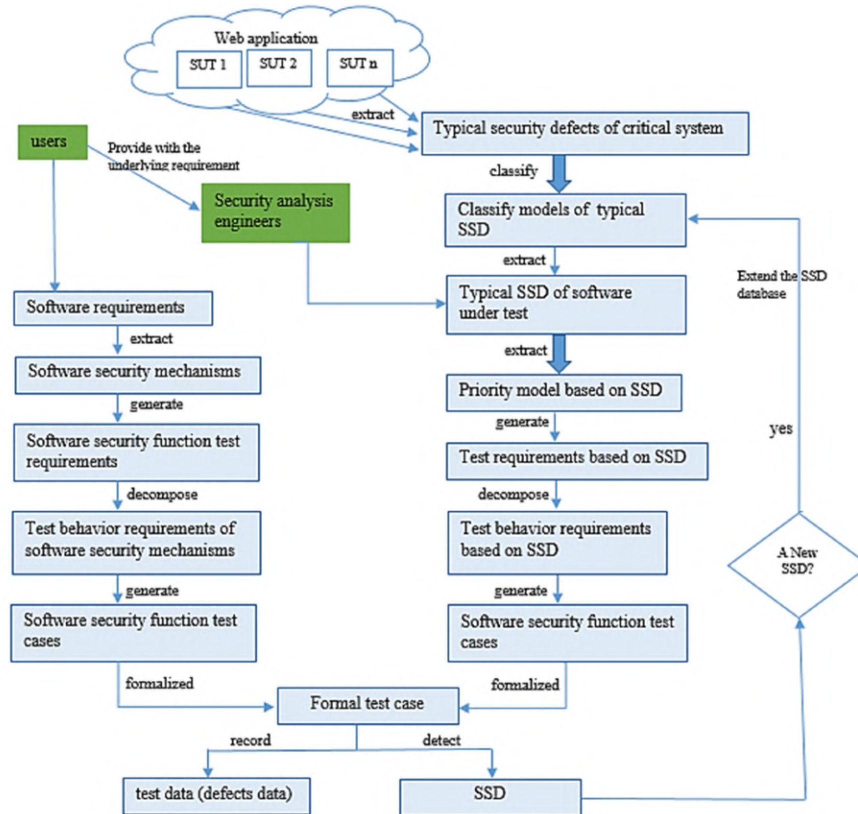
Tabel 3. Review pengujian kemanan perangkat lunak

No.	Tahun	Penulis	Judul
1.	2010	Zhanwei Hui, Song Huang, Bin Hu, dan Yi Yao	*Software Security Testing Based on Typical SSD:A Case Study*
2.	2015	DONG Fangquan, DONG Chaoqun, ZHANG Yao, dan LIN Teng	*Binary-oriented Hybrid Fuzz Testing*
3.	2016	Pan Ping, Zhu Xuan, dan Mao Xinyue	*Research on Ssecurity Test for Application Software Based on SPN*
4.	2016	Aziz Ahmad Rais	*Interface-based Software Testing*

A. *Software Security Testing Based on Typicall SSD : A Case Study*

Zhanwei Hui [12] menghadirkan model *Software Security Testing* (SST) berdasarkan *Soft-*

ware Security Defects (SSD) dan menganalisanya. *Software security defects* (cacat keamanan perangkat lunak) adalah celah keamanan pada perangkat lunak yang dapat mengakibatkan pelanggaran dalam kebijakan keamanan. Kerentanan (*vulnerability)* perangkat lunak adalah representasi dari SSD sebagai fungsi perangkat lunak. Integrasi dimodelkan pada gambar 5.



Gambar. 5 Model integrasi pengujian keamanan perangkat lunak berbasis SSD *[*12]

Model integrasi ini dibagi menjadi dua yang kompatibel dengan SSD. Di bagian kiri adalah proses pengujian fungsi keamanan tradisional, yang menguji mekanisme keamanan perangkat lunak, dan menghasilkan *test case*. Pengujian keamanan perangkat lunak tradisional lebih memperhatikan mekanisme keamanan *Software under Test* (SUT), namun tidak mempertimbangkan SSD secara spesifik yang mungkin mempengaruhi fungsi keamanan SUT [13].

