فصلنامه علمی-ترویجی پدافند غیرعامل

سال پنجم، شماره ۱، بهار ۱۳۹۳، (پیاپی ۱۷): صص ۴۱-۵۰

امن‌سازی نرم‌افزار مبتنی بر کدنویسی امن

سلیمان جرجم[۱]، مهدی دهقانی[۲]

تاریخ دریافت: ۹۲/۰۴/۰۵
تاریخ پذیرش: ۹۲/۱۰/۱۶

چکیده

امنیت نرم‌افزار به روش‌های تولید نرم‌افزارهای امن می‌پردازد. مفاهیمی‌که برای امنیت نرم‌افزار اهمیت دارد شامل مدیریت مخاطرات سامانه، زبان و بستر برنامه‌نویسی، ممیزی نرم‌افزار، طراحی برای امنیت و آزمون‌های امنیتی است. به بیان دیگر، رعایت تمهیدات امنیتی در فرآیند تولید نرم‌افزار را می‌توان پدافند غیرعامل در حوزه نرم‌افزار دانست. در اغلب موارد، اشتباهات برنامه‌نویسی که به سادگی قابل اجتناب هستند، منجر به بروز آسیب‌پذیری‌های قابل سوءاستفاده در نرم‌افزارها می‌شوند. بررسی‌ها و تحلیل‌های صورت گرفته که بر روی هزاران آسیب‌پذیری گزارش شده انجام شده است، نشان می‌دهد که اکثر آسیب‌پذیری‌ها از تعداد کمی‌خطاهای برنامه‌نویسی عمومی ناشی می‌شوند. ما می‌دانیم که هرچه زودتر یک آسیب‌پذیری کشف شود، برطرف نمودن آن آسان‌تر و ارزان‌تر می‌باشد. بکارگیری چرخه حیات توسعه نرم‌افزار امن که امنیت را در هر مرحله از توسعه برای شناسایی اولیه آسیب‌پذیری‌های بالقوه رسیدگی می‌نماید، در امن‌سازی نرم‌افزار اهمیت زیادی دارد. بنابراین توسعه‌دهندگان بایستی درک صحیحی از روش‌های کدنویسی امن برای کاهش خطاهای امنیتی و توسعه مناسب برنامه‌های کاربردی امن داشته باشند. این مقاله رهنمون‌هایی برای کدنویسی امن پیشنهاد می‌دهد.

کلیدواژه‌ها: کدنویسی، امنیت، زبان‌های برنامه‌نویسی،پدافند غیرعامل، سرریز بافر، آسیب‌پذیری

۱- دانشجوی کارشناسی ارشد دانشگاه جامع امام حسین(ع) sjorjam@ihu.ac.ir - نویسنده مسئول

۲- مربی و عضو هیئت علمی دانشگاه جامع امام حسین(ع) mdehghany@ihu.ac.ir

۱- مقدمه

هزینه برای سازمانی که نرم‌افزاری حاوی آسیب‌پذیری امنیتی منتشر می‌سازد، بسیار زیاد است. لذا با توجه به اینکه یکی از اهداف مهم پدافند غیرعامل تقلیل آسیب پذیری‌ها و کاهش خسارات و صدمات تأسیسات، تجهیزات و نیروی انسانی مراکز حیاتی، حساس و مهم نظامی و غیرنظامی کشور در برابر تهدیدات و حملات دشمن، و نیز بلایای غیر طبیعی می‌باشد[۱]. اجتناب از این قبیل آسیب‌پذیری‌ها نیازمند آگاهی رسانی و تعهد در بخشی از سازمان است که برنامه‌های کاربردی نرم‌افزاری را توسعه می‌دهد.

اغلب کلاس‌های برنامه‌نویسی ابتدا به سبک کدنویسی خوب[1] تاکید می‌کنند، که در واقع همان کدنویسی امن می باشند[۲]. نوشتن کد امن از جمله بخش‌های ضروری توسعه نرم‌افزاری امن می‌باشد. بررسی‌ها و تحلیل‌های صورت گرفته که بر روی هزاران آسیب‌پذیری گزارش‌شده انجام شده است، نشان می‌دهد که اکثر آسیب‌پذیری‌ها از تعداد کمی خطاهای برنامه‌نویسی مشترک ناشی می‌شوند[۳].

برای ایجاد زمینه ورود به بحث، از تعاریف زیر استفاده می‌گردد:

- رخنه امنیتی، نقصی در برنامه کاربردی یا اجزاء نرم‌افزار می‌باشد که وقتی با شرایط لازم ترکیب می‌گردد، می‌تواند موجب آسیب‌پذیری نرم‌افزاری شود[۴].
- آسیب‌پذیری، هر ویژگی از یک سیستم کامپیوتری است که امکان نقض خط مشی‌های امنیتی آن سیستم را به فرد یا افرادی بدهد[۵].
- بهره‌برداری، بخشی از نرم‌افزار یا روشی است که از آسیب‌پذیری‌های امنیتی برای تجاوز به سیاست امنیتی آشکار به‌کار گرفته می‌شود[۴].

