Patch Management dan Analisis Keamanan Software Patch

Tugas Akhir Mata Kuliah Keamanan Perangkat Lunak
EL5215

Oleh
AULIAK AMRI
NIM: 23215077
Program Studi Magister Teknik Elektro

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2016

Daftar Isi

Pendahuluan

Patch merupakan bagian software yang dirancang untuk memperbaiki atau memperbaharui software, seperti memperbaiki kelemahan keamanan, penambahan fitur, memperbaharui fungsi yang sudah ada, dan mengubah behavior dari software tersebut [1]. Sistem yang tidak dilakukan patch, dan terhubung ke jaringan, rentan untuk dieksploitasi selama penggunaannya bahkan sejak proses instalasi. Dalam keadaan yang rentan terhadap serangan, sistem tersebut tidak seharusnya terbuka [2].

Kerentanan keamanan terus ditemukan pada software sehingga menjadi sangat penting untuk menjaga agar tetap up to date, yaitu dengan melakukan patch sesuai dengan kerentanan keamananya. Saat ini banyak sistem memiliki perangkat yang dapat mendownload dan menginstal update keamanan secara otomatis. Perangkat tersebut berguna untuk memperkecil jarak antara penemuan kelemahan sistem dan ketersediaan patch [2].

Namun demikian, mekanisme update otomatis merupakan hal yang problematik dalam organisasi. Administrator IT perlu melakukan pre-assessement terhadap patch yang baru dirilis. Selain itu, dampak dari pemasangan patch juga perlu dikaji misalnya failure yang terjadi setelahnya. Beberapa perangkat dapat digunakan untuk scheduling dan deployment patch, namun tetap memerlukan campur tangan manusia selama proses patch. Oleh karena itu, perlu dilakukan patch management. Tiga tahapan utama dalam proses patch management yaitu i) notifikasi, ii) scheduling, dan iii) deployment dan post deployment [1]. Beberapa metode patch yaitu Shadow Patching [3], TestTalk [4], pendekatan teori game [5], dan patch dinamis pada embedded software [6].

Implementasi dari patch management pada perusahaan dilaporkan memakan biaya yang cukup besar, lebih dari $2 milyar pada tahun 2002 [7]. Permasalahan lainnya adalah banyak mekanisme software update yang tidak aman. Analisis terhadap beberapa software update menunjukkan bahwa proses update hanya mengandalkan jaringan internet, bukan menggunakan secure content distribution. Selain itu, kurangnya penggunaan mekanisme keamanan dasar seperti verifikasi digital signature pada proses update. Hal ini menyebabkan proses tersebut rentan terhadap serangan man-in-the-middle (MITM). Channel update yang terbuka dan unprotected merupakan backdoor yang dapat digunakan untuk menyebarkan malicious code [8]. Patch itu sendiri bahkan dapat digunakan oleh attacker untuk melakukan serangan [9]. Pendekatan untuk mengidentifikasi patch yang tepat dilakukan dengan

menganalisis perbedaan antara dua file yang dapat dieksekusi [10]. Melalui informasi yang diperoleh dari perbedaan patch, kelemahan keamanan atau bugs suatu software dapat diketahui [11]. Makalah ini membahas tentang patch management, metode patch, dan analisis terhadap keamanan proses patch.

Software Patch

Patch merupakan bagian software yang dirancang untuk memperbaiki atau memperbaharui software, seperti memperbaiki kelemahan keamanan, penambahan fitur, memperbaharui fungsi yang sudah ada, dan mengubah behavior dari software tersebut [1].

Suatu sistem rentan terhadap serangan keamanan selama proses instalasi, sebelum dilakukannya patch dan proses hardening sistem. Selain itu juga, selama penggunaannya, sistem yang tidak dilakukan patch dan terhubung ke jaringan rentan terhadap eksploitasi keamanan oleh attacker. Pondasi keamanan yang baik memerlukan proses instalasi, patch, dan konfigurasi sistem yang tepat [2].

Dalam kondisi yang rentan, sebuah sistem tidak seharusnya terbuka melainkan tetap dalam jaringan terlindungi selama pengembangan sistem. Jaringan tersebut terisolasi dari pihak luar dimana sistem operasi dan semua patch diinstal dengan removable media seperti DVD. Cara lainnya adalah dengan membuat jaringan dengan akses terbatas dimana tidak ada akses inbound dan memiliki akses outbound hanya pada situs yang diperlukan untuk proses instalasi dan patch sistem [2].

