1st International Conference on new achievements in Science & Technology
24th August 2017
اولین کنفرانس بین المللی دستاوردهای نوین در علوم و تکنولوژی
۲ شهریور ماه ۱۳۹۶ – تهران

چرخه توسعه امنیت نرم‌افزار برای توسعه چابک(Agile-SDL)

محمد علی ترکمانی۱و۲*، ریحانه نوروزی۳، و هادی ناصری۴

۱ دانشجوی دکتری، گروه مهندسی کامپیوتر، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران،
Torkamani_ali@yahoo.com
۲ واحد پژوهش کارخانجات مخابراتی ایران، شیراز، ایران،
torkamani@itmc.ir
۳ باشگاه پژوهشگران جوان و نخبگان، نی‌ریز دانشگاه آزاد اسلامی، نی‌ریز، ایران،
noroozireihaneh@gmail.com
۴ باشگاه پژوهشگران جوان و نخبگان، استهبان دانشگاه آزاد اسلامی، استهبان، ایران
Nasseri_hadi@yahoo.com

چکیده

امروزه متدولوژیهای چابک به دلیل مزایای زیادی که دارند بسیار محبوب هستند و در پروژه های مختلف خصوصاً پروژه های کوچک و متوسط مورد استفاده قرار میگیرند. اما در توسعه نرم‌افزار با روش‌های چابک، به امنیت توجه کافی نشده است. به دلیل اینکه روش‌های چابک، سریعاً بر روی مشخصه‌های در حال ساخت متمرکز می‌شوند که نیازهای مستقیم مشتری را برآورده می‌سازد و امنیت هم یک نیاز مشتری است، عدم چشم‌پوشی از امنیت حائز اهمیت است. در این مقاله ترکیب متدولوژی SDL مایکروسافت که یک متدولوژی کامل برای توسعه نرم افزار امن است با متدولوژی های چابک توسعه نرم افزار به‌گونه‌ای که اصول هر دو (فرآیند SDL و روش‌های چابک) برقرار باشد مورد بررسی قرار میگیرد. این راهکار برای تیم‌های توسعه نرم افزار که می‌خواهند برنامه‌های کاربردی امن‌تری با استفاده از روش‌های چابک بسازند مناسب خواهد بود.

واژه‌های کلیدی: مهندسی نرم افزار امن، چرخه توسعه امنیت نرم افزار(SDL) ، توسعه چابک، SDL چابک(Agile-SDL).

1st International Conference on new achievements in Science & Technology
24th August 2017
اولین کنفرانس بین المللی دستاوردهای نوین در علوم و تکنولوژی
۲ شهریور ماه ۱۳۹۶ – تهران

Security Development Lifecycle for Agile Development (Agile-SDL)

Mohammad ali, Torkamani [1,2*]; Reihaneh, noroozi[3]; Hadi, Nasseri[4]

1- Ph.D. student ,Department of Computer Engineering, Yasouj Branch, Islamic Azad University, Yasouj, Iran, *Torkamani_ali@yahoo.com*
2-*R&D Department, Iranian Telecommunication Manufacturing Company, Shiraz, Iran,* torkamani@itmc.ir
3- Department of Computer Engeenier, neyriz Branch, Islamic Azad University, Neyriz ,Iran, *noroozireihaneh@gmail.com*
4- Department of Computer Engeenier, Estahban Branch, Islamic Azad University, Estahban ,Iran, *Nasseri_hadi@yahoo.com*

Abstract

Nowadays, Agile methodologies are popular due to big advantages and are applied in software development particularly in various small and medium software projects. But security has not been given the attention it needs when developing software with Agile methods. Since Agile methods focus on rapidly creating features that satisfy customers' direct needs, and security is a customer need, it's important that it not be overlooked .The goal of Agile-SDL is to meld the proven Microsoft Security Development Lifecycle (SDL) with Agile methodologies in a way that maintains the principles of both the Agile methods and the SDL process. In this article a survey of Agile-SDL is presented. This approach is suitable for software development teams who are trying to build secure application programs with agile methods.

Keywords: *Secure software engineering, SDL, Agile development, Agile-SDL.*

۱- مقدمه

بسیاری از سازمان‌های توسعه، برای ساخت برنامه‌های کاربردی‌شان، از روش‌های مدیریت و توسعه نرم‌افزاری چابک استفاده می‌کنند. در گذشته، امنیت در توسعه نرم‌افزار با روش‌های چابک، به اندازه کافی مورد توجه قرار نگرفته است. به دلیل اینکه روش‌های چابک، بر روی مشخصه‌های در حال ساخت که نیازهای مستقیم مشتری را برآورده می‌سازد، متمرکز می‌شوند و امنیت هم یک نیاز مشتری است، عدم چشم‌پوشی از امنیت حائز اهمیت است. در دنیای بسیار بهم پیوسته امروزی، هر جا نیازمندی‌های حریم خصوصی و قانونی قوی برای حفاظت از داده‌های خصوصی وجود دارد امنیت باید به عنوان یک اولویت مهم در نظر گرفته شود. امروزه این احساس وجود دارد که روش‌های چابک، کدهای امنی را به وجود نمی‌آورند؛ تحلیل‌های بیشتر نشان می‌دهد که این احساس واقعیت دارد. در بازار امروز، به روش‌های چابک امن به‌صورت تخصصی پرداخته نشده است. این مسئله خوشایند نبوده و باید تغییر کند. تنها راه تغییر روشهای چابک درک نیاز به روشهای چابک امن و انجام اقدامات موثر برای ادغام نیازمندی‌های امنیتی با روش‌های توسعه چابک است[۷،۸]..