۲-تأثیر آسیب‌پذیری‌ها و هزینه‌های مرتبط

برابر گزارش‌های موسسه ملی استاندارد و فناوری[2]، تا کنون بیش از ۱۵۰۰۰ آسیب‌پذیری گزارش‌شده و به بانک اطلاعاتی آسیب‌پذیری‌ها و افشاء شده‌های مشترک[3] اضافه شده اند. رشد تعداد آسیب‌پذیری‌های گزارش شده موجب افزایش نگرانی‌ها شده است. شیوه توسعه ضعیف، سبب بروز عیوب و ایجاد آسیب‌پذیری‌های بسیار مهم می‌گردد. اغلب اوقات، برنامه نویس نمی‌تواند این آسیب‌پذیری‌ها را در مراحل آغازین چرخه توسعه شناسایی نماید. زمانی که محصول روانه بازار گردید، برخی آسیب‌پذیری‌ها احصاء و توسط محققین گزارش می‌شود. آسیب‌پذیری‌هایی که توسط نفوذگران یا مهاجمین کشف و مورد بهره‌برداری قرار می‌گیرند معمولا تا زمانی که خسارت قابل توجهی به بار نیاورند، مورد توجه قرار نمی‌گیرند. و حالا زمانی برای توسعه و ارائه یک وصله[4] جهت رفع یک آسیب‌پذیری است. زمانی که یک وصله در دسترس قرار می‌گیرد، بایستی با سامانه‌ها یا برنامه‌های کاربردی جاری برای برطرف نمودن مشکل امنیتی پیکربندی گردد. در هر حال، این نوع از سامانه مدیریت وصله برای چندین مورد آسیب‌پذیری خیلی کاربردی نیست. نخست اینکه، فرآیند طولانی جهت توسعه و انتشار وصله لازم می‌باشد و دوم اینکه، احتمال دارد که وصله با سامانه یا برنامه کاربردی موجود منطبق نباشد و از این رو، نمی‌توان بدون آزمایش در یک محیط ویژه آن را مستقر نمود. شکست در پیکربندی صحیح وصله امنیتی می‌تواند موجب بروز یک موضوع بحرانی امنیتی گردد.

عموماً، تأثیر آسیب‌پذیری و مدیریت وصله شامل هزینه‌های زیر می‌باشد:

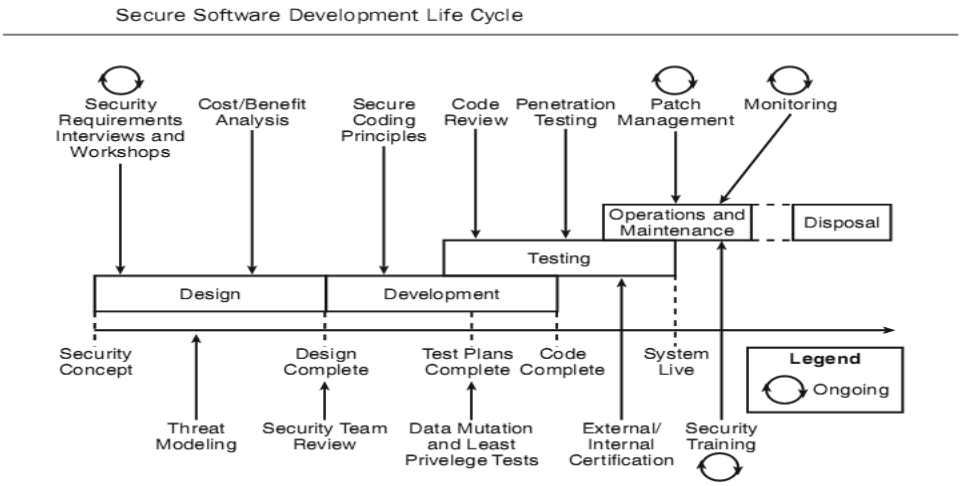
- ارزیابی آسیب‌پذیری
- تولید و توسعه وصله
- بازیابی، نصب و آزمایش
- اعلام وصله
- پشتیبانی از موضوعات مدیریت وصله
- زمان توقف کاربرد وصله

در واقع، زمانی که کلیه عوامل بررسی گردید، هزینه رفع یک موضوع امنیتی با توسعه و کاربردی نمودن یک وصله تقریباً ۶۰ برابر هزینه رفع اشکال[5] امنیتی در مراحل آغازین SDLC می‌باشد [۶].

شکل (۱) چرخه حیات توسعه نرم‌افزار امن را نشان می‌دهد[۷].

هزینه مازاد ممکن برای شرکتی که نرم‌افزاری حاوی آسیب‌پذیری امنیتی را ارائه می‌دهد، وجود دارد. چنین الگوی

1- Good coding style
2- National Institute of Standard and Technology
3- Common Vulnerabilities and Exposures
4- Patch
5- Bug



شکل ۱- چرخه حیات توسعه نرم‌افزار امن [۷]

فعالیتی می‌تواند موجب خسارت به شهرت شرکت، ضرر به قیمت سهام و دیگر واحدهای ارزشی آن گردد.

۳-امن‌سازی نرم‌افزارها

نرم‌افزار اکنون یکی از اجزای اصلی تجارت قلمداد شده و بسیاری از جنبه‌های زندگی روزمره را فراگرفته است. علیرغم رشد روزافزون نرم‌افزار و سرعت چشمگیر تولید نرم‌افزارهای کاربردی، هم‌چنان تولید نرم‌افزارهایی با قابلیت اعتماد بالا و امنیت قابل قبول از مشکلات اصلی در صنعت تولید نرم‌افزار به حساب می‌آید.

راه‌حل امن‌سازی نرم‌افزارها در دو ایده کلی نهفته است:

- در نظر داشتن امنیت در فرایند تولید نرم‌افزار (مهندسی نرم‌افزار امن)
- نفوذ و عیب‌یابی امنیتی و رفع نقایص با وصله‌های امنیتی پس از تولید

طبق گزارش گروه پاسخگویی رویدادهای رایانه‌ای[1]، ۱۰ نقطه ضعف شناخته‌شده معروف، دلیل اصلی ۷۵٪ مشکلات امنیتی گزارش‌شده در نرم‌افزارها است. بنابراین می‌توان گفت که اگر تولید کنندگان نرم‌افزار با این ده مشکل امنیتی آشنایی داشته باشند و در تولید نرم‌افزار آن‌ها را رعایت کنند، حدوداً ٪۷۵ مشکلات امنیتی نرم‌افزارها کاهش می‌یابد [۸].

امنیت نرم‌افزار در مورد روش تولید نرم‌افزارهای امن است. مفاهیمی‌که برای امنیت نرم‌افزار اهمیت دارد شامل مدیریت مخاطرات سامانه، زبان و بستر برنامه‌نویسی، ممیزی نرم افزار، طراحی برای امنیت و آزمون‌های امنیتی است. امنیت نرم افزار بیشتر مربوط

به طراحی امن نرم‌افزار و آموزش به تحلیل‌گران، معماران و پیاده‌سازان برای امنیت برنامه است.