Oleh karena terus ditemukannya kelemahan keamanan, sangat penting sekali software dijaga agar terus up to date dengan cara menginstal patch terkait kelemahan keamanan tersebut. Untuk melakukan patch, software telah dilengkapi dengan fitur untuk mendownload dan menginstal patch. Fitur ini perlu dikonfigurasi dan digunakan untuk memperkecil waktu antara ditemukannya kelemahan keamanan dan ketersediaan patch [2].

Patch terkadang dapat menyebabkan ketidakstabilan sistem. Oleh karena itu, untuk sistem dimana ketersediaan dan waktu operasionalnya sangat penting, update otomatis sebaiknya tidak digunakan. Semua patch perlu diuji dan divalidasi sebelum patch benar-benar diinstal pada sistem [2].

Mekanisme update otomatis adalah sesuatu yang problematik dalam lingkungan perusahaan. Administrator IT harus melakukan penilaian terhadap dampak dari dilakukannya patch terhadap sistem yang ada. Penilaian tersebut tidak mungkin dilakukan jika fitur update otomatis digunakan. Selain itu, penilaian juga perlu dilakukan untuk mengetahui kemungkinan adanya kegagalan pada aplikasi lain yang sedang berjalan. Semua ini perlu didokumentasikan untuk tujuan audit dan recovery dari kegagalan sistem [1].

Patch Management

Meskipun bertujuan untuk memperbaiki program, patch terkadang mendatangkan masalah baru misalnya terganggunya fungsi lainnya. Patch management adalah proses dalam menggunakan rencana dan strategi untuk memilih patch apa yang akan digunakan terhadap sistem apa pada waktu tertentu.

Selain itu, manajemen patch dalam perusahaan memiliki berbagai macam tantangan [1] seperti di bawah ini.

1) Kurangnya standardisasi pada software, hardware, maupun services.
2) Pelanggan menginginkan kebijakan manajemen patch yang berbeda untuk memenuhi kebutuhannya.
3) Jumlah tenaga kerja dan biaya yang tidak efektif.