Di bagian kanan adalah proses pengujian berbasis SSD, yang disebut juga *adverse testing*. Pada akhir model, proses integrasi ini akan menghasilkan jenis *test case* yang tidak hanya memvalidasi kebijakan keamanan perangkat lunak, namun juga memberi kepercayaan pada pengguna bahwa *Software under Test (*SUT) dapat melawan bentuk serangan pada *security* [12]. Saat pengujian, penguji menggunakan algoritma simpul pohon untuk SSD. Berikut algoritmanya [13]:

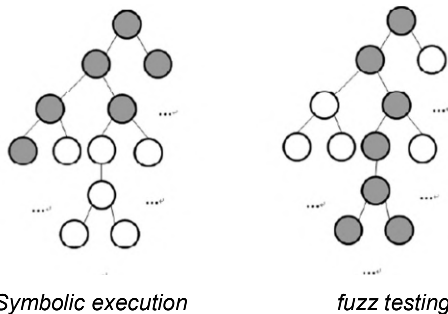
a. Pilih *security defects* sebagai akar permasalahan
b. Security defects adalah objek yang diuji dan merupakan ancaman yang sedang dianalisis.
c. Menganalisis *security defects*
d. Menganalisis secara langsung pengujian *security defects*. Tandai yang cacat sebagai *node* utama, dan tandai keseluruhan sebagai *sub-node*nya. Tandai tipe *node* utama menggunakan logika (*AND / OR*) sebagai hubungan logika dengan *sub-node*.
e. Dekomposisi setiap sub-node

f. Jika bisa terdekomposisi, tandai sebagai node utama dari sub-node tadi, dan lakukan seperti langkah kedua.
g. Ulangi langkah (1) ke langkah (3) sampai semua sub-node tidak bisa didekomposisi atau tidak perlu didekomposisi.

Hasil yang peneliti temukan yaitu adanya indikasi bahwa model pengujian keamanan berbasis SSD yang terintegrasi memang bisa menjadi proses yang efektif. Dengan menjalankan beberapa tes, peneliti berhasil menemukan beberapa *bug* (terkait dengan SSD). Model ini menghasilkan urutan tes secara otomatis [12].

A. Binary-oriented Hybrid Fuzz Testing

Tahun 2015 DONG Fangquan [14] membuat model pengujian *hybrid fuzz.* Model ini menggabungkan pengujian *fuzz* dan *symbolic execution*. Pengujian *fuzz* menemukan *bug* program dengan menjalankan program dengan input acak yang dihasilkan, sehingga ditemukan keadaan *crash*. Meskipun pengujian *fuzz* mampu mengeksplorasi jauh ke dalam program secara efisien, namun tidak dapat menjamin *code coverage ratio* dalam beberapa situasi. Sebaliknya, *symbolic execution* dapat sekaligus mengeksplorasi banyak jalur (*path*) yang bisa diambil oleh sebuah program di bawah masukan yang berbeda [15]. Perbedaan keduanya dapat dilihat pada graph berikut ini:



Gambar 6. Dua metode dalam penerapan *Binary-oriented Hybrid Fuzz Testing* [14]

Gambar 6 menunjukkan *symbolic execution* dapat mengeksplorasi semua jalur program yang dapat dieksekusi secara luas sedangkan pengujian *fuzz* mengeksplorasi keseluruhan jalur program secara mendalam. Peneliti telah menggabungkan kelebihan pengujian *fuzz* dan *symbolic execution* dan merancang pengujian berupa *Binary-oriented Hybrid Fuzz Testing* [14]. Cara kerjanya yaitu membagi dua teknik utama yaitu pengujian *fuzz* dan *symbolic execution*.

Pengujian *fuzz* sebagai *front-end*, menjalankan program dengan input acak. Analisis kode biner dijadikan jembatan antara pengujian *fuzz* dan *symbolic execution*. Jika ditemukan *code coverage ratio* meningkat maka *symbolic execution* sebagai *back-end* mulai bekerja. *Symbolic execution* memecahkan kendala dari *path* dan menghasilkan masukan tes baru dengan *path* baru yang berbeda. Kemudian menjalankan *front end* dan ulangi proses ini. Algortitma pengujian *binary-oriented hybrid fuzz* ditunjukkan pada algoritma 1.