اگرچه موضوع حفاظت از نرم‌افزار بعد از تولید آن در جای خود بسیار مهم است ولی باید توجه داشت که حفاظت از نرم‌افزاری که به دلیل نقص در تولید در مقابل حملات، آسیب‌پذیر است بسیار سخت‌تر از نرم‌افزاری است که در دل خود سازوکارهای حفاظتی برای مقابله با تهدیدهای موجود را دارد. مثال زیر این موضوع را روشن‌تر می‌سازد:

"با این‌که می‌توان از حملات سرریز بافر با استفاده از مشاهده ترافیک HTTP که از پورت ۸۰ می‌رسد، جلوگیری نمود (بعد از تولید نرم‌افزار) ولی راه‌حل بهتر، کدنویسی صحیح برای مقابله اصولی و بنیادی با حملات سرریز بافر در مرحله تولید نرم افزار است."

۴- آسیب‌پذیری‌های شناخته‌شده ناشی از کدنویسی

در امنیت نرم‌افزار، واژه "آسیب‌پذیری" به ضعف یا نقص در کلیه مراحل تولید نرم‌افزار اشاره دارد که بلافاصله پس از بهره‌برداری، می‌توان سیاست امنیتی سازمان را نقض نمود و به مهاجم اجازه دسترسی غیرمجاز داد. در ادامه نگاهی اجمالی به برخی آسیب‌پذیری‌های معمول می‌اندازیم.

1- Computer Emergency Response Team

```
/** Example of Buffer Overflow **/

Int main(int argc, char const *argv[])
{
Char buffer1[5]="VXYZ";
Char buffer2[5]="PQRS";
Strcpy(buffer2, argv[1]);
Printf("buffer1: %s, buffer2: %s\n", buffer1,
buffer2);}
```

شکل ۲- آسیب‌پذیری سرریز بافر

۴-۱- سرریز بافر

سرریز بافر زمانی رخ می‌دهد که ورودی یک برنامه اجازه می‌یابد تا داده‌ای را در حافظه اختصاص‌یافته[1] بنویسد. مهاجم می‌تواند کنترل کامل برنامه کاربردی را به‌دست‌آورد یا اینکه برنامه را از طریق بهره‌برداری از سرریز بافر در هم شکند[2]. زبان‌های C و C++ معمولا تحت تأثیر قرار می‌گیرند. برخی زبان‌ها، مانند جاوا، C# و ویژوال بیسیک دارای ساز و کار بررسی محدوده یک آرایه و انواع رشته‌ای بومی می‌باشند، که معمولا از دسترسی مستقیم به حافظه ممانعت می‌کنند. از این‌رو، این زبان‌ها کمتر در معرض سرریزهای بافر قرار می‌گیرند.

در شکل(۲)، آرگومان بدون بررسی اندازه‌اش در buffer۲ کپی می‌شود. این ریزش منجر به آسیب‌پذیری سرریز بافر می‌گردد.

۴-۲- سرریز عدد صحیح[3]

سرریز عدد صحیح، زمانی رخ می‌دهد که متغیر صحیح تلاش می‌کند تا مقداری بزرگتر از دامنه معتبر را به‌عنوان نتیجه یک عملیات ریاضی ذخیره نماید [۹]. C و C++ زبان‌های ناامنی هستند و مستعد تبدیل یک سرریز عدد صحیح به یک سرریز بافر می‌باشند. برخی زبان‌ها، از قبیل جاوا و Ada، یک بررسی کننده نوع عدد صحیح را پیاده‌سازی می‌کنند، که تا حد زیادی منجر به کاهش خطر می‌گردد. در شکل(۳)، ممکن است متغیر عدد صحیح "number" پیوسته مقادیر کوچک‌تر از MAX_NUM ایجاد نماید و در نتیجه، باعث سرریز عدد صحیح شود. این سناریو همچنین به تعداد MAX_Num-۱ بایت از متغیر buffer را بازنویسی خواهد کرد.

1- Allocated memory
2- Crash a program
3- Integer
4- Format String Attacks

```
/** Example of Integer Overflow **/

short int number =0;
char buffer[large_value];

while (number <MAX_NUM)
{
umber +=getInput(buffer+number);
}
```

شکل ۳- آسیب‌پذیری سرریز عدد صحیح

۴-۳- حملات رشته قالب[4]

این حمله زمانی رخ می‌دهد که داده تامین‌شده توسط کاربر از رشته ورودی، توسط برنامه کاربردی به عنوان یک دستور ارزیابی گردد؛ به روشی که یک مهاجم می‌تواند داده را از stack فراخوانی کند، کد دلخواه را اجرا کند یا اطلاعات را افشاء نماید. ورودی نامناسب ممکن است حاوی پارامترهای رشته‌ای قالب نظیر %x یا %n باشد؛ اغلب، زبان‌های آسیب‌پذیر C و C++ می باشند. اگر آرگومان ورودی حاوی یک پارامتر رشته‌ای مانند %x یا %n باشد، برنامه خروجی غیرمنتظره‌ای را چاپ خواهد کرد. در شکل(۴) استفاده از printf(argv[۱]) به جای printf("%s",argv[۱]) موجب بروز یک آسیب‌پذیری رشته قالب جدی می‌گردد.

```
/** Example of Format String Attack **/
Int main(int args, char *argv[])
{
Printf(argv[1]);Return 0;}
```

شکل ۴- آسیب‌پذیری حملات رشته قالب

۴-۴- تزریق دستور[5]

تزریق دستور زمانی رخ می‌دهد که داده نامناسب درون ورودی جاسازی شده و به پوسته[6] (یا یک مفسر زبان) انتقال یابد. در نتیجه، ورودی به عنوان گونه‌ای از انواع دستورات تفسیر می‌گردد. برنامه‌هایی که در زبان‌هایی نوشته می‌شوند که نمی توانند اعتبارسنجی ورودی مناسبی انجام دهند ممکن است در برابر این نوع حملات آسیب‌پذیر باشند.