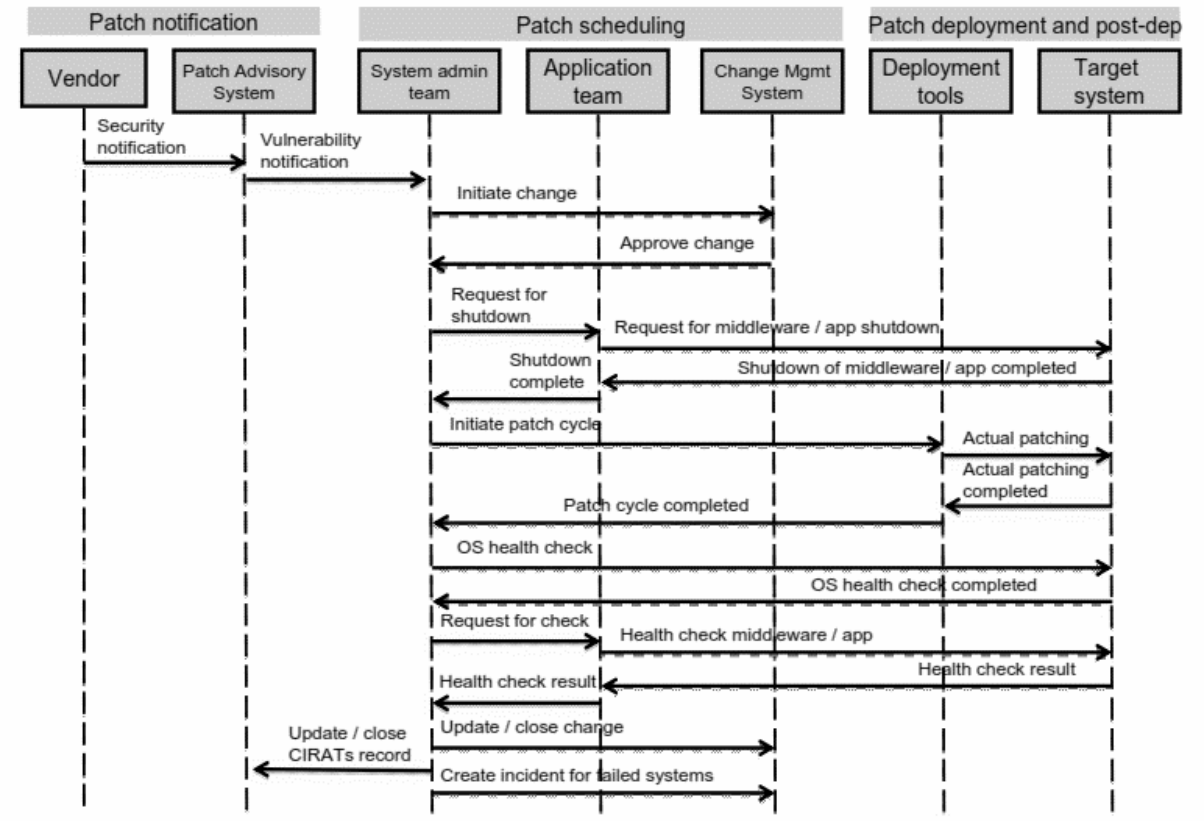
Patch Management Tool

Ada dua tipe patch management tool yaitu tool dari ISV vendor dan tool dari pihak ketiga. ISV vendor memiliki mekanisme sendiri untuk melakukan update software. Contoh dari ISV vendor yaitu Windows Server Update Service (WSUS) dan Red Hat Network (RHN), sedangkan tool dari pihak ketiga misalnya adalah Tivoli Endpoint Manager (BigFix), VMware vCenter Update Manager. Tool ini menyediakan fitur penjadwalan dan pemasangan patch yang relatif mudah digunakan. Namun demikian, tetap memerlukan campur tangan manusia selama proses manajemen patch [1].

Patch Management Process Workflow

Manajemen patch pada lingkungan perusahaan tidak dilakukan hanya dengan penggunaan tool untuk pemasangan patch. Hal ini karena setiap perubahan dapat menyebabkan kegagalan sistem. Pada lingkungan perusahaan, permasalahan ini dapat mempengaruhi bisnis jika tidak

dikendalikan secara hati-hati. Oleh karena itu, tindakan pengamanan perlu dilakukan pada proses patch untuk memperkecil risiko.



Gambar 1. Alur proses manajemen patch [1]

Gambar diatas merupakan alur dari proses manajemen patch yang terdiri dari i) notification, ii) scheduling, dan iii) deployment and post-deployment [1].

Notification

Pada tahap notifikasi, saat patch dikeluarkan oleh vendor, sistem mendeteksi ketersedian patch tersebut. Ketersediaan patch dapat diketahui secara otomatis dari sistem notifikasi vendor maupun dicek secara manual pada periode waktu tertentu. Patch Advisory System bertanggung jawab untuk memberitahukan System Admin Team yang asetnya akan terkena dampak sehingga bisa dilakukan penjadwalan, pemasangan, dan pengujian. Patch Advisory System juga digunakan untuk melacak siklus hidup patch, misalnya, apakah patch telah berhasil atau gagal, pada bagian mana patch dipasang, siapa yang telah melakukan pemasangan, pada waktu

kapan pemasangan dilakukan, dan jika patch tidak dipasang tepat pada waktunya, apakah pengguna diberitahu, dan lain sebagainya.

Scheduling

Pada tahapan ini, semua pihak, seperti administrator sistem, support team, application owner, dan customer, diinformasikan mengenai patch baru. Setiap tim melakukan negosiasi untuk menentukan sistem apa yang akan dilakukan patch, kapan patch akan dipasang, apakah ada pengecualian yang perlu dibuat, dan lain sebagainya. Negosiasi yang dilakukan oleh berbagai pihak membutuhkan waktu yang tidak sebentar. Terlebih lagi setiap tim bisa saja berada pada lokasi geografis dan zona waktu yang berbeda. Oleh karena itu, proses ini bisa memakan waktu yang lama untuk mencapai keputusan yang disepakati.

Deployment and post-deployment

Setiap tindakan yang telah disepakati pada tahap sebelumnya akan dieksekusi pada tahap ini. Eksekusi dilakukan dengan saling koordinasi antar tim yang melaksanakannya. Ketika patch sistem operasi akan dilakukan, tim aplikasi terlebih dahulu akan mematikan aplikasi dan middleware yang sedang berjalan untuk mencegah kerusakan data dan masalah ketidakkonsistenan lainnya. Saat patch telah selesai dilakukan, berbagai kegiatan pengecekan dan uji regresi akan dilakukan untuk memverifikasi bahwa perubahan yang dilakukan tidak memberikan dampak buruk pada sistem. Pada proses ini, penggunaan tool pemasangan patch hanya dilakukan pada tahap tertentu dari banyak tahap yang dilakukan.

Metode Patch Lainnya

Beberapa metode patch yaitu Shadow Patching, TestTalk, pendekatan teori game, dan patch dinamis pada embedded software dijelaskan pada bagian berikut ini.