Algoritma 1 *binary-oriented hybrid fuzz testing*

```
1  Input: program P, testing goals Goals
2  while Goals is NOT attained DO
3    coverage_flag=TRUE
4    while coverage_flag==TRUE DO
5      Fuzzing_test() //fuzz testing
6      coverage_monitor() //monitor the code
       coverage ratio //the number of covered
       basic blocks increases
7      if BasicBlock_Num has increased then
8       DONothing
9      else
10       coverage_flag=FALSE
11     endwhile
12     //the code coverage ratio does not in-
       crease
13     Symbolic_Execute()
14     generate New Cases
15     goto 2
16     endwhile
```

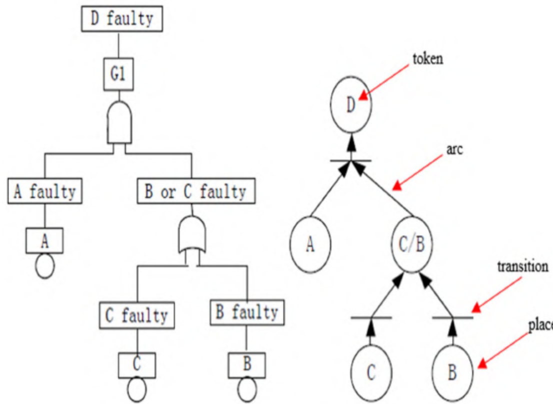
Algoritma 1 menunjukkan *pseudo code* dari *binary-oriented coverage ratio* tidak meningkat pada pengujian *fuzz*, maka ditetapkan *flag FALSE* (jika tidak tidak dilakukan apapun yang ditunjukkan sebagai baris ke 4-10). Kemudian sistem beralih ke *symbolic execution* dan menghasilkan *test case* baru dan mengeksekusi ulang *loop* (baris 13-15). Percobaan ini membuktikan bahwa pengujian untuk program biner dapat mencapai *code coverage ratio* yang lebih tinggi dalam kondisi yang logis.

Ternyata pengujian *fuzz* konvensional tidak dapat menjamin *code coverage ratio* dalam banyak keadaan. Peneliti memperkenalkan metode pengujian baru yaitu pengujian *hybrid fuzz*, untuk menguji program biner. Percobaan ini membuktikan bahwa pengujian *hybrid fuzz* untuk program biner dapat mencapai *code coverage ratio* yang lebih tinggi dalam keadaaan yang logis [14].

B. Research on security test for application software based on SPN

Tahun 2016 Pan Ping [16] menganalisis keamanan perangkat lunak aplikasi dengan parameter SPN (*Stochastic Petri nets*). SPN tidak hanya dapat menggambarkan arsitektur perangkat lu-

nak dengan baik, namun juga menyajikan proses kerja perangkat lunak. Secara sederhana digambarkan dalam sebuah diagram pohon dan model SPN ditunjukkan pada gambar 7.



Gambar 7. Model sederhana diagram pohon dan SPN [16]

Pemetaan SPN didasari oleh proses Markov. *Places* merepresentasikan keadaan dalam sebuah koponen atau sistem dan dapat berhubungan di dalam satu fase. *Transition* digunakan untuk menggambarkan peristiwa yang terjadi dalam sistem. Biasanya akan menghasilkan modulasi ke status sistem tersebut. *Token* adalah penanda atau identitas yang berada di *places*. Kehadiran token di suatu tempat menunjukkan bahwa kondisi telah sesuai dengan fungsi yang diberikan [17].

Tujuan penggunaan SPN dalam analisis keamanan perangkat lunak adalah untuk mengetahui keadaan yang menyebabkan perangkat lunak menjadi *failed*. Berikut adalah langkah-langkah SPN untuk pengujian dan analisa perangkat lunak [16]:

Pertama kita perlu menganalisis risiko perangkat lunak untuk mengkonfirmasi permintaan keamanan dari *software development*. Kemudian menentukan keadaan perangkat lunak yang tidak aman. Pada tahap ini, perancang perangkat lunak perlu memiliki pemahaman tentang proses perancangan perangkat lunak. Selain itu menentukan keadaan berbahaya yang tidak sesuai dengan tujuan perancangan perangkat lunak serta menetukan persyaratan keamanan dan spesifikasi desain [16].