متخصصان مهندسی نرم‌افزار متدولوژی‌های مختلفی را برای توسعه نرم‌افزار امن ارائه نموده‌اند که معروف‌ترین آنها متدولوژی MCGraw [۱،۳] و SDL مایکروسافت [۲،۳،۷،۸،۱۰] است. این متدولوژی‌ها در فازهای مختلف توسعه نرم‌افزار تکنیک‌ها و ابزارهای خاصی را برای تامین امنیت نرم‌افزار ارائه نموده‌اند [۴]. مدلسازی UMLSec در فازهای تحلیل و طراحی توسعه نرم‌افزار امن [۴،۱۱،۱۲]، استفاده از ابزارهای مدل‌سازی دیگر نظیر نمودار Abuse case که سناریوی استفاده کاربر و سوءاستفاده نفوذگر از سیستم را به صورت همزمان نشان می‌دهد [۳،۴]، مدل‌سازی تهدیدها با استفاده از رهیافت STRIDE مایکروسافت [۵،۸،۹] که در واقع نمودار آن نیز نوعی DFD توسعه‌یافته برای نشان دادن تهدیدها است [۵]، ترسیم درخت‌های حمله که یک نوع درخت AND-OR است [۵]، مرور کد در فاز کدنویسی [۸،۹]، انجام تست نفوذ در انتهای مراحل توسعه نرم-افزار [۳،۸،۹] و مدیریت ریسک‌های امنیتی در سرتاسر چرخه حیات نرم‌افزار [۴،۷] از جمله فعالیت‌های مهمی است که باید در توسعه نرم‌افزار انجام شود.

مایکروسافت به ایجاد مجموعه‌ای از اصلاحات به منظور دستیابی به امنیت در فرآیند توسعه نرم‌افزار به نام SDL مبادرت ورزیده است [۷،۸]. متدولوژی SDL مایکروسافت بر مبنای اصول راهنمای امنیت و حریم خصوصی بنا شده است. پیروی از این اصول می‌تواند منجر به تولید نرم‌افزاری امن و قابل اعتماد گردد [۶]. SDL تعداد آسیب‌پذیری‌ها در نرم‌افزار در حال ارسال را تا بیش از ۵۰ درصد کاهش می‌دهد. اما از نظر متدولوژی چابک، روش SDL بسیار سنگین است زیرا برای حفظ امنیت محصولات بسیار عظیم مثل Windows و Microsoft Office طراحی‌شده است که هر دو دارای چرخه توسعه طولانی هستند [۷]. اگر کاربران روش‌های چابک بخواهند از SDL استفاده کنند باید دو تغییر ایجاد کنند. اول اینکه فزودن SDL به فرآیندهای چابک باید اندک باشد. یعنی تیم توسعه برای هر ویژگی نرم افزار ، فعالیت‌های SDL را به‌اندازه کافی انجام دهد قبل از اینکه شروع به کار روی ویژگی دیگر کند. دوم اینکه مراحل توسعه‌ی (طراحی، اجرا، وارسی و انتشار) مرتبط با SDL در یک قالب سریع‌تر سازماندهی شوند زیرا روش‌های چابک را به کار نگرفته‌اند. برای این منظور تیم SDL مایکروسافت یک روش موثر سریع و امن را توسعه داده و به کار گرفته است که چرخه توسعه امنیت برای توسعه چابک یا SDL-Agile نام دارد [۸].

در این مقاله چرخه توسعه امنیت برای توسعه چابک[1] مورد بررسی قرار میگیرد.

[1] Agile Development

ترکیب SDL و توسعه چابک

چون چرخه‌های انتشار سریع، یک هفته طول می‌کشند، بنابراین تیم‌ها برای تکمیل تمام نیازمندی‌های SDL در هر انتشار زمان کافی ندارند. از طرف دیگر، مسائل کلیدی امنیتی وجود دارد که SDL باید آنها را برآورده کند و نباید برای هیچ انتشاری حتی مسایل کوچک نادیده گرفته شوند. ادغام این دو تکنیک آنچنان هم دشوار نیست، در قلب آن، SDL وظایفی را تعریف می‌کند و این وظایف می‌تواند توسط یک فرآیند توسعه چابک ترسیم گردد [۸].

۲- نیازمندی‌های SDL چابک

یک اسب بارکش[1] برای توسعه چابک اسپرینت[2] است؛ اسپرینت یک دوره زمانی کوتاه معمولاً ۱۵ تا ۶۰ روز است که در آن مجموعه‌ای از ویژگی‌ها و داستانها، طراحی، توسعه و آزمایش می‌شوند و سپس به‌طور بالقوه به مشتریان تحویل داده می‌شوند. لیست ویژگی‌های اضافه‌شده به محصول، backlog محصول نامیده می‌شود و قبل از شروع یک اسپرینت فهرستی از مشخصه‌ها از backlog محصول انتخاب می‌گردد و به backlog اسپرینت اضافه می‌شود. SDL با این مفهوم همخوانی دارد. نیازمندی‌های SDL به‌عنوان وظایف ارائه می‌شوند و به backlog های محصول و اسپرینت اضافه می‌گردند. سپس این وظایف توسط اعضای تیم جهت تکمیل انتخاب می‌شوند. شما می‌توانید وظایف کوچک SDL را که به backlog اضافه می‌شوند داستان‌های غیرکارکردی[3](کیفی) در نظر بگیرید. برای تطبیق نیازمندی‌های سنگین SDL با چارچوب سبک چابک، SDL چابک، هر نیازمندی و توصیه SDL را در یکی از سه دسته تعریف‌شده زیر قرار می‌دهد [۸].