Shadow Patching

Saat dilakukan pemasangan patch, operasional software terkadang perlu dihentikan sementara yang mengakibatkan terganggunya layanan software tersebut. Penghentian layanan ini sering disebut dengan maintenance window. Shadow patching [3] adalah teknik manajemen patch dengan memanfaatkan kemampuan virtualisasi. Metode ini memungkinkan administrator sistem untuk melakukan proses patch diluar maintenance window, seperti melakukan download patch, pemasangan patch, dan pengujian post-installation patch.

Pada framework Shadow Patching, proses patch, pengujian, dan troubleshooting dilakukan pada Virtual Machine (VM) yang merupakan kloning dari VM yang asli sehingga proses patch tidak mempengaruhi sistem yang sebenarnya. Perubahan file system pada VM kloning direkam dan selanjutnya disatukan dengan VM asli. Down time pada VM asli hanya terjadi saat operasi penggabungan ini, sehingga menjadi lebih cepat dan reliable dibandingkan dengan cara tradisional. Selain itu, tahapan pengujian regresi dan troubleshooting disembunyikan yang mengakibatkan maintenance window secara signifikan dapat dipersingkat.

TestTalk

Sebuah sistem akan tetap rentan terhadap serangan setelah diketahui memiliki kelemahan keamanan dan belum dilakukan update. Attacker dapat masuk ke dalam sistem dengan mengeksploitasi kelemahan keamanan tersebut. Pemasangan patch secara manual dapat memakan waktu dan rawan terjadi kesalahan.

Securibot merupakan framework untuk melakukan pengecekan keamanan dan patching secara otomatis. Framework ini melakukan pengujian keamanan menggunakan security profile dan security update. Selain itu, framework ini juga dapat mendeteksi sistem yang terkena serangan menggunakan attack signature. Securibot memungkinkan vendor sistem untuk merilis patch dalam format yang bisa dibaca mesin sehingga secara otomatis dapat mengecek update keamanan dan menginstal patch.

Security update pada framework terdiri dari TestTalk security scanner dan TestTalk security patcher. Scanner menentukan apakah terdapat kelemahan keamanan atau tidak. Jika terdapat kelemahan keamanan, patcher akan memperbaikinya [4].

Pendekatan teori game

Pada sistem komersial banyaknya kelemahan keamanan sangat besar sehingga hanya sebagian yang dapat diperbaiki karena sumber daya yang terbatas. Oleh karena itu, memilih kelemahan keamanan tersebut menjadi tantangan bagi manajer keamanan. Bagi attacker, memilih kelemahan keamanan yang akan diserang juga merupakan tantangan baginya. Strategi kedua pihak dapat dijelaskan dengan menggunakan framework Game Theory (GT). Skenario pertahanan atau penyerangan dapat dilakukan mapping ke dalam varian dari model GT yaitu Search Games. GT dapat memprediksi tindakan mereka melalui distribusi probabilitas dari pilihan yang mungkin. Hal ini dapat membantu secara semi otomatis dalam memilih manajemen patch dengan sumber daya yang terbatas [5].

Dynamic Patch pada Embedded Software

Metode dynamic patch dilakukan untuk melakukan patch aplikasi embedded multitasking real-time system selama aplikasi tersebut berjalan. Metode tersebut menggunakan teknik modifikasi binary dan mengotomatisasi keseluruhan proses patch. Melalui metode ini dimungkinkan untuk memasukkan dan menghapus kode program tanpa menggunakan kode sumber yang asli. Selama aplikasi berjalan, kode patch dapat langsung diaktifkan [6].

Keamanan Patch

Pengguna komputer melakukan update software dengan berbagai macam cara, beberapa secara manual, otomatis, bahkan ada yang tidak melakukannya. Selain itu, proses manajemen patch pada perusahaan dilaporkan dapat menghabiskan dana lebih dari $2 milyar pada tahun 2002 [7]. Ditambah lagi, semakin berkembangnya teknologi perangkat mobile, sensor dan RFID membawa tantangan tersendiri dalam melakukan update embedded software secara aman.

Client dapat menggunakan content distribution network untuk mendownload software update secara aman. Analisis dilakukan terhadap beberapa mekanisme update pada software yang populer. Hasil analisis menunjukkan bahwa update dilakukan hanya mengandalkan jaringan internet, tidak menggunakan secure content distribution. Mekanisme tersebut memiliki kelemahan terhadap serangan man-in-the-middle [8].

Penerapan teknik keamanan pada mekanisme update software seperti verifikasi digital signature tidak dilakukan. Mekanisme update tersebut dibiarkan terbuka dan tidak terlindungi sehingga channel update tersebut dapat menjadi backdoor untuk menyebarkan kode program berbahaya [8].