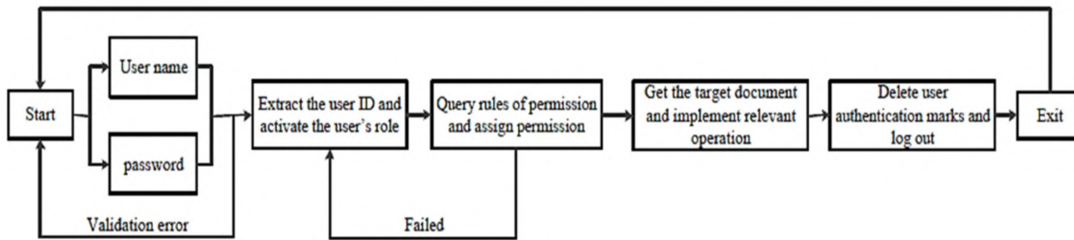
Selanjutnya menetapkan model SPN dari perangkat lunak, sehingga dapat dibangun grafik SPN. Buatlah daftar kumpulan tanda yang dapat dicapai. Lakukan analisis terhadap tanda yang dapat dicapai dengan mudah agar diperoleh mengapa sistem dikatakan gagal. Sesuai dengan analisis kegagalan perangkat lunak tadi , maka penguji dapat kembali ke tahap perancangan perangkat lunak *(software design stage*). Berikutnya akan diterapkan analisis keandalan dan keamanan perangkat ke *software design stage* dan *software testing*, serta desain *test case* [16].

Pengujian keamanan pada perangkat lunak pada aplikasi biasanya mencakup *user authentication* atau pengujian otentikasi pengguna,pengujian sistem jaringan dan pengujian *database*. Peneliti [16] juga menggambarkan operasi logika sederhana keamanan *user authentication*. Kontrol akses pengguna yang sederhana dalam perangkat lunak dijadikan sebagai gambaran metode pengujian keamanan perangkat lunak aplikasi berdasarkan SPN.

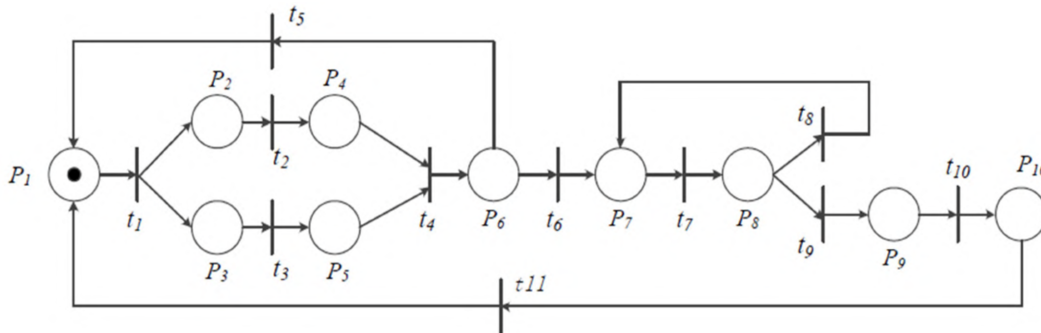
Gambar 8 menunjukkan diagram alir operasi logika *user security authentication*. Dari contoh operasi logika ini lah yang nantinya akan dikembangkan model SPN yaitu digambarkan pada *petri nets.* Gambar 8 merupakan diagram alur operasi logika sederhana dari *user security authentication*.

Proses spesifiknya adalah sebagai berikut: Input berupa *user name* dan *password* untuk *login authentication* saat menggunakan aplikasi. Jika autentikasi berhasil, sistem akan mengekstrak *user* ID dan mengaktifkan *user* dan kemudian memberikan izin kepada *user*. Setelah mendapatkan izin untuk masuk dalam aplikasi, pengguna dapat melakukan operasi yang diinginkan. Sistem akan menghapus *user authentication* saat *log out*, dan akhirnya kembali ke keadaan awal. Jika digambarkan dalam model SPN maka akan dimodelkan seperti gambar 9.

