소프트웨어 개발 방법론 이해

2016156007

소프트웨어학과 김영민

안전한 소프트웨어를 만들기 위해서 필요한 보안 활동들을 적용하여 개발하는 방법을 SW 개발보안 방법론이라고 합니다. 안전한 소프트웨어의 자격을 갖추기 위해서는 신뢰성을 위협받는 상황에서도 시스템을 신뢰 가능한 상태로 유지할 수 있도록 설계된 소프트웨어를 말합니다. 안전한 소프트웨어 개발을 위해서 SW개발보안 방법론에서는 요구사항 분석, 설계, 구현, 테스트, 유지보수와 같은 단계를 거처서 보안을 유지하고 있습니다.

첫째로, 요구사항분석 단계에서는 사용자의 관점에서 시스템의 요구사항을 수집하게 됩니다. 시스템의 기능, 성능, 인터페이스, 요구사항 등이 포함되고 분석 단계 작업 시에 누락된 요구사항이 없는지, 해당 요구사항은 제한된 사항에서 구현 가능한 것인지 등 타당성 조사를 하게 됩니다. 사용자의 요구사항과 시스템의 기능을 문서화해서 요구사항 명세서를 만들고 보안 활동으로 요구사항을 식별하는 보안활동을 추가합니다. 시스템으로 관리할 정보와 어느 정도의 보안 등급으로 정보를 지켜야 할지 등의 점검 작업이 이루어 집니다.

둘째로, 설계 단계에서는 요구사항 분석단계에서 산출된 요구사항들을 구현을 준비하여 설계도면에 옮기는 과정입니다. 설계 단계에서는 구현될 시스템의 성능에 직접적인 영향을 주게 되고 이 단계에서 산출물로 설계 명세서를 만들게 됩니다. 또한 이 단계에서 설계 명세서와 요구사양서를 토대로 사용자 지침서와 시험 계획서를 작성합니다. 보안 활동으로는 시스템을 분석해서 위협을 인지, 도출해내고 개발 보안 가이드가 제시하는 작업을 기존 프로세스에 추가하는 작업을 수행합니다.

셋째로, 구현 단계에서는 설계의 결과를 가지고 실제 프로그래밍을 하는 단계로 코딩을 거처 시스템의 기능이 실제 수행 가능한 형태로 나타내는 작업을 시행합니다. 구현 시에 발생하는 문제점은 보통 분석과 설계 단계에서 잘못 되어있는 경우가 많으므로 전 단계인 요구사항 분석과 설계단계에서 많은 시간과 투자가 필요합니다. 산출물로는 소스코드, 단위테스트 결과서, 결함-오류 보고서, 오류코드 정의서 같은 것들이 있습니다. 보안 활동으로는 표준 코딩정의서 또는 소프트웨어 개발 보안 가이드를 모든 개발자들이 준수하여 지키도록 하고 단위 테스트를 통해 보안 약점을 충분히 제거하며, 진단 작업을 통해서 소스코드 수준의 안정성이 보장되도록 해야 합니다.

넷째로, 테스트 단계에서는 통합테스트, 시스템테스트, 인수테스트를 거처 시스템의 오류를 발견하고 수정하는 단계로 여러가지 방법을 통해서 테스트하고 요구사항 분석, 설계, 구현의 모든 과정에 대한 최종점검을 시행합니다. 이 단계에서 통합테스트 결과서, 시스템 이행계획서와 같은 것들이 생성됩니다. 보안 활동으로는 설계 단계에서 도출된 위협들이 구현단계에서 해당 취약점들이 없는 앱이 개발되었는지 검증하는 작업을 수행합니다.

다섯째로, 유지보수 단계에서는 소프트웨어를 이용할 때 발생하는 문제점을 수정하고 새로운 기능을 추가해서 전보다 더 안정적인 소프트웨어로 발전시키는 단계입니다. 보통 시스템은 개발할 때부터 유지보수에 대비하여 만들어 지기 때문에 분석과 설계 단계에서의 사전 준비가 중요합니다. 보안 활동으로는 각 개발단계에서 노력하여 안정적인 소프트웨어를 만들었음에도 불구하고 발생될 수 있는 보안사고에 대해서 관리하고 대응하는 사후 관리가 중요합니다.

Secure Coding 7가지 요소

1. 입력 데이터 검증 및 표현

프로그램에 입력을 했을 때 해당 값의 검증이 누락되거나 잘못 검증, 잘못된 형식 지정, 일관되지 않은 언어 사용 등으로 인해 발생되는 보안 약점입니다.