۱-۳ نیازمندی‌های مخصوص هر اسپرینت

اولین دسته، از نیازمندی‌های SDL تشکیل‌شده که برای امنیت بسیار ضروری هستند و هیچ نرم‌افزاری بدون برخورداری از آن‌ها نباید انتشار یابد. این دسته، دسته مخصوص هر اسپرینت نامیده می‌شود. طول یک اسپرینت هر اندازه که باشد (چه دو هفته چه دو ماه)، باید تمام نیازمندی‌های SDL در دسته مخصوص هر اسپرینت را برآورده سازد در غیر اینصورت اسپرینت ناقص فرض شده و نرم‌افزار نمی‌تواند انتشار یابد. این مطلب شامل همه انتشارهای نرم‌افزار به مخاطبان می‌شود، انتشار یک محصول برای ساخت[4]،انتشار سرویس‌های آنلاین به وب[5] و انتشار پیش دید آلفا/ بتا[6]. نمونه‌هایی از نیازمندی‌های مخصوص هر اسپرینت عبارتند از:

- اجرای ابزارهای تحلیل به‌صورت روزانه یا برای هر ساخت.
- مدل‌های تهدید تمام ویژگی‌های جدید.
- اطمینان از اینکه هر عضو پروژه حداقل یک واحد آموزش امنیت را در سال گذشته به اتمام رسانده است.
- استفاده از کتابخانه‌های فیلترکردن ورودی‌ها و فرار کردن[7] از خروجی‌ها برای برنامه‌های تحت وب. فیلتر کردن ورودی-ها و فرار کردن از خروجی‌ها دو اصل اساسی در برنامه‌نویسی وب است که بدون رعایت آنها برنامه شما مورد سوء استفاده هکرها قرار میگیرد.
- استفاده از رمزنگاری قوی در کدهای جدید (AES ، RSA و SHA-256 یا بهتر).

---

[1] workhorse
[2] sprint
[3] non-functional stories
[4] released to manufacturing (RTM)
[5] released to Web (RTW)
[6] alpha/beta preview release
[7] filtering and escaping libraries

۳-۲ نیازمندی‌های bucket

دسته دوم نیازمندی‌های SDL از وظایفی تشکیل‌شده که باید به طور منظم در طول عمر پروژه اعمال شوند اما به اندازه‌ای بحرانی نیستند که برای هر اسپرینت الزامی باشند. این دسته، دسته bucket نامیده شده و به سه واحد جداگانه از وظایف تقسیم می‌شود که عبارتند از: وظایف وارسی (آزمایش‌کننده‌های فازی و سایر ابزارهای تحلیل)، وظایف مرور طراحی و وظایف برنامه‌ریزی. تیم‌های تولید باید در هر اسپرینت، بجای تکمیل تمام نیازمندی‌های bucket، فقط یک نیازمندی SDL از هر bucket را تکمیل کنند [۸]. جدول زیر حاوی نمونه‌ای از وظایف هر واحد است.

جدول ۱: نمونه‌ای از دسته‌های bucket [۱۳].

وظایف وارسی	وظایف مرور طراحی	وظایف برنامه‌ریزی
ActiveX fuzzing	هدایت یک مرور طراحی	ایجاد اسناد پشتیبان حریم خصوصی
تحلیل سطح تهاجم	مرور طراحی رمز شده[1]	به‌روزرسانی مسئولان پاسخگویی امنیتی
تحلیل باینری (BinScope)	نام‌گذاری اسمبلی و APTCA	به‌روزرسانی طرح down شبکه
آزمایش File fuzz	کنترل حساب کاربری	تعریف/ به‌روزرسانی نوار باگ‌های امنیتی

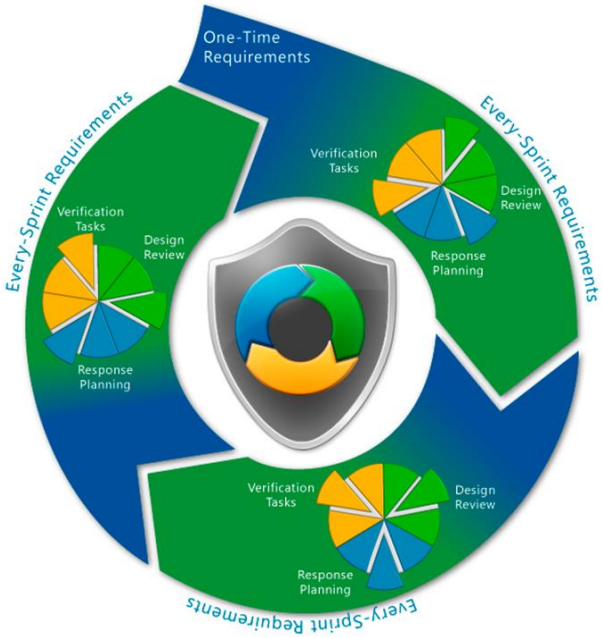
به عنوان نمونه تیم باید یک نیازمندی وارسی، یک نیازمندی مرور طراحی و یک نیازمندی برنامه‌ریزی در هر اسپرینت (علاوه بر نیازمندی‌های مخصوص هر اسپرینت که قبلاً بحث شد) را تکمیل است. برای اسپرینت یک، تیم ممکن است آزمایش ActiveX fuzzing، مرور طراحی و بروزرسانی نوار باگ‌های امنیتی از جدول را انتخاب کند. برای اسپرینت دو، تیم می‌تواند تحلیل باینری، هدایت یک مرور طراحی و بروزرسانی طرح شبکه را انتخاب کند. تعیین نوع وظایف از هر bucket برای هر اسپرینت به عهده تیم‌های تولید است [۸]. SDL سریع، هیچ دخالتی در اولویت‌بندی وظایف برای این نیازمندی‌ها ندارد. مثلا اگر تیم معین کرد که بهتر است نیازمندی file fuzzing برای هر دو اسپرینت یک بار انجام شده ولی آزمایش SOAP fuzzing لازم است برای هر ده اسپرینت یک بار انجام شود، این مطلب قابل‌قبول است. لازم به ذکر است که هیچ نیازمندی نباید کاملاً نادیده گرفته شود. مشخص است که هر نیازمندی در SDL تعدادی از مسائل امنیتی یا حریم خصوصی یا هر دو را شناسایی یا منع می‌کند. بنابراین هیچ نیازمندی SDL مربوط به نیازمندی‌های واحدها نمی‌تواند بیشتر از شش ماه ناقص بماند [۸].