Survei terhadap Software Update System

Analisis telah dilakukan untuk mengetahui ketahanan beberapa software update system terhadap man-in-the-middle attack (MITM). Hasil analisis menunjukkan bahwa beberapa sistem memiliki kelemahan terhadap serangan MITM [8].

Pada Apple MacOS Software Update misalnya, setiap binary update ditandatangani untuk menjamin integritas dan autentikasi software. Setiap update memiliki file dengan nama “signature” yang berisi 1024 byte kode hash dari file instalasi. Setiap file instalasi dilakukan pengecekan terhadap signature yang hanya dapat ditandatangani oleh private key Apple Computer Corp dimana public key tersedia pada media instalasi sistem operasinya. Pada

MacOS server, administrator sistem dapat melakukan mirror update pada jaringan lokal agar client dapat menginstal update sehingga attacker tidak dapat menyisipkan malicious update.

Software	Platform	Authenticated Connection?	Authenticated Binaries?
Apple Software Update	MacOS	no	yes
Windows Update	Windows	partially	yes
Adobe Acrobat	MacOS	no	yes
Microsoft Office	MacOS	no	yes
Mozilla Firefox	Windows	partially	no
Fugu	MacOS	no	no
McAfee VirusScan	Windows	no	no
McAfee VirusScan Enterprise	Windows	unknown	yes
McAfee Virex	MacOS	no	no*
Debian	Linux	no	yes

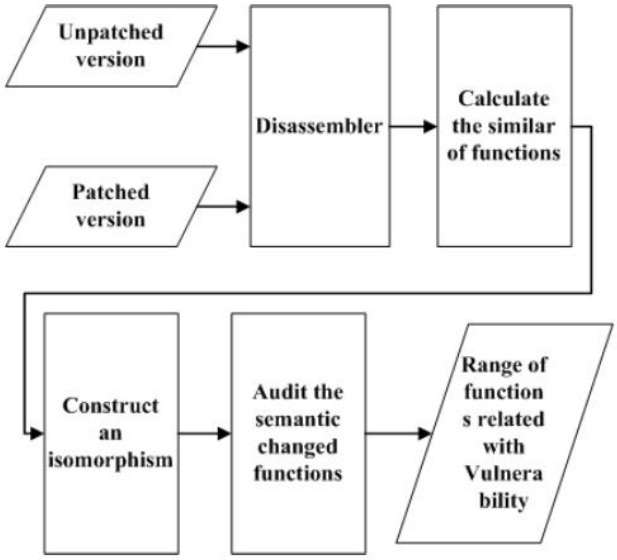
Tabel 1. Hasil analisis terhadap software update system [8]

Tabel diatas menunjukkan bahwa mekanisme software update yang dilakukan ada yang menggunakan koneksi terautentikasi maupun binary update terautentikasi, bahkan ada yang tidak kedua-duanya. Pengujian dilakukan dengan menganalisis network trace dan melakukan serangan MITM pada setiap software package. Serangan MITM dilakukan untuk mengetahui apakah koneksi dan file update dilakukan autentikasi atau tidak.

Kelemahan keamanan pada binary patch

Tujuan dari dilakukannya patch adalah untuk memperbaiki kelemahan software. Namun demikian, patch juga dapat digunakan sebagai alat untuk melakukan serangan oleh attacker [9]. Selain itu, banyak pengguna yang tidak melakukan update tepat waktu membuat komputer mereka rentan terhadap serangan “delayed patch attack”.

Kelemahan keamanan pada patch dapat diketahui dengan menganalisis dua versi executable file [10]. Pendekatan yang dilakukan adalah dengan menemukan dan mengaudit perbedaan antara keduanya. Hal ini dilakukan dengan merepresentasikan executable file ke dalam fungsi callgraph [12]. Isomorfisme dari dua callgraph dibuat untuk melihat perubahan semantiknya. Perubahan semantik kemudian diaudit untuk mengetahui apakah perubahan tersebut merupakan kelemahan keamanan atau bukan.



Gambar 2. Arsitektur sistem untuk analisis dua file program [10]

Gambar diatas adalah proses yang dilakukan untuk menganalisis perbedaan antara dua executable file untuk menemukan kelemahan keamanan pada patch. Proses tersebut dilakukan dengan dua tahap, tahap pertama membangun isomorfisme dari dua callgraph dan tahap kedua mengaudit perubahan semantik untuk mengetahui apakah perubahan tersebut terkait dengan kelemahan keamanan.

Langkah-langkah yang dilakukan adalah sebagai berikut.