예) 신뢰할 수 없는 URL 주소로 자동 접속 연결(CWE-601 URL Redirection to Untrusted Site ('Open Redirect'))

사용자 인증을 필요로 하는 페이지에 접근하게 되면, 사이트는 로그인 페이지로 이동시킨 뒤 로그인에 성공하게 되면 다시 이전 페이지로 돌아가기 위해 리 다이렉트 주소를 받게 됩니다. 이때 악의적인 사용자가 리 다이렉트 되는 페이지의 값을 악성 사이트로 변소해서 사용자에게 전송하게 하면 사용자의 로그인 정보가 노출되게 됩니다. 이러한 보안 약점을 대처하기 위해서 불필요한 리 다이렉트 기능을 제거하고 목적지 주소에 대한 화이트 리스트를 적용해서 대응할 수 있습니다. 사용자가 입력한 값을 자동 연결할 사이트 주소로 사용하는 경우에는 입력된 값이 화이트 리스트에 존재하는지 확인해야 합니다.

1. 보안 기능

인증, 접근제어, 기밀성, 암호화, 권환 관리 등 보안 기능을 부적절하게 구현할 때 발생할 수 있는 보안 약점으로 적절한 인증 없는 중요한 기능을 허용할 때, 부적절한 인가 등이 포함됩니다.

예) 중요한 자원에 대한 잘못된 권한 설정 (Incorrect Permission Assignment for Critical Resource)

소프트웨어가 중요한 보안 자원에 대해서 읽기 또는 수정 권한을 의도하지 않게 허가할 경우에 권한을 가지지 않은 사용자가 해당 자원을 사용하는 것을 말합니다. 자원이 프로그램 구성, 실행 또는 민감한 사용자의 데이터와 관련이 있는 경우에는 특히 위험한 보안 약점입니다. 이러한 보안 약점을 완화시기키 위해서는 설정 파일, 실행파일, 라이브러리 등은 소프트웨어 관리자만이 읽고 쓰도록 설정하고 설정파일과 같이 매우 중요한 자원을 사용하는 경우에는 허가 받지 않은 사용자의 접근이 가능한지 검사를 시행합니다.

즉 중요 자원에 대해서는 특정 사용자에게만 권한을 부여하도록 설정해야 합니다.

1. 시간 및 상태

동시에 또는 거의 동시에 수행하는 것을 지원하는 병렬 시스템이나 하나 이상의 프로세스가 동작되는 환경에서 시간 및 상태를 부적절하게 관리해서 발생할 가능성이 있는 보안 약점입니다.

예) 경쟁 조건 : 검사시점과 사용시점 (Time-of-check Time-of-use (TOCTOU) Race Condition)

병렬 시스템에서 자원을 사용하는 시점과 검사하는 시점이 다르기 때문에 자원의 상태가 변하는 경우가 자주 발생하게 됩니다. 하나의 자원에 대해서 동시에 검사시점과 사용시점이 달라 생기는 보안 약점이 발생하는데 이로 인해 동기화 오류와 함께 교착 상태 등과 같은 문제점들이 발생하게 됩니다. 소프트웨어는 이렇게 예상하지 못한 자원의 상태를 받았을 때 잘못된 작업을 수행할 수 있고 악의적인 공격자가 검사시점과 사용시점 사이에서 시스템에 영향을 줄 수 있게 되면 이후 파일, 메모리 또는 멀티 스레드 프로그램 변수와 같은 공유 리소스에서도 영향을 받을 수 있습니다. 이러한 보안 약점을 완화시키기 위해서는 여러 프로세스가 접근하여 사용하는 공유 자원을 동기화 구문을 활용하여 한번에 하나의 프로세스만 접근 가능하도록 하면서, 성능에 끼치는 영향을 최소화하기 위해서 임계코드 주변에만 동기화 구문을 활용합니다.

1. 에러 처리

에러를 처리하지 않은 경우나 또는 에러를 불충분하게 처리해서 에러 정보에 시스템 내부정보같은 중요한 정보가 포함될 때 발생할 수 있는, 에러를 제대로 온전하게 처리하지 않아 발생하는 보안 약점입니다.

예)오류 메시지를 통한 정보노출(Information Exposure Through an Error Message, MITRE)

소프트웨어가 환경, 사용자 또는 관련 데이터에 대한 민감한 정보가 포함된 오류 메시지를 생성할 때 발생하는 보안 약점입니다. 악의적인 공격을 시작하는데 유용하게 사용될 수 있는 민감한 정보들이 오류 메시지에 포함되어 외부에 제공하는 경우 공격자의 악성행위를 도울 수 있게 됩니다. 예외 발생시나 예외 이름, 스택 트레이스를 출력하는 경우에 공격자가 프로그램 내부 구조를 쉽게 파악할 수 있기 때문입니다. 이 보안 약점을 완화시키기 위해서는 오류 메시지가 정해진 사용자에게 최소한의 정보만 제공하도록 설정하고 소스코드에서 예외상황은 내부적으로 처리하고 사용자에게 민감한 정보를 포함하는 오류를 출력하지 않도록 사전에 작성된 오류 메시지를 출력하도록 설정합니다.