۳-۳ نیازمندی‌های یک‌باره

تعدادی از نیازمندی‌های SDL هستند که هنگام شروع یک پروژه جدید با SDL سریع، موردنیاز می‌باشند [۱۴]. این وظایف فقط یک بار[2] برای پروژه مطرح شده و بعد از اتمام نیازی به تکرار آن‌ها نیست. این دسته از نیازمندی‌های SDL سریع، نیازمندی‌های یک‌باره[3] نامیده می‌شوند. بهتر است نیازمندی‌های یک‌باره به سادگی و با سرعت تکمیل شوند به‌استثناء "ایجاد یک مدل تهدید پایه". این وظایف کوتاه هستند اما ممکن است در یک پروژه تعداد زیادی از آنها وجود داشته باشد، بنابراین چون هنگام شروع SDL سریع، تیم باید نیازمندی‌های مخصوص هر اسپرینت و یک نیازمندی از هر bucket را تکمیل کند، تکمیل همه این وظایف در یک اسپرینت عملی نخواهد بود [۷،۸]. SDL سریع جهت رفع این مشکل، برای تکمیل هر نیازمندی یک‌باره یک دوره مهلت می‌دهد. این دوره (مهلت) بسته به اندازه و پیچیدگی نیازمندی، عموماً بین یک ماه تا یک سال بعد از شروع پروژه متغیر است. برای مثال انتخاب یک مشاور امنیتی، وظیفه آسانی بوده در مدت یک ماه باید انجام شود، اما بروزرسانی

[1] Review crypto design
[2] One-Time Requirements
[3] one-time

پروژه برای استفاده از آخرین نسخه کامپایلر یک وظیفه طولانی و دشوار است و یک مهلت یک‌ساله برای تکمیل دارد. شکل ۱ این فرآیند را نشان می‌دهد [۷،۸].



شکل۱: فرآیند SDL سریع [۱،۲].

۴-۳ محدودیت‌ها

مشکل اصلی که برای SDL سریع وجود دارد، تطبیق کل SDL به یک چرخه انتشار کوتاه مدت است، زیرا هر نیازمندی SDL طی یک چرخه انتشار دو یا سه ساله تکمیل می‌شود. بنابراین منطقی نیست که این کار برای یک چرخه نشر دو یا سه هفته‌ای انجام شود. دسته‌بندی نیازمندی‌های SDL به سه دسته مخصوص هر اسپرینت[۱]، یک‌باره[۲] و سه bucket ای[۳] راه‌حل SDL سریع برای کنار آمدن با این مسئله بغرنج است. اما تأثیر این دسته‌بندی می‌تواند منجر به رها شدن موقت بعضی از نیازمندی‌های SDL برای بعضی از انتشارها گردد. تیم SDL مایکروسافت معتقد است که این ایده، برای تیم‌ها با چرخه‌های انتشار کوتاه موقعیت خوبی است و بهترین ترکیب از امنیت، توسعه مشخصه‌ها و سرعت انتشار را بهمراه دارد. با وجودی که SDL سریع برای تیم‌هایی با چرخه‌های انتشار کوتاه طراحی شده است اما تیم‌های با چرخه‌های انتشار طولانی‌تر هم می‌توانند از فرآیند SDL سریع استفاده کنند. اما اعمال امنیتی یا نیازمندی‌هایی که یک تیم، فقط یک مرتبه نیاز به تکمیل آن‌ها در SDL سبک آبشاری کلاسیک دارد، ممکن است در SDL سریع، پنج یا شش دفعه (یا بیشتر) در طول دوره یک پروژه طولانی تکرار شود [۷،۸].این مسئله لزوماً چیز بدی نیست و به ایجاد یک محصول امن‌تر کمک می‌کند.

4- به‌کارگیری وظایف SDL برای اسپرینت‌ها

بخش قبل بر روی نیازمندی‌های ویژه SDL سریع متمرکز بود. این بخش وظایف مرتبط با SDL و چگونگی به‌کارگیری آن‌ها در چارچوب توسعه چابک را مورد توجه قرار می‌دهد.