1) Binary file dibongkar menjadi rangkaian perintah x86 oleh IDA Pro
2) Menghitung kesamaan fungsi pada rangkaian perintah
3) Isomorfisme dua versi file dibuat dengan menghitung kesamaan antar graph
4) Perubahan semantik fungsi diaudit untuk mengetahui apakah terkait dengan kelemahan keamanan.

Menemukan perbedaan semantik

Berikut ini akan dijelaskan metode untuk menghitung kesamaan antar fungsi. Isomorfisme kemudian dibuat berdasarkan penghitungan kesamaan antar graph.

Menghitung kesamaan antar fungsi

Fungsi direpresentasikan sebagai Control Flow Graphs (CFG) yang tersusun atas blok-blok dasar. Tiap blok dasar berisi rangkaian perintah x86 dan hanya mempunya satu entry point dan satu exit point. Kesamaan antar fungsi dihitung dengan membentuk isomorfisme dari CFG.

CFG dikonversikan ke dalam rangkaian blok dasar, yang tersusun atas address awal dari tiap blok dasar. Longest Common Subsequence (LCS) dari blok dasar dibuat. LCS diperlukan untuk menentukan kesamaan antar blok dasar yang tersusun atas rangkaian perintah x86. Misalnya terdapat fungsi A dan B, kesamaan antar kedua fungsi tersebut dihitung dengan rumus berikut, dimana fungsi $length(x)$ menyatakan banyaknya perintah di dalam x.

$$Similarity(A,B) = \frac{2 \times length(LCS(A,B))}{length(A) + length(B)}$$

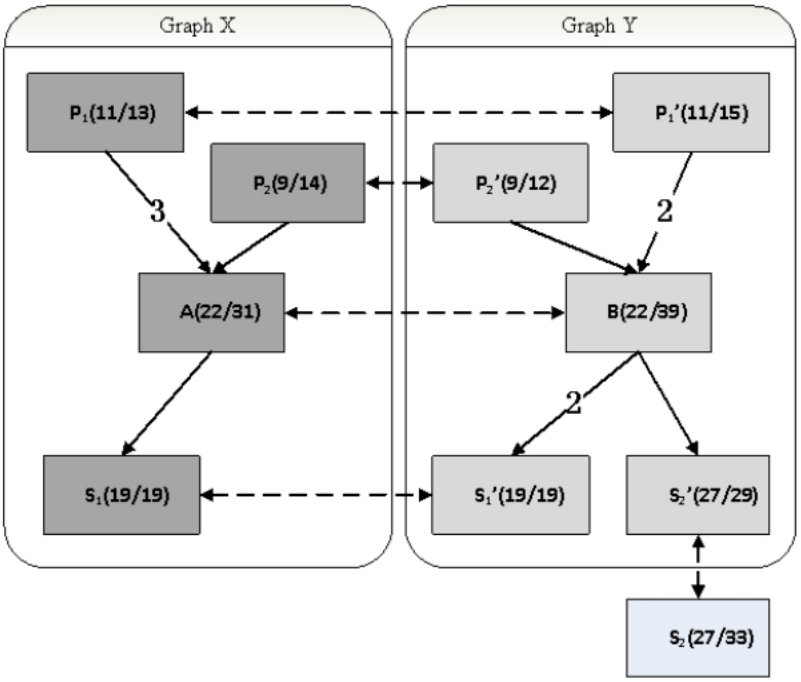
Membandingkan fungsi dengan callgraph

Binary file dapat direpresentasikan dengan callgraph yang menunjukkan bahwa setiap fungsi tidak terpisah-pisah melainkan terhubung satu dengan yang lainnya. Tidak seperti CFG, node dalam callgraph tidak dapat diurutkan karena address awal tidak reliable. Informasi yang diperoleh dari callgraph digunakan untuk menghitung kesamaan antar graph. Misalnya terdapat dua graph X dan Y, kesamaan antar graph tersebut dihitung dengan rumus sebagai berikut.

$$Similarity(X,Y) = \frac{2 \times \sum_{i,j} length(LCS(X_i, Y_j))}{\sum_i length(X_i) + \sum_j length(Y_j)}$$

Gambar 3 merupakan contoh graph dari dua fungsi A dan B dimana node sebelum dan setelahnya dilakukan mapping yang ditunjukkan dengan garis putus-putus. Angka di dalam kurung pada setiap fungsi menyatakan panjang LCS dan panjang fungsi secara berurutan. Berdasarkan rumus diatas, kesamaan dua graph tersebut dihitung dengan cara sebagai berikut.

$$Similarity(X,Y) = \frac{2 \times (11 \times 2 + 9 + 22 + 19)}{(13 \times 3 + 14 + 31 + 19 + 15 \times 2 + 12 + 39 + 19 \times 2 + 29)} = 0.573705$$



Gambar 3. Contoh kesamaan antar graph [10]

Berdasarkan dua kesamaan diatas, perbandingan dilakukan dengan cara berikut.