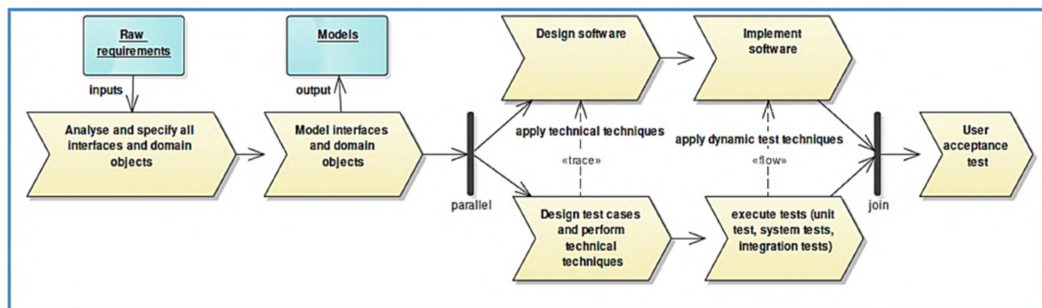
Gambar 9 menunjukkan model SPN dari operasi logika *user security authentication*. *State* mewakili setiap keadaan parsial sistem perangkat lunak. *Transition* akan di*triger* untuk sistem perangkat lunak dari satu *state* ke *state* lain. Pada saat proses *running* keseluruhan sistem perangkat lunak, *token* atau *petri nets* tersebut terus-menerus akan melakukan transfer. Transfer dikatakan sebagai fungsi perangkat lunak akan melakukan berbagai operasi saat proses dijalankan. P_1-P_{10} merepresentasikan keadaan dari operasi yang diinginkan. *Transition* t_2-t_4 digunakan untuk memverifikasi nama pengguna dan *password login*. *Transition* t_5 -t_{11} adalah operasi untuk menetapkan peran dan kontrol akses izin kepada pengguna [16].



Gambar 8. Diagram alir operasi logika *user security authentication* [16]



Gambar. 9 Model SPN dari operasi logika *user security authentication* [16]



Gambar 10. Proses pengujian berdasarkan *interface-base software* [18]

C. Interface-based software testing

Tahun 2016 Aziz Ahmad Rais [18] menyediakan perangkat lunak berbasis *interface* dengan teknik pengujian yang lebih baik dalam mengukur kualitas perangkat lunak. Pengujian ini bersifat otomatis untuk menghasilkan kualitas perangkat lunak, melalui pengujian di awal, serta meningkatkan keseluruhan kemampuan uji [18]. Konsep dari pengujian berbasis *interface* ini dimodelkan seperti pada gambar 10.

Konsep dibalik pengujian berbasis *interface* bukan berarti menunda desain dan eksekusi namun menunggu sampai semua kelas yang akan diimplementasikan sudah siap. Seperti yang terlihat pada gambar 10 bahwa *raw requirement* (sebagai input) akan dianalisis dan dimodelkan dalam *interface* yang sudah ditentukan. Sebagai contoh *client registration requirement*, persyaratan pertama yang ditentukan bahwa perangkat lunak mengharuskan klien mendaftar sebelum mereka meminta layanan lain. Dengan demikian, layanan registrasi klien dapat ditentukan oleh satu *interface*, sementara klien mengakses layanan lain melalui *interface* GUI. Setelah pengumpulan data dan manipulasi selesai, layanan pendaftaran klien harus menyimpan data klien dalam sebuah repositori [18].

Business requirements interface inilah yang menentukan layanan registrasi klien. Menyimpan

data, mengakses layanan registrasi klien melalui GUI, dan menerapkan berbagai jenis validasi dilakukan oleh *technical design interface*. Setelah memisahkan setiap fungsi, akan mempermudah dalam merancang *interface* untuk layanan yang disediakan oleh perangkat lunak. Setelah menentukan *interface* layanan perangkat lunak, penguji dapat menulis *test case* misalnya uji unit, atau uji otomatis. Pengembang *software* dapat mengidentifikasi semua *technical interface*, merancang perangkat lunak, dan kemudian menerapkannya. *Interface* dapat dengan mudah dimodelkan menggunakan UML (*Unified Modeling Language*) [18].

Untuk merancang tes dan menerapkan proses pengujian dalam proyek pengembangan perangkat lunak, dibutuhkan sebuah strategi efektif untuk melaksanakan *test case*. Salah satu cara yaitu menerapkan pengujian berbasis *interface*. Dapat disimpulkan bahwa kualitas perangkat lunak menunjukkan bahwa perangkat lunak berkualitas tinggi bergantung pada pengujian yang efektif dan berhasil. Sementara pengujian yang sukses bergantung pada teknik pengujian yang tepat. Teknik ini mendukung pengujian yang otomatis, dan pengembangan teknik pengujian secara paralel [18].