برنامه دستور یونیکس "head filename' را اجرا می‌کند و ۱۰ خط اول فایلی را که به عنوان ورودی داده شده است نمایش می‌دهد. "strcat' دستور را با نام فایل الحاق می‌کند و دستور "system' آن را اجرا می‌نماید. از آنجا که اعتبارسنج ورودی وجود

5- Command Injection
6- Shell

ندارد، در شکل(۵) مهاجم می‌تواند دستور نامناسب را در ورودی تزریق نماید، مانند "hello.c; rm welcome.c".

اجرای این برنامه باعث نمایش بخش اول hello.c و حذف فایل welcome.c می‌گردد.

```
/** Example of Command Injection **/
Int main(char* args, char** argv)
{
Char command[MAX]="head";
strcat(command, argv[1]);
system(command);}
```

شکل ۵- آسیب‌پذیری تزریق دستور

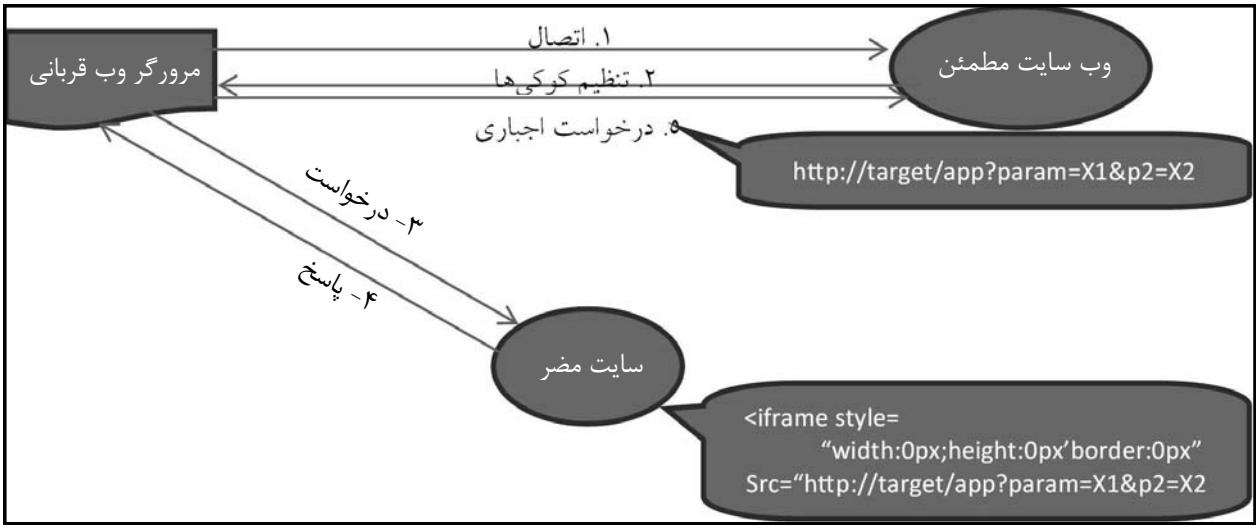
۴-۵- Cross-site scripting

یک آسیب‌پذیری cross-site scripting (XSS) معمولا در برنامه‌های کاربردی تحت وب پیدا می‌شود که مهاجم داده نامناسب را به طریقی وارد می‌نماید که سازوکارهای کنترل دسترسی را کنار خواهد گذاشت. در نتیجه، داده نامناسب به مرورگر وب سرویس گیرنده منعکس می‌گردد (XSS ناپایدار) یا در سمت سرویس‌دهنده ذخیره می‌گردد (XSS پایدار)، و می‌تواند موجب سرقت اطلاعات حساس گردد، از قبیل کوکی‌ها (XSS پایه-DOM). با بهره‌برداری از این آسیب‌پذیری، مهاجم می‌تواند وب‌سایت‌ها را بدشکل[1] نماید، حملات فیشینگ را اجرا کند، پیوندهای نامناسب را در صفحات وب مطمئن تزریق نماید یا اینکه اطلاعات محرمانه را به وب‌سایت‌های نامطمئن دیگر ارسال نماید. اگر اعتبارسنجی ورودی سمت-سرور به درستی پیاده‌سازی نشده باشد، پس هر زبانی که برای ساختن برنامه‌های کاربردی تحت وب استفاده شود (برای نمونه، PHP، C#، VB.NET، ASP.NET، J2EE و صفحات سرور فعال(ASP)) در مقابل این نوع حملات آسیب‌پذیر است.

۴-۶- جعل درخواست cross-site

جعل درخواست cross-site(CSRF[2]) زمانی رخ می‌دهد که کاربری مجبور به اجرای اقدامات ناخواسته در یک برنامه کاربردی تحت وبی گردد که مورد تایید آن قرار گرفته است. کلیه برنامه‌های تحت وبی که فقط از تایید اصالت ثابت (کوکی[3] یا شناسه جلسه[4]) استفاده می‌کنند ممکن است از این قبیل حملات آسیب پذیرند. به‌وسیله مدیریت و کنترل نمودن این حمله، مهاجم می‌تواند کلیه انواع اقدامات که به سودش می‌باشد را اجرا نماید؛ برای نمونه سرمایه‌های موجود در حساب قربانی را به حساب مهاجم منتقل نماید.

شکل(۶) نمونه‌ای از CSRF را تشریح می‌نماید. در این سناریو، ابتدا قربانی گواهی‌نامه اتصال[5] را به وب‌سایت مطمئن می‌فرستد. به‌محض تأیید هویت موفق مرورگر، وب قربانی کوکی‌های ارسال‌شده وب‌سایت مطمئن را ذخیره می‌کند. حال، اگر قربانی از سایت مضر بازدید نماید، سپس سایت حمله کننده می‌تواند یک حمله CSRF را بر علیه قربانی اقدام نماید. اساسا، سایت مضر از



شکل ۶- تشریح جعل درخواست cross-site

1- deface
2- Cross-site request forgery
3- Cookie
4- Session
5- login credential

گواهی‌نامه‌های ذخیره‌شده قربانی در قالب کوکی‌ها استفاده می‌کند و درخواست‌های HTTP را به وب‌سایت مطمئن ارسال می کند. وب‌سایت مطمئن نیز عمل درخواست‌شده را اجرا می‌کند، که این درخواست بر اساس کوکی‌های ذخیره‌شده، تایید اصالت گردیده است.