---

[1] every-sprint
[2] one-time
[3] three bucket

۴-۱ آموزش امنیت

هر عضو از تیم پروژه باید حداقل یک واحد آموزش امنیت را در هر سال به اتمام برساند. اگر بیش از ۲۰ درصد از اعضای یک پروژه این نیازمندی را برآورده نکنند، نیازمندی با عدم موفقیت مواجه می‌شود (و در نتیجه اسپرینت هم با شکست مواجه شده و محصول اجازه انتشار ندارد). برای تهیه فهرستی از واحدهای مربوط به نیازمندی‌های آموزش SDL، به رهبر اسپرینت خود مراجعه کنید. شما همچنین می‌توانید برای توصیه‌ها و واحدهای آموزشی به SDL Pro Network مراجعه کنید. به علاوه، مهندسان و آزمایش‌کنندگان که وظایف مربوط به امنیت یا SDL را انجام می‌دهند بهتر است قبل از اعمال این وظایف بر روی اسپرینت، دانش امنیتی لازم را کسب کنند. در این حالت، آموزشهای مناسب آموزشهایی هستند که مربوط به مفاهیم امنیتی مشخصه‌های توسعه‌یافته یا آزمایش‌شده در اسپرینت می‌باشند. نمونه‌هایی از این موارد عبارتند از آسیب‌پذیری‌های تزریق XSS، آسیب‌پذیری‌های تزریق SQL، سرریزهای‌بافری، سرریزهای اعداد صحیح، اعتبار سنجی ورودی‌ها، مسائل ویژه زبان‌ها (PHP، Java، C#) و خطاهای رایج رمزنویسی. اگر کسی در تیم توسعه بخواهد نقش رهبر امنیتی یا متخصص امنیتی را بر عهده بگیرید بهتر است آموزش‌های امنیتی عمیق‌تر و وسیع‌تری را به‌عنوان بخشی از آموزش‌های مداوم کسب کند. در دسترس بودن متخصص امنیتی برای تیم و مهم‌تر از آن برای مشتری یک مزیت محسوب می‌شود.

۴-۲ تجهیز و اتوماسیون

از نظر یک فرآیند امنیتی موفق، ابزارهایی که وظایف امنیتی را به طور خودکار انجام می‌دهند، حائز اهمیت هستند زیرا هر چه اعمال مربوط به نیازمندی‌ها بیشتر خودکارسازی شوند امنیت بهتر می‌شود. البته هیچ ابزاری نمی‌تواند در برقراری امنیت جایگزینی انسان باشد. استفاده از ابزار مقیاس‌پذیری را فراهم می‌کند زیرا یک ابزار می‌تواند بدون خستگی و اشتباه تعداد زیادی کد را مورد پویش قرار دهد یا کدهای باینری را بررسی کند. به‌هرحال در نظر داشته باشید که فقط اجرای ابزارها، یک محصول نرم‌افزاری را امن نمی‌سازد. SDL سریع نیازمند یک بار اجرای ابزارهای زیر برای هر اسپرینت است و توصیه می‌کند که آن‌ها روزانه یا به عنوان بخشی از فرآیند ساخت و بررسی اجرا گردند:

۱. کدهای .NET. که عبارتند از:

- CAT.NET: یک ابزار تحلیل باینری کد است که به توسعه‌دهنده کمک می‌کند انواع آسیب‌پذیری‌های رایج نظیر XSS، تزریق SQL و تزریق Xpath را پیدا کنند [۱۵].
- FxCop: یک ابزار تحلیل باینری است که تجمع‌های چارچوب .NET. کامپایل شده را برای نقاط ضعف رایج برنامه‌نویسی امتحان می‌کند. FxCop علاوه‌بر بررسی‌های امنیتی، دامنه‌هایی مانند عملکرد، محل‌سازی و ... را بررسی می‌کند. مایکروسافت ابزار Fxcop را بصورت داخلی در ابزار تحلیل کد ویژوال استودیو قرار داده است. البته ابزارFxCop قابل دانلود و قابل استفاده می‌باشد [۱۶].

۲. کدهای محلی: مسائلی که توسط ابزار تحلیل کدهای استاتیک برای کدهای مدیریت نشده شناخته شده‌اند را ترمیم کنید. در Microsoft Visual Studio همراه با کامپایلر C و C++ از سوئیچ /analyze برای کدهای محلی درسیستم‌عامل‌های هدف استفاده کنید و تمامی هشدارهای کمینه موردنیاز (Min SDL) را برطرف سازید [۷،۸].

۴-۳ مدل‌سازی تهدید: بنیاد و اساس SDL

مدل تهدید باید به عنوان یک پایه و اساس برای محصول در نظر گرفته شود. چه محصول جدید باشد و چه قبلاً تحت توسعه‌یافته باشد، باید یک مدل تهدید به عنوان بخشی از اعمال طراحی اسپرینت ساخته شود. فرآیند مدل‌سازی تهدیدها بهتر است زمان‌بندی‌شده باشد و فقط محدود به بخش‌هایی از محصول باشد که در حال حاضر موجود هستند یا درحال توسعه می‌باشند. هنگامی‌که یک مدل تهدید موجود باشد، سایر اقدامات اضافی برای به‌روزرسانی مدل تهدید، نیازمند تغییرات ناچیز و تدریجی خواهند بود. یک مدل تهدید قسمت مهمی از فرآیند امن‌سازی محصول است زیرا یک مدل تهدید خوب به موارد زیر کمک می‌کند:

۱. تعیین مسائل بالقوه طراحی امنیت.
۲. انجام تحلیل سطح تهاجم و مؤلفه‌های در معرض بیشترین ریسک.
۳. انجام فرآیند آزمایش فاز.