1) Insisialisasikan ambang batas yaitu 1.0.
2) Tentukan M yang merupakan himpunan semua mapping yang mungkin.
3) Ambil semua elemen M yang memiliki nilai kesamaan yang lebih tinggi dari pada ambang batas. Semua element tersebut disebut sebagai himpunan W.
4) Untuk setiap elemen W, hitung kesamaan antar graph sebagai *Value*.
5) Mapping yang memiliki *Value* tertinggi di dalam W ditetapkan sebagai *accepted*. Hapus semua mapping lain di dalam W dan M yang dimulai atau diakhiri oleh fungsi yang sama dengan mapping yang *accepted*.
6) Kembali ke langkah 3) hingga W menjadi kosong.
7) Kurangi ambang batas sebesar 0.1, kemudian ke langkah 2) hingga ambang batas menjadi 0 atau M menjadi kosong.

Pada awalnya, karena belum ada mapping yang dibuat, kesamaan antar graph akan sama dengan kesamaan antar fungsi. Oleh karena itu, inisialisasi ambang batas ditentukan setinggi mungkin yaitu 1.0 untuk memastikan bahwa mapping dibentuk pada fungsi yang benar-benar sama. Selama berjalannya proses perbandingan, ambang batas dikurangi sebesar 0.1 untuk membentuk mapping antar fungsi yang memiliki kesamaan yang lebih rendah.

Audit perbedaan semantik

Perbedaan semantik pada patch dapat disebabkan oleh penambahan fitur pada program, bukan karena kelemahan keamanan. Untuk menentukan apakah perbedaan semantik terkait dengan kelemahan keamanan atau bukan, audit perlu dilakukan pada perbedaan semantik tersebut. Sebagai contoh, data yang rusak akan diperbaiki dengan penambahan fungsi 'strlen' pada versi patch. Ini berarti bahwa kode program ini menjadi lebih aman dan kode yang bersesuaian pada versi yang tidak dilakukan patch menjadi berbahaya.

Pendekatan audit dilakukan dengan langkah-langkah sebagai berikut.

1) Insisialisasikan *M* sebagai himpunan semua fungsi yang vulnerable pada versi file terdahulu.
2) Ambil elemen dari *M* sebagai *v*.
3) Temukan semua path yang mungkin yang berisi sedikitnya satu perbedaan terhadap kode yang vulnerable di *v*.
4) Telusuri setiap path. Jika perbedaan tersebut dibuat untuk memperbaiki data yang rusak, maka tandai *v* dan fungsi yang bersesuaian sebagai *dangerous*.
5) Kembali ke langkah 2) hingga *M* menjadi kosong.

Melalui informasi yang diperoleh dari perbedaan dua file patch, kelemahan keamanan atau bugs suatu software dapat diketahui. Deskripsi kelemahan keamanan dibuat berdasarkan patch secara otomatis dan metode fuzzy dilakukan untuk menggali bugs yang tidak diketahui [11].

Rekomendasi

Secure update memiliki banyak tantangan dari sisi teknis, ekonomi, dan sosial. Beberapa rekomendasi untuk menjamin secure update sebagai berikut [8].

Membangun standard untuk secure update

Software berskala kecil cenderung menggunakan metode update software dengan caranya sendiri. Biaya pengembangan secure update system terlalu besar untuk setiap proyek perangkat lunak. Hal ini akan jauh lebih baik jika setiap pengembang perangkat lunak untuk membangun standar terbuka untuk melakukan secure update. Sangat tidak efektif jika setiap perusahaan menciptakan secure channel sendiri, tidak menggunakan salah satu dari sistem yang telah

dipahami dengan baik seperti SSL/TLS atau IPSec. Sebuah badan standardisasi perlu mengambil tindakan untuk kepentingan semua pihak.

Secure notification

Dari sudut pandang client, update software terdiri dari dua operasi, yaitu notifikasi keberadaan update dan instalasi update itu sendiri. Diduga bahwa banyak sistem notifikasi rentan terhadap penyerang yang mengelabui client sehingga mereka tidak mengetahui bahwa terdapat update yang tersedia. Serangan MITM dapat dengan mudah merespon semua permintaan update dengan "Tidak ada update yang tersedia saat ini." SFSRO mencakup autentikasi direktori dari semua file dalam file system. Dengan demikian, client dapat memverifikasi bahwa server merespon dengan "Tidak ada update yang tersedia" adalah memang sesuatu yang sebenarnya.