حملات CSRF بر پایه این فرض می‌باشند که وب سایت مطمئن، مرورگر کاربر را تایید اعتبار می‌نماید.

۴-۷-تزریق‌های SQL

تزریق‌های SQL دستورات یا پرس‌وجوهای SQLای هستند که درون داده ورودی کاربر به نحوی جاسازی شده‌اند که سازوکارهای کنترل دسترسی را کنار خواهد گذاشت و سپس می‌توانند نمایش دهند، اضافه نمایند، حذف کنند یا داده‌ها را در بانک اطلاعات تغییر دهند. به‌محض بهره‌برداری، مهاجم می‌تواند هر داده درون بانک اطلاعاتی شامل مشخصات فردی، از قبیل شماره کارت‌های اعتباری، اسامی، آدرس‌ها و دیگر اطلاعات حساس را بدزدد. هر زبان برنامه‌نویسی که به بانک اطلاعاتی متصل می‌گردد تحت تأثیر است؛ برای نمونه، C#، PHP، ASP.NET وJSP. در شکل(۷)، مفسر SQL فقط نام کاربری را بررسی خواهد کرد، به عنوان نیمه دوم کوئری(1=1 or) همیشه صحیح می‌باشد.

```
/** Example of  SQL Injection **/

<from method="post" action="Login_Account.php">
  <input type="text" name="username">
  <input type="password" name="password">
</form>
```

در صورت فقدان ارزیابی صحیح ورودی، مهاجم می‌تواند برای بهره‌برداری از این نوع آسیب‌پذیری، ورودی نامناسب زیر را تولید نماید:

```
Select * from LOGIN where username='john_smith' and
  password='' or 1=1;
```

شکل ۷- آسیب‌پذیری تزریق SQL

۴-۸- ارجاع نامطمئن شئ مستقیم[1]

ارجاع نامطمئن شئ مستقیم، زمانی رخ می‌دهد که برنامه‌نویس از اعطای مجوز استفاده نکرده باشد و ارجاعی آشکارا به شئ پیاده‌سازی‌شده داخلی از قبیل URL، مقدار قالب[2]، فایل، شاخه[3] یا رکورد بانک اطلاعاتی داشته باشد. یک مهاجم می‌تواند با این روش ارجاعات را برای کسب دسترسی به اطلاعات حساس دستکاری کند. کلیه زبان‌هایی که از توسعه برنامه‌های کاربردی استفاده می‌کنند در مقابل این نوع حمله آسیب‌پذیر می باشند. مهاجم می‌تواند از "/.." برای پرش از موقعیت جاری به شاخه هدف برای دسترسی به برخی فایل‌ها یا منبع حساس که او مجوز دسترسی ندارد استفاده نماید.

۴-۹- کنترل خطای نامناسب و نشت اطلاعات[4]

زمانی که یک برنامه‌نویس به درستی کنترل خطا را پیاده‌سازی نکند، برنامه کاربردی ممکن است درباره وضعیت داخلی، پیکربندی سامانه یا منابع (برای نمونه، سرویس‌دهنده وب Apache نسخه ۲.۲.۸) اطلاعاتی را منتشر نماید. این اطلاعات می‌تواند برای حمله به سامانه استفاده گردد. کنترل خطای نامناسب موجب از کار افتادن، خاتمه غیر عادی[5]، پایان دادن یا شروع مجدد برنامه می‌گردد و همگی مستعد برای ممانعت از سرویس‌دهی می‌گردند. اگر کنترل خطای نامناسب اتفاق بیفتد، زبان‌های برنامه‌نویسی از قبیل C#، VB.NET و جاوا، که از استثنائات استفاده می کنند؛ زبان‌هایی نظیر C، C++، ASP و PHP، که از مقادیر برگشتی تابع خطا استفاده می‌کنند؛ و نیز سایر زبان‌های برنامه‌نویسی در برابر نشت اطلاعات آسیب‌پذیر هستند. پیغام خطای نامناسب ممکن است اطلاعات مهمی درباره وب سرور، سیستم عامل، شماره پورت و دیگر محصولات مورد استفاده را فاش کند.

۴-۱۰- ذخیره ناامن و استفاده نامناسب از رمزنگاری

ذخیره کلمات عبور، کلیدها، گواهی‌نامه‌ها و دیگر اطلاعات محرمانه به شکل متن ساده ممکن است موجب ذخیره ناامن گردد. استفاده از کلیدهای کوتاه (مانند DES از کلیدهای ۵۶بیتی استفاده می‌کند)، الگوریتم ضعیف (برای مثال، RC۴ و MD۵)، الگوریتم‌های غیراستاندارد، اعداد تصادفی رمزنگاری-ضعیف در یک مولد عدد شبه تصادفی(PRNG)[6]، مدیریت کلید نامناسب (برای مثال، کلیدهای ذخیره‌شده در مکان ناامن) و کدنویسی[7] سخت کلیدها ممکن است موجب نفوذ در اطلاعات حساس گردد. برخی پروتکل‌های معمول مانند HTTP شامل مذاکره الگوریتم‌های رمزنگاری می‌باشد و ممکن است اجازه استفاده از الگوریتم‌های

1- Insecure Direct Object Reference
2- Form Parameter
3- Directory
4- Improper Error Handling And Information Leakage
5- Abort
6- Pseudo Random Number Generator
7- Hard Coding

ناامن را فراهم نماید.

۴-۱۱- زمان بررسی در مقابل زمان استفاده (وضعیت مسابقه[1])

هنگامی که یک برنامه کاربردی در مراحل جدا اجرا می‌شود، این امکان وجود دارد که اجرای برنامه کاربردی بین مراحل مختلف قطع شود و یک قطعه کد دیگر اجرا شود. برای مثال، موردی را که در آن یک برنامه کاربردی، داده را در یک فایل می‌نویسد در نظر بگیرید. فرض کنید این برنامه دو مرحله داشته باشد. مرحله اول، چک کردن دسترسی فایل و مرحله دوم، نوشتن مقادیر بر روی این فایل است. در این حالت امکان دارد که هنگامی که چک کردن دسترسی فایل کامل شد و قبل از مرحله نوشتن، مقدار این فایل را جایگزین فایل دیگری کنیم، اگر نفوذگر بتواند از این آسیب‌پذیری بهره‌برداری کند، امنیت به راحتی نقض خواهد شد.