بهتر است مدل تهدید برای ارائه تمام مشخصه‌ها و قابلیت‌های جدید که در هر اسپرینت اضافه می‌شوند به‌روز شود. حتی اگر قابلیت‌ها بدون تغییر بمانند بهتر است مدل تهدید برای ارائه تمام تغییرات اساسی طراحی به‌روز شود [۷،۸]. مدلسازی تهدیدها بر مبنای رهیافت STRIDE انجام می‌شود. جدول ۲ شرح می‌دهد که STRIDE به چه معنا است و ویژگی هایی که به چالش کشیده می‌شوند، چیست [۵،۸،۹].

جدول ۲ : واژه STRIDE [۵،۸،۹]

ویژگی مورد نظر ما	تهدید
Authentication	Spoofing
Integrity	Tampering
Nonrepudiation	Repudiation
Confidentiality	Information Disclosure
Availability	Denial of Service
Authorization	Elevation of Privilege

ابزار پیشنهادی مایکروسافت برای مدلسازی تهدیدها MSThreatModelingTool [۱۷] است.

۴-۴ آزمایش fuzzing

آزمایش fuzzing یک تکنیک آزمون امنیتی شدیداً مؤثر است به ویژه هنگامیکه تا کنون از آزمایش fuzzing بر روی محصول استفاده نشده باشد. مدل تهدید باید تعیین کند چه بخش‌هایی از برنامه‌کاربردی تحت آزمایش fuzzing قرار گیرند. درصورتی‌که هیچ مدل تهدیدی وجود نداشته باشد، باید لیست آغازین موارد پر خطر را نمایش دهد. بعد از اینکه این لیست کامل شد نقاط ورود، مورد آزمایش fuzzing قرار می‌گیرند. برای مثال نقاط انتهایی تائید هویت نشده یا دور از دسترس، نسبت به نقاط انتهایی تایید هویت شده یا محلی دارای ریسک بیشتری هستند. حسن آزمایش fuzzing این است که هنگامی‌که یک کامپیوتر یا گروهی از کامپیوترها برای آزمایش fuzzing برنامه‌کاربردی پیکربندی می‌شوند می‌توانند به اجرا ادامه دهند؛ و فقط درهم‌شکستگی‌ها[1] نیاز به تحلیل دارند. اگر از ابتدای آزمایش fuzzing هیچ درهم‌شکستگی وجود نداشته باشد، احتمالاً آزمایش

[1] crash

fuzzing کافی نبوده و باید یک وظیفه جدید برای تحلیل اینکه چرا آزمایش‌های fuzzing موفق نشده‌اند ایجاد شده و اصلاحات ضروری را تولید کند [۷].

۴-۵ استفاده از یک اسپایک

هنگامیکه باگ‌های امنیتی متراکم بوده و کدها در معرض خطر هستند از یک اسپایک[1] برای تحلیل و سنجش کدهای ناامن استفاده می‌شود. یک شاخص مهم برای تراکم اشکالات امنیتی سن کدها است. بر اساس تجربه‌های آزمایش‌کننده‌ها و توسعه‌دهندگان مایکروسافت، هر چه کد قدیمی‌تر باشد تعداد اشکالات امنیتی پیداشده در آن بیشتر است. اگر پروژه شما دارای تعداد زیادی کدهای خطرناک یا کدهای موروثی باشد بهتر است تا آنجایی که امکان دارد نقاط آسیب‌پذیر این کدها را بیابید. این کار توسط یک اسپایک صورت می‌گیرد. یک اسپایک، یک پروژه جانبی زمان‌بندی‌شده است که دارای هدف به‌خوبی تعریف‌شده‌ای است (یافتن اشکالات امنیتی). می‌توان این اسپایک را به عنوان یک ترغیب امنیتی کوچک در نظر گرفت. هدف ترغیب امنیتی در SDL، ارتقاء و به‌روزرسانی کدهای خطرناک در زمانی کوتاه نسبت به طول دوره پروژه است. توجه داشته باشید که ترغیب امنیتی تعمیر باگ‌های امنیتی را مطرح نمی‌کند و بیشتر برای تعیین میزان آسیب‌پذیر بودن اشکالات، آن‌ها را مورد تحلیل قرار می‌دهد. اگر تعداد زیادی اشکال امنیتی در کدهایی که دارای ارتباطات شبکه‌ای هستند یا داده‌های حساس را به کار می‌برند پیدا شد، این کدها نه تنها باید تعمیر گردند بلکه باید برای جستجوی آن‌ها در باگ‌های امنیتی یک اسپایک راه‌اندازی شود. نمونه‌هایی از تحلیل صورت گرفته طی یک اسپایک عبارتند از [۸]:

- تمام کدها: به جستجوی نقاط آسیب‌پذیر ورودی که منجر به سرریزهای بافری و اعداد صحیح می‌شوند بپردازید. همچنین کلمات عبور و استفاده از کلیدهای ناامن را در طول الگوریتم‌های رمزنویسی ضعیف جستجو کنید.
- کدهای وب: به جستجوی نقاط آسیب‌پذیر که در اثر اعتبارسنجی نامناسب ورودی کاربر به وجود می‌آیند مثل CSS بپردازید.
- کدهای پایگاه داده‌ای: به جستجوی آسیب‌پذیری‌های تزریق SQL بپردازید.
- ایمن برای کنترل‌های برنامه‌نویسی ActiveX: به‌مرور خطاهای C/C++، نشت اطلاعات و عملیات خطرناک بپردازید.