1. 코드 오류

타입 변환 오류, 자원의 부적절한 반환 같은 개발자의 실수에 의해서 발생할 수 있는 코딩 오류에 의해 유발되는 보안 약점입니다.

예) 초기화되지 않은 변수 사용 (Use of Uninitialized Variable, MITRE)

초기화되지 않은 변수를 사용했을 때 예측할 수 없거나 의도하지 않은 결과를 초래하는 보안 약점입니다. C언어 및 C++같은 일부 언어의 경우 스택 메모리에 저장되는 지역변수는 생성될 때 자동으로 초기화 되지 않는데, 초기화 되지 않은 변수를 사용할 경우 컴퓨터가 임의로 지정한 값을 사용하게 되어 의도하지 않은 결과를 출력하거나 예상치 못한 동작을 수행할 수 있습니다. C 나 C++은 가끔 발생하는 문제이지만 Perl이나 PHP의 경우에는 꽤 자주 발생하게 되는 보안 약점입니다. 초기화 되지 않은 스택 메모리 영역의 변수에 저장된 임의의 값은 이전 함수에서 사용되었던 내용 정보를 포함하고 있기 때문에 공격자는 이러한 부분을 공략하여 메모리에 저장되어 있는 값을 읽거나 또는 특정 코드를 실행할 수 있게 됩니다. 이 보안 약점을 완화하기 위해서는 반드시 모든 변수를 사용하기전에 올바른 초기값을 집어넣어서 초기화 해주어야 합니다. 변수를 명시적으로 선언할 필요가 없는 언어를 사용하는 경우에는 선언되지 않거나 알 수 없는 변수를 보고하는 모드에서 소프트웨어를 실행하거나 컴파일 하는 것이 좋습니다.

1. 캡슐화

중요한 데이터나 기능을 충분히 캡슐화 하지 못한 경우나 잘못 사용할 때에 발생하는 보안 약점입니다. 이로 인해서 정보가 노출되거나, 악의적인 공격자에 의해서 권한 문제등이 발생할 수 있습니다.

예) 제거되지 않고 남은 디버그 코드 (Leftover Debug Code, MITRE)

일반적으로 디버깅 또는 테스트 목적으로 특별히 설계된 코드는 개발이 완료된 시점에서 제거해야 합니다. 만약 이 디버깅 코드가 남겨진 상태로 배포가 될 경우에 공격자는 식별과정을 우회할 수 있거나 혹은 의도하지 않은 정보나 제어 정보가 노출될 수 있습니다. 이러한 예상치 못한 진입 경로는 설계 혹은 테스트 중에 고려되지 않은 코드이고 응용 프로그램의 예상 작동 조건을 벗어나므로 보안 약점이 발생할 수 있습니다. 이러한 보안 약점을 완화하기 위해서는 해당 소프트웨어를 배포하기전에 반드시 디버그 코드 삭제 여부를 확인해야 합니다. 일반적으로 자바 개발자의 경우에 웹 응용 프로그램을 작성할 때 디버그 용도의 코드를 main()에 개발한 후에 이를 삭제하지 않은 경우가 흔합니다. 디버깅을 위해 작성한 main() 메서드의 경우 개발이후에는 불필요하기 때문에 작성이 끝난 후에는 main()메서드를 삭제하는 것이 좋습니다.

1. API 오용

API를 사용할 때에 의도된 사용법이 아니거나, 보안에 취약한 API를 부주의하게 사용할 경우에 발생할 수 있는 보안 약점입니다.

예) 취약한 API 사용 (Use of Potentially Dangerous Function)

보안상으로 취약한 API를 이용하거나 API가 의도되지 않는 방식으로 작동될 경우 발생할 수 있는 취약점입니다. 대표적으로 DNS lookup에 의존한 보안, 완전히 지워지지 않는 힙 메모리, 안전하지 않는 함수 이용 등이 존재합니다. 보안상 문제가 없다 하더라도 함수를 잘못된 방식으로 사용할 때 역시 보안 문제를 발생시킬 수 있습니다. 이러한 보안문제를 완화하기 위해서는 보안문제로 인해 금지된 함수대신에 대체할 수 있는 안전한 함수를 사용하는 것이 바람직합니다. 예를 들어 금지된 API들인 strcpy(), strncat(), sprint()등 대신에 strcpy()\_s, strncat\_s(), sprint\_s()를 사용해주는 것입니다. 취약한 API의 경우에는 strtol() 같은 함수 대신에 작은 크기의 부호 있는 정수인 int, short, char 같은 자료형 반환에 사용하면 범위 제한 없이 값을 평가할 수 있게 됩니다. 만약 대체가 가능한 API를 사용할 수 없는 경우에는 API의 인자와 반환값을 검사하는 것이 좋습니다.