۵- توسعه نرم‌افزار امن

"توسعه نرم‌افزار امن" فرآیند توسعه‌ای از طراحی، ساخت، و آزمایش نرم‌افزار است که امنیت را در هر مرحله دخیل می‌کند و هدفش این است که مخاطرات امنیتی معرفی‌شده در هر مرحله از چرخه حیات توسعه نرم‌افزار امن (SSDLC) را کاهش دهد. بخش‌های زیر نگاهی به مراحل SSDLC خواهد انداخت [۱۰ و۱۱].

۵-۱-معماری و طراحی امن

هدف اصلی مرحله "معماری و طراحی امنیت" کاهش تعداد آسیب‌پذیری‌ها قبل از آغاز توسعه است. این یک فرآیند مرور تکراری برای تحلیل معماری و طراحی برنامه کاربردی از جنبه امنیتی است. مرور شامل تجزیه‌وتحلیل نواحی بحرانی نرم‌افزار برای یکپارچه‌سازی مفاهیم امنیت پایه است؛ به طور مثال، تصدیق هویت، اعطای مجوز، محرمانگی و جامعیت.

نکته: حذف رخنه‌های امنیتی در سطح طراحی، آسان تر و مقرون به صرفه‌تر از حذف آن در دیگر مراحل SSDLC می‌باشد.

۵-۱-۱-مدل‌سازی تهدیدات

مدل‌سازی تهدیدات به دنبال توصیف و توسعه تهدیدات وارده است. اولین گام شناسایی تهدیداتی است که ممکن است به برنامه کاربردی یا محصول آسیب بزنند، و سپس شناسایی آسیب‌پذیری‌های وابسته به تهدیدات می‌باشد. گام بعدی شامل تجزیه و تحلیل حملات ممکن و احتمالی است که بر اثر تهدیدات و اقدامات کنترل برنامه‌ریزی‌شده برای کاهش احتمال این قبیل حملات انجام می‌گیرد.

۵-۱-۲-ملاحظه طراحی امن

این مرحله، راه‌حل‌ها را برای تهدیدات بالقوه ممکن در طراحی خود یکپارچه می‌نماید. مفهوم مبنایی در پشت هر آسیب‌پذیری و حمله برای ایجاد یک طراحی امن عمومی، بایستی به درستی فهمیده شود. سپس،

طراحی می‌تواند کلیه پیش‌نیازهای امنیتی برنامه‌کاربردی را در ذهن بپروراند.

همچنین توسعه‌دهندگان می‌توانند از "الگوهای طراحی امن [۱۲]" در برخورد با موضوعات امنیتی مرتبط و حل مشکلات امنیتی شناخته‌شده استفاده نمایند.

۵-۲- روش‌های کدنویسی امن

هدف عمده مرحله "اقدامات کدنویسی امن" گسترش آگاه‌سازی درباره امنیت نرم‌افزار در جامعه توسعه‌دهندگان نرم‌افزار است. برنامه‌ای که با استانداردهای کدنویسی امن نوشته می‌شود، به‌مراتب امن‌تر از برنامه‌ای است که این استانداردها را رعایت نکرده است. به نظر می‌رسد کار اضافه‌ای انجام می‌شود، اما استانداردهای کدنویسی امن نه فقط احتمال مواجهه با آسیب‌پذیری را کاهش می‌دهد، بلکه دخول غیرمترقبه دیگر رخنه‌ها[2] را نیز کاهش می‌دهد.

لطفا برای تشریح مطالب روش‌های کدنویسی امن، به بخش "۶-رهنمون‌های کدنویسی امن" مراجعه نمایید.

۵-۳- آزمایش امنیت نرم‌افزار

آزمایش امنیت نرم‌افزار، مرحله مستقلی از SSDLC برای اعمال بررسی صحت قبل از ارائه کد می‌باشد. این آزمایش، فرایندی از کشف رخنه‌های امنیتی در کد داده شده است. در ادامه، رویکردهایی برای هدایت آزمایش امنیتی ارائه می‌گردد.

۵-۳-۱- مرور و بررسی کد

مرور کد، فرایندی از آزمایش امنیت نرم‌افزار می‌باشد که توسعه‌دهنده برنامه، بررسی‌کننده‌های دو نفره متقابل و یک گروه آزمایش کننده تضمین کیفیت بررسی کد را با هم انجام می‌دهند.

1- Race Condition

2- Flaws

بررسی کد ممکن است دستی یا خودکار باشد. در حالت بررسی دونفره متقابل، توسعه‌دهندگان نرم‌افزار می‌توانند کدهای منبع یکدیگر را برای یافتن اشکالات امنیتی قبل از ارائه کد به گروه تضمین کیفیت بررسی نمایند.

در ادامه، تعدادی از رهنمون‌های بررسی کد ارائه می‌گردد:

- آسیب‌پذیری‌های معمول مطرح شده در بخش ۴ را دنبال کنید.
- فراخوانی‌های سیستم را شناسایی نمائید. به طور مثال توابع System(), popen(), execclp(), execcpv() در زبان C/C++ به عنوان رابط سیستم عامل می‌باشند؛ و یا در SQL تابع xp_cmdshell اجازه اجرای هر دستوری از سیستم‌عامل را می‌دهد.
- از ابزارهای بررسی کد منبع برای امتحان آن و یافتن ریزش‌های امنیتی استفاده نمائید. برای مثال، ابزار تجزیه‌وتحلیل PREfast از تجزیه‌وتحلیل ایستا برای یافتن نواقص در C یا C++ استفاده می‌کند، و Flawfinder از بانک اطلاعات درونی حاوی توابع C یا C++ با مشکلات معروف و شناخته‌شده استفاده می‌کند.