تمام ابزارهای مناسب تحلیل که در دسترس تیم هستند باید طی اسپایک اجرا گردند و تمام حفره‌ها اولویت‌بندی و ثبت شوند. حفره‌های امنیتی بحرانی مثل یک سرریز بافری در یک مؤلفه شبکه شده یا یک آسیب‌پذیری تزریق SQL باید به‌عنوان موارد برنامه‌ریزی نشده با اولویت زیاد در نظر گرفته شوند.

۴-۶ استثناءها

گردش کار استثناء[2] نیازمندی‌های SDL، در SDL سریع تا حدی متفاوت از SDL کلاسیک است. استثناءها در SDL کلاسیک برای کل دوره انتشار داده می‌شوند اما این مطلب برای پروژه‌های چابک عملی نیست. یک انتشار از پروژه سریع ممکن است فقط چند روز طول بکشد تا اسپرینت بعدی کامل شود، در این شرایط تجدید هر هفته استثناءها برای مدیران پروژه منجر به هدر دادن زمان خواهد شد. به این منظور، تیم‌های پروژه که SDL سریع را دنبال می‌کنند می‌توانند یک استثناء را در طول اسپرینت (که برای اسپرینت‌های طولانی‌تر به‌خوبی عمل می‌کند) یا برای یک مدت زمان مشخص نه بیشتر از شش ماه (که برای اسپرینت‌های کوتاه‌تر به‌خوبی عمل می‌کند) به کار گیرند. هنگام مرور استثناء نیازمندی، مشاور امنیتی می‌تواند بر اساس طول مدت درخواست شده استثناء، شدت استثناء را به اندازه یک سطح کم یا زیاد کند (بنابراین میزان ارشدیت مدیر که برای تائید استثناء موردنیاز است کم یا زیاد می‌شود). برای مثال یک تیم را در نظر بگیرید که برای یک نیازمندی با اولویت متوسط، درخواست یک استثناء می‌دهد که این استثناء مستلزم تائید مدیر است. اگر تیم استثناء را فقط برای یک دوره زمانی کوتاه

[1] Spike
[2] Exceptions

درخواست دهد مثلاً دو هفته، مشاور امنیتی ممکن است شدت را به کم کاهش دهد که فقط به تائید از جانب رهبر امنیتی تیم نیاز است. از طرف دیگر اگر تیم شش ماه کامل را درخواست دهد مشاور امنیتی ممکن است شدت را به مهم افزایش دهد که به دلیل افزایش ریسک، مستلزم پایان دادن از جانب مدیریت ارشد است. علاوه بر به‌کارگیری استثناءها برای نیازمندی‌های خاص، تیم‌ها نیز می‌توانند یک استثناء را برای کل یک bucket درخواست دهند. به طور معمول تیم‌ها باید حداقل یک نیازمندی از هر bucket را برای هر اسپرینت تکمیل کنند اما اگر یک تیم نتواند حتی یک نیازمندی از یک bucket را تکمیل کند تیم یک استثناء را برای پوشش کل bucket درخواست می‌دهد. تیم می‌تواند یک استثناء را برای طول اسپرینت یا برای یک دوره زمانی خاص (نه بیشتر از شش ماه) درخواست دهد. اما به دلیل گستردگی استثناء، استثناءهای مربوط به bucket به عنوان مهم تلقی شده و نیاز به تائید حداقل یک مدیر ارشد دارند [۷،۸].

۷-۴ مرور نهایی امنیت

مرور نهایی امنیت[1]، شبیه به مرور نهایی امنیت در SDL آبشاری کلاسیک، در انتهای هر اسپیرینت سریع مورد نیاز است اما مرور نهایی امنیت SDL سریع، محدود می‌باشد. مشاور امنیتی فقط نیاز است به مرور موارد زیر بپردازد:

- تمام نیازمندی‌های مخصوص هر اسپرینت کامل شده‌اند یا استثناءها برای آن نیازمندی‌ها اعطا شده است.
- حداقل یک نیازمندی از هر دسته از نیازمندی‌های bucketها کامل شده است (یا یک استثناء برای آن bucket اعطا شده است).
- هیچ یک از نیازمندی‌های bucketها بیش از شش ماه، ناقص رها نشده است (یا یک استثناء اعطا شده است).
- نیازمندی‌های یک‌باره، از دوره مهلت خود فراتر نروند (یا استثناءهایی اعطاشده‌اند).
- هیچ باگ امنیتی که از آستانه شدت تعیین شده بالاتر است، نباید باز باشد

بعضی از این وظایف ممکن است نیاز به تلاش از جانب خود مشاور امنیتی داشته باشد برای اینکه از کامل بودن آن‌ها اطمینان حاصل شود (برای مثال مدل‌های تهدید باید مرور گردند) اما در کل مرور نهایی امنیت مربوط به SDL سریع به‌طور قابل‌ملاحظه‌ای سبک‌تر از مرور نهایی امنیت مربوط به SDL کلاسیک است [۷،۸].