Simpulan

Dalam tulisan ini, tinjauan berbagai pendekatan dan teknik pengujian keamanan telah ditinjau dan temuan ditabulasikan dalam urutan kronologis. Selain itu tulisan ini juga mendefinisikan dan mengklasifikasikan teknik pengujian keamanan perangkat lunak yang biasa digunakan pada umumnya. Sebagian besar teknik pengujian keamanan diimplementasikan pada berbagai tahap SDLC (*Software Development Life Cycle)*. Hal ini dikarenakan pengujian keamanan perangkat lunak berperan penting untuk menghindari serangan atau ancaman dari luar. Beberapa peneliti dalam penelitianya juga mencoba menerapkan model dan teknik pengujian terbaru seperti: pengujian berdasarkan SSD (*Software Security Defects),* pengujian *hybrid fuzz,* pengujian dengan parameter SPN (*Stochastic Petri nets*), serta pengujian berbasis *interface*. Dari keseluruhan pengujian ternyata berdampak baik dalam proses pengujian.

Daftar Pustaka

[1] Y. H. Tung, S. C. Lo, J. F. Shih, and H. F. Lin, “An integrated security testing framework for secure software development life cycle,” in *The 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016.

[2] A. Sethi, “A review paper on levels, types & techniques in software testing,” in *International Journal of Advanced Research in Computer Science*, 2017, vol. 8, no. 7.

[3] R. Kumar, S. A. Khan, and R. A. Khan, “Software security testing a pertinent framework,” in *Journal of Global Research in Computer Science (JGRCS)*, vol. 5, no. 3, pp. 23-27, March. 2014.

[4] N. Mahendra and S. A. Khan, “A categorized review on software security testing,“ in *International Journal of Computer Applications*, vol. 154, no. 1, Nov. 2016.

[5] S. Krishnaveni, D. Prabakaran, and S. Sivamohan, “Analysis of software security testing techniques in cloud computing,” in *International Journal of Modern Trends in Engineering and Research*, vol. 02, Issue. 01, Jan. 2015.

[6] G. McGraw, “Software security testing,” *IEEE Security & Privacy*, Sep.2004.

[7] M. Khari, Vaishali, and P. Kumar, “ Embedding security in Software Development Life Cycle (SDLC),” in *International Conference on Computing for Sustainable Global Development*, 2016, pp. 3805-4421.

[8] J. Irena, “software testing methods and techniques,” *IEEE Computer Society*, 2008, pp. 30-35.

[9] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, “Software testing techniques: a literature review,“ in *6th International Conference on Information and Communication Technology for The Muslim World,* 2016.

[10] N. Jenkins, *A Software Testing Primer.* San Francisco, California: Creative Commons, 2008.

[11] Guru99, “Alpha testing Vs Beta testing,” 2017. [Online]. Available: https://www.guru99.com/alpha-beta-testing-demystified.html. [Accessed: 07 Dec 2019].

[12] Z. Hui, S. Huang, B. Hu and Y. Yao, "Software security testing based on typical SSD: A case study,"in *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, Chengdu, 2010, pp. V2-312-V2-316.

[13] S. Huang, Z. Hui, L. Wang, and X. M. Liu, "A Case Study of Software Security Test Based On Defects Threat Tree Modeling," in *International Conference on Multimedia Information Networking and Security,* 2010.

[14] D. Fangquan, D. Chaoqun, Z. Yao, and L. Teng, " Binary-oriented hybrid fuzz testing," in *IEEE,* 2015, pp. 4799-8355.

[15] R. Baldoni, E. Coppa, D. Cono D'Elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," in *Cyber Grand Challenge highlights from DEF CON 24*, Aug. 2016.

[16] P. Ping, Z. Xuan, and M. Xinyue, "Research on security test for application software based on SPN," in *13th Global Congress on Manufacturing and Management, GCMM*, 2016, pp. 1140 – 1147.

[17] School of Informatic University of Eidenburgh Scotland, "Stochastic Petri Nets," 2017. [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/pm/Note7. [Accessed: 10 Dec 2019].

[18] A. A. Rais, "Interface-based software testing," in *Journal Of Systems Integration*, 2016.