http://dx.doi.org/10.7236/JIWIT.2012.12.5.237
JIWIT 2012-5-30

실행코드 취약점 분석 프로세스 방법론

A Methodology for Security Vulnerability Assessment Process on Binary Code

황성운*

Seong-Oun Hwang

요 약 공격 대상 소프트웨어의 취약점을 악용한 사이버 공격이 급속히 증가하여 왔다. 그러나, 이러한 취약점을 탐지하고 대처하는 것은 매우 어렵고 시간이 많이 걸리는 작업이다. 이 문제를 효과적으로 대응하기 위하여, 본 논문에서는 실행코드 상에서의 체계적인 보안 취약점 분석 프로세스 방법론을 제시한다. 구체적으로, 본 연구진은 기존 취약점을 웹 환경 유무, 대상 소프트웨어 특성 등을 고려하여 분류하고 취약점 리스트 및 범위를 결정하는 접근법을 택하였다. 향후 연구 방향으로는 현재 도출된 방법론을 좀 더 구체화하고 검증하는 과정이 필요하다.

Abstract Cyber attacks have rapidly increased by exploiting the underlying vulnerabilities in the target software. However, identifying and correcting these vulnerabilities are extremely difficult and time consuming tasks. To address these problems efficiently, we propose a systematic methodology for security vulnerability assessment process on binary code in the paper. Specifically, we first classified the existing vulnerabilities based on whether the target software run in a Web environment and features of the software. Based on the classification, we determined the list and scope of the vulnerabilities. As our future research direction, we need to further refine and validate our methodology.

Key Words : Security Vulnerability, Security Analysis, Fuzzing, Tainting, Cyber Attack

Ⅰ. 서 론

소프트웨어는 특성상 복잡도가 높고 규모가 방대하여 취약점 분석이 어려우며 이 때문에 해당 소프트웨어의 