۵-۳-۲-آزمون نفوذپذیری

یک آزمون‌گر نفوذ، ارزیابی معیارهای امنیت برنامه کاربردی را به شیوه آزمایش جعبه سیاه انجام می‌دهد. او هیچ دانشی از معماری یا کد منبع برنامه کاربردی ندارد. نتایج این فرآیندهای ارزیابی بایستی مستند گردد و به درخواست‌کننده آزمون ارائه گردد.

۵-۳-۳-آزمون مرحله[1]

آزمون مرحله به معنی آزمون برنامه کاربردی در مقابل داده ناهنجار برای دیدن اینکه چگونه واکنش نشان می‌دهد می‌باشد. اگر برنامه کاربردی از کار افتاد، بنابراین بایستی اشکال جدیدی گزارش گردد و در مدل تهدید به‌روز گردد. هر زمان که برنامه کاربردی تغییر یابد یا به‌روز شود، آزمون مرحله باید اجرا گردد.

روش‌های اجرای مرحله، از آزمون جعبه سیاه استفاده می‌کنند. این روش اجازه شناسایی اغلب آسیب‌پذیری‌های معمول را می‌دهد. برای مثال، سرریز بافر، cross-site scripting و تزریقات SQL.

1- Fuzz

۶-رهنمون‌های کدنویسی امن

بسیاری از رهنمون‌های کدنویسی امن توصیه‌هایی برای جلوگیری از نوشتن برخی الگوهای کدنویسی که ممکن است منجر به آسیب‌پذیری‌هایی شوند، ارائه می دهند [۱۳]. روش‌های کدنویسی امن شامل رهنمون‌های عمومی است که می‌تواند برای ارتقاء امنیت نرم‌افزار بدون توجه به زبان برنامه‌نویسی مورد استفاده قرار گیرد:

- ارزیابی ورودی مؤثر و کارا الزامی است. همانگونه که در بخش۴ ذکر شد، اغلب آسیب پذیری‌ها نتیجه فقدان یا ارزیابی ورودی ناقص می‌باشد. یک شیوه خوب پیشنهاد می‌کند هر ورودی کاربر مورد بررسی دقیق قرار گیرد و تصفیه شود. بهتر این است که یک فهرست سفید از پارامترها و قالب‌های ورودی شناخته‌شده مورد انتظار، به جای تکیه بر یک فهرست سیاه از ورودی‌های بد شناخته‌شده ایجاد نمایید.
- انجام طراحی نرم‌افزار و دیگر مراحل مهندسی نرم‌افزار، تسهیل کننده کد ساخت‌یافته، کوچک و ساده می‌باشد. به‌علاوه، از یک فهرست بررسی[2] کدنویسی امن استفاده نمایید.
- برای پیگیری تغییرات ایجادشده در کد و سند، از کنترل نسخه یا پیکربندی استفاده نمایید؛ این اقدام امکان عقب‌گرد[3] ساده به نسخه قبلی را در حالت اشتباه جدی به‌وجود می‌اورد. کنترل نسخه، جوابگویی و کاهش زمان توسعه را تسهیل می‌کند.
- هرگز به ورودی دستورات SQL اعتماد نکنید. از دستورات SQL که به درستی تعریف شده است، استفاده نمایید. از اتصال رشته و جایگزینی رشته برای ایجاد دستورات SQL استفاده نکنید.
- از کتابخانه‌ها (مانند، کتابخانه ضد cross-site scripting) برای محافظت در برابر اشکالات امنیتی در حین توسعه برنامه کاربردی وب استفاده نمایید.
- همچنین تجربه توصیه می‌کند از کامپایلرهای نسخه آخر استفاده نمایید، که اغلب شامل دفاع‌هایی در مقابل خطاهای کدنویسی است؛ برای نمونه، کامپایلر GCC از کد در برابر

2- Checklist
3- Rollback

سرریزهای بافر جلوگیری می‌کند.

- همانگونه که قبلا گفته شد، الگوهای طراحی امنیت را می‌توان برای رفع نگرانی‌های مرتبط با امنیت مشابه و فراهم نمودن راه‌حل‌هایی برای مشکلات شناخته‌شده استفاده نمود.
- بهتر این است که برای کدنویسی از بررسی خطا یا استثناء مناسب استفاده نمایید. مقادیر برگشتی هر تابع، به خصوص توابع مرتبط با امنیت را بررسی نمایید. همچنین، نشت اطلاعات حساس از کاربران غیر قابل اعتماد را بررسی نمایید.
- سیاست امنیتی را برای ممنوع نمودن استفاده از توابع نامناسب که کد را تضعیف می‌کنند، اتخاذ نمایید. همان‌گونه که در بخش "آزمایش امنیت نرم‌افزار" توضیح داده شد، از یک فرآیند بررسی کد جفتی و آزمایش امنیت پشتیبانی نمایید.
- ورودی HTML را رمزگذاری نمایید. مهاجمین از ورودی بدخواه برای هدایت و اجرای انواع حملات XSS استفاده می‌کنند. رمزنگاری هر ورودی کاربر می‌تواند مرورگر وب کلاینت را از تفسیر این کدها به عنوان کد اجرایی رها سازد. داده حساس را در کوکی‌ها ذخیره نکنید.
- کلیه داده‌های محرمانه را با استفاده از روش‌های رمزنگاری قوی، رمز نمایید. مدیریت کلید را به دقت اجرا نمایید. از یک الگوریتم رمزنگاری قوی و منتشر شده با طول کلید مناسب استفاده نمایید. بکار بستن الگوریتم رمزنگاری تاییدشده FIPS[1] را تشویق نمایید. پروتکل‌های امنیتی را با رمزنگاری ذاتا ضعیف (مانند SSL V۲) و اعداد تصادفی رمزنگاری ضعیف به‌کار نبندید.
- استفاده از شیوه‌های کدنویسی امن شامل حفظ آگاهی درباره آسیب‌پذیری‌های شناخته‌شده و جدید و خطاهای نرم‌افزاری با مطالعه انجمن‌های گفتگوی مرتبط با امنیت، مجله‌ها، مقالات تحقیقی و خبرنامه‌ها می‌باشد.
- هر سازمان باید به توسعه‌دهندگانش آموزش دهد که چگونه کد امن بنویسند (با برگزاری یک سمینار یا ارائه یک دوره آموزشی). ارائه درس باید راهبردها و اقدامات مناسب برای کم نمودن تهدیدات رایج را پوشش دهد. باید به ویژگی‌های امنیتی زبان‌های برنامه‌نویسی و چگونگی پیاده‌سازی آن ویژگی‌ها برای ساخت یک برنامه کاربردی امن تاکید نمود.