[1] Final Security Review(FSR)

نتیجه گیری

امروزه بسیاری از توسعه‌دهندگان از روش‌های چابک توسعه نرم‌افزار استفاده می‌کنند. اما در توسعه نرم‌افزار با روش‌های چابک، امنیت آنچنان‌که لازم است مورد توجه قرار نگرفته است. تقریبا در تمامی نرم‌افزارهای امروزی، امنیت باید به‌عنوان یک اولویت مهم در نظر گرفته شود ولی تحلیل‌های انجام شده توسط مایکروسافت نشان داده است که روش‌های چابک، کدهای امنی را به وجود نمی‌آورند. در بازار امروز، به روش‌های چابک امن به‌صورت تخصصی پرداخته نشده است. این مسئله خوشایند نبوده و باید تغییر کند. مایکروسافت به ایجاد مجموعه‌ای از اصلاحات فرآیند توسعه نرم‌افزار به نام SDL مبادرت ورزیده است. نشان داده‌شده که SDL تعداد آسیب‌پذیری‌ها در نرم‌افزار در حال ارسال را تا بیش از ۵۰ درصد کاهش می‌دهد. اما از نقطه نظر چابک، متدولوژی SDL بسیار سنگین است زیرا از ابتدا برای امن کردن محصولات بسیار عظیم مثل Windows و Microsoft Office طراحی‌شده است که هر دو دارای چرخه توسعه طولانی هستند.

چون چرخه‌های انتشار سریع، یک هفته طول می‌کشند، بنابراین تیم‌ها برای تکمیل تمام نیازمندی‌های SDL در هر انتشار زمان کافی ندارند. از طرف دیگر، مسائل کلیدی امنیتی وجود دارد که SDL باید آنها را برآورده کند و نباید برای هیچ انتشاری حتی مسایل کوچک نادیده گرفته شوند. ادغام این دو تکنیک آنچنان هم دشوار نیست، در قلب آن، SDL وظایفی را تعریف می‌کند و این وظایف می‌تواند توسط یک فرآیند توسعه چابک ترسیم گردد

اگر کاربران روش‌های چابک بخواهند از SDL استفاده کنند باید دو تغییر ایجاد کنند. اول اینکه فزودن SDL به فرآیندهای چابک باید اندک باشد. یعنی تیم توسعه برای هر ویژگی نرم افزار ، فعالیت‌های SDL را به‌اندازه کافی انجام دهد قبل از اینکه شروع به کار روی ویژگی دیگر کند. دوم اینکه مراحل توسعه‌ی (طراحی، اجرا، وارسی و انتشار) مرتبط با SDL در یک قالب سریع‌تر سازماندهی شوند زیرا روش‌های چابک را به کار نگرفته‌اند. برای این منظور تیم SDL مایکروسافت یک روش موثر سریع و امن را توسعه داده و به کار گرفته است که چرخه توسعه امنیت برای توسعه چابک یا SDL-Agile نام دارد.

در این مقاله ترکیب متدولوژی SDL مایکروسافت که یک متدولوژی کامل برای توسعه نرم‌افزار امن است با متدولوژی‌های چابک توسعه نرم‌افزار به‌گونه‌ای که اصول هر دو (فرآیند SDL و روش‌های چابک) برقرار باشد مورد بررسی قرار گرفت. این راهکار برای تیم‌های توسعه نرم‌افزار که می‌خواهند برنامه‌های‌کاربردی امن‌تری با استفاده از روش‌های چابک بسازند مناسب خواهد بود.

مراجع

[۱] ژولیا، آ.. سین، ب.. روبرت ج، ا.. گری، م، ۱۳۹۴، مهندسی امنیت نرم‌افزار—راهنمای مدیران پروژه، مترجم: سپیدنام، قدرتالله، جلد اول، انتشارات علوم رایانه.

[۲] ترکمانی، م، ۱۳۹۶، مهندسی نرم‌افزار امن با رویکرد علمی و کاربردی، جلد اول، انتشارات ارسطو، ویراست دوم.

[۳] ترکمانی، م، ۱۳۹۵ مهندسی نرم‌افزار امن، جلد اول، انتشارات ارسطو، ویراست سوم.

[۴] ترکمانی، محمدعلی، نجاتیان، صمد، ۱۳۹۶، "مدلسازی نرم افزار امن با استفاده از UMLSec"، کنفرانس ملی مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه آزاد اسلامی واحد سپیدان.

[۵] ترکمانی، محمدعلی، نجاتیان، ۱۳۹۶، "رهیافت STRIDE برای مدل‌سازی تهدیدها در مهندسی نرم افزار امن"، کنفرانس ملی مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه آزاد اسلامی واحد سپیدان.

[۶] ترکمانی، محمدعلی، نیکنام، مجتبی، ۱۳۹۶، "اصول راهنمای مایکروسافت برای امنیت و حریم خصوصی نرم افزار"، کنفرانس ملی مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه آزاد اسلامی واحد سپیدان.

[7] Microsoft SDL, 2012, Version 5.2.

[8] Microsoft Security Lifecycle (SDL),2017, http://www.microsoft.com/security/sdl/default.aspx.

[9] Allen, J., Barnum, S., Ellison, R and at. Al, 2008, “Software security engineering”, Addison Wesley Profession.

[10] Howard, M and Liper, S, 2006, the Security Development Lifecycle: SDL.

[11] Jürjens, J, 2002, *UML Sec: Extending UML for secure systems development*, ACM.

[12] Jürjens, J, 2004, secure systems development with UML, Springer.

[13] https://www.microsoft.com/en-us/SDL/discover/sdlagile-bucket.aspx.

[14] https://www.microsoft.com/en-us/SDL/discover/sdlagile-onetime.aspx.

[15] https://www.microsoft.com/en-us/download/details.aspx?id=19968.

[16] https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx.

[17] https://www.microsoft.com/en-us/download/details.aspx?id=49168.