Mengikuti prinsip desain terbuka

Sebuah prinsip desain terbuka menyatakan bahwa lebih baik untuk merancang sebuah sistem yang aman dengan teknologi yang dapat ditinjau oleh publik dari pada dengan teknologi buatan sendiri.

Mengasumsikan infrastruktur tidak aman

Semakin besar jaringan internet, semakin besar dan beragamnya infrastruktur jaringan. File system dapat beroperasi pada server yang tidak terpercaya keamanannya, dimana server adalah salah satu dari banyak komponen yang tidak dapat dipercaya. Untuk memastikan secure update, dalam membangun rancangan sebaiknya memiliki asumsi untuk tidak percaya pada infrastruktur. Mekanisme update harus tetap melakukan verifikasi pada autentikasi update end-to-end.

Kesimpulan

Sistem yang tidak dilakukan patch, rentan untuk dieksploitasi oleh attacker. Patch dirancang untuk memperbaiki dan memperbaharui sistem. Namun demikian, patch dapat menyebabkan ketidakstabilan sistem. Oleh karena itu, perlu perencanaan yang baik dalam memilih metode apa yang akan digunakan untuk melakukan proses patch. Manajemen patch dilakukan untuk menggunakan rencana dan strategi dalam memilih metode patch yang akan digunakan. Manajemen patch terdiri dari tiga tahapan yaitu notification, scheduling, dan deployment and

post-deployment. Metode patch lainnya juga dibahas yaitu Shadow Patching, TeskTalk, pendekatan teori game, dan patch dinamis pada embedded software.

Proses update pada beberapa software masih memiliki kerentanan terhadap serangan MITM dan tidak dilakukan verifikasi dengan digital signature. Hal ini dapat menjadi backdoor untuk menyebarkan kode program berbahaya oleh attacker. Selain itu, patch dapat digunakan oleh attacker untuk melakukan penyerangan. Patch yang memiliki kelemahan keamanan dapat diketahui dengan menganalisis dua versi executable file. Tidak hanya itu, melalui patch juga dapat diketahui kelemahan keamanan atau bugs suatu software.

Rekomendasi diberikan untuk membangun sistem update software yang aman yaitu dengan membuat standar untuk secure update, notifikasi update yang aman, menggunakan prinsip desain terbuka, dan menggunakan asumsi bahwa insfrastruktur tidak aman.

Referensi

[1] H. Huang, S. Baset, C. Tang, A. Gupta, K. N. M. Sudhan, F. Feroze, R. Garg and S. Ravichandran, "Patch Management Automation for Enterprise Cloud," *IEEE Network Operations and Management Symposium,* 2012.

[2] W. Stallings and L. Brown, "Operating System Security," in *Computer Security Principles and Practice*, New Jersey, Pearson Education, Inc., 2014.

[3] D. Le, J. Xiao, H. Huang and H. Wang, "Shadow Patching: Minimizing Maintenance Windows in a Virualize Enterprise Environment," *10th CNSM and Workshop,* 2014.

[4] C. Liu and D. J. Richardson, "Automated Security Checking and Patching Using TestTalk," in *Proceedings ASE*, 2000.

[5] G. Gianini, M. Cremonini, A. Rainini, G. L. Cota and L. G. Fossi, "A Game Theoretic Approach to Vulnerability Patching," *ICTRC2015,* 2015.

[6] M. Ekman and H. Thane, "Dynamic Patching of Embedded Software," *RTAS07,* 2007.

[7] G. Brandman, "Patching the enterprise," *ACM Queue,* March 2005.

[8] A. Bellissimo, J. Burgess and K. Fu, "Secure Software Updates: Disappointments and New Challenges," *HotSec06,* 2006.

[9] D. Brumley, P. Poosankam, D. Song and J. Zheng, "Automatic patch-based exploit generation is possible: techniques and implications," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, 2008.

[10] Y. Song, Y. Zhang and Y. Sun, "Automatic vulnerability locating in binary patches," *International Conference on Computational Intellegence and Security,* 2009.

[11] S. Letian, F. Jianming, C. Jing and P. Guojun, "PVDF: An Automatic Patch-Based Vulnerability Description and Fuzzing Method," *CSC,* 2014.

[12] D. Callahan, "Construction the procedure call multigraph," *IEEE Transaction on Software Engineering,* 1990.