1- Federal Information Processing Standards

۷-نتیجه‌گیری

در این مقاله انواع آسیب‌پذیری‌های شناخته شده ناشی از کدنویسی مرور شد. این نکته حائز اهمیت است که توسعه دهندگان با آسیب‌پذیری‌های امنیتی شناخته شده آشنا باشند تا از نوشتن کدی که توسط این‌گونه آسیب‌پذیری‌ها مورد بهره برداری قرار می‌گیرند، امتناع ورزند. اجرای دوره‌های آموزشی توسعه امن نرم‌افزار مشتمل بر چگونگی نوشتن کد امن برای اجتناب از آسیب‌پذیری‌های اشاره شده در این مقاله نقش مؤثری در ارتقاء امنیت و کیفیت نرم‌افزارها خواهد داشت. توسعه‌دهندگان بایستی درک صحیحی از روش‌های کدنویسی امن برای کاهش خطاهای امنیتی و توسعه مناسب برنامه‌های کاربردی امن داشته باشند. در فاز تست نیز می‌توان با استفاده از روش‌های تست کدهای منبع (استفاده از ابزارهای Fuzzing، تحلیل کد ایستا، بازبینی و ارزیابی کد، تست‌های واحد و ...) موجب کاهش هزینه نگهداری و پشتیبانی از نرم‌افزارها گردید. کاهش رخنه‌های امنیتی موجب افزایش پایداری و امنیت برنامه کاربردی و حوزه نرم‌افزار می‌گردد، که گامی مؤثر در جهت برآورده نمودن اهداف مهم پدافند غیرعامل (تقلیل آسیب‌پذیری‌ها و کاهش خسارات و صدمات تأسیسات، تجهیزات و نیروی انسانی مراکز حیاتی، حساس و مهم نظامی و غیرنظامی کشور در برابر تهدیدات و حملات دشمن، و نیز بلایای طبیعی) می‌باشد.

مراجع

۱. پایگاه اطلاع‌رسانی پایداری ملی، سازمان پدافند غیرعامل کشور، شهریور (**۱۳۹۲**) "http://paydarymelli.ir/fa/about"

2. Nance Kara, Hay Brian, and Bishop Matt, "Secure Coding Education: Are We Making Progress?," in 16th Colloquium for Information Systems Security Education, Orlando, June (**2012**).

3. Robert C. Seacord, the CERT C Secure Coding Standard, 1st ed.: Addison-Wesley Professional, (**2008**).

4. Timothy Morrow, Robert Seacord, John K.Bergey, and Philip Miller, "Supporting the Use of CERT Secure Coding Standards in DoD Acquisitions," Carnegie Mellon University, Showcase CMU/SEI-2012-TN-016, October (**2012**).

5. P. Ammann and D. Wijesekera, "Scalable, Graph-Based Network Vulnerability Analysis," vol. 18 No. 22, pp. 217-224, (**2002**).

6. Soo Hoo Kevin, w. Sudbury Andrew, and R. Jaquith Andrew, "Tangible ROI through Secure Software Engineering: Special issue on Return on Security Investment," Secure Business Quarterly, Q4 (**2001**)

7. Russell L. Jones and Rastogi Abhinav, "Secure Coding: Building Security into the Software Development Life Cycle," Information Systems Security, vol. 20, no. 3, pp. 29-39, (**2004**).

8. H. Cheng Kwok, "Baking in Security During the Systems Development Life Cycle," CrossTalk the journal of Defense Software Engineering, vol. 03, pp. 22-25, March (**2007**).

9. Howard Michael, LeBalance David, and Viega John, 19 Deadly Sins of Sotware Security: Programming Flaws and How to Fix Them , 1st ed.: McGraw-Hill Osborne Media, (**2005**).

10. Davis Noopor, "Secure Software Development Software Life Cycle Processes: A technology Scouting Report," CMU/SEI-2005-TN-024, December (**2005**).

11. Michael Howard and Steve Lipner, The Security Development Life Cycle, 1st ed.: Microsoft Press, May (**2006**).

12. Erich Gamma, Richard Helm, Johnson Ralph, and John Vlissides, Design Patterns CD: Elements of Reusable Object-Oriented Software, 1st ed.: Addison-Wesley Professional, May (**1998**).

13. Markus Aderhold, Gebhardt Alexander, and Mantel Heiko, "Choosing a Formalism for Secure Coding: FSM vs. LTL," TU Darmstadt, Technical TUD-CS-2013-0180, June (**2013**).

Securing software based on secure coding

S. Jorjam [1]

M. Dehghani [2]

Abstract

Software security is concerned with methods of producing secure software. Concepts that are important to software security contain system risks management, programming language, platform, software audits, designing security and security tests. In other words, compliance with the security provisions in the production process can be considered an issue of passive defense in the field of software. In most cases, the programming mistakes that are easily avoided, lead to exploitable vulnerabilities in software. Reviews and analyses performed on thousand reported vulnerabilities, suggest that most vulnerabilities arise from a small number of common programming errors. We know that as soon as a vulnerability is discovered, it is easier and cheaper to fix. Application of the safe software development lifecycle, which investigates the security in each step of development to identify early potential vulnerabilities in each stage of development for the early identification of potential vulnerabilities is of utmost importance in securing software.Therefore, developers should understand secure coding techniques in order to reduce security errors and appropriate development of secure applications. This article suggests guidelines for secure coding.

Key Words: *Coding, Security, programming languages, Passive defense, buffer overflow, vulnerability*

1- MS Candidate, Imam Hussein Comprehensive University, (sjorjam@ihu.ac.ir) - Writer in Charge
2- Instructor and Academic Member of Imam Hussein Comprehensive University, (mdehghany@ihu.ac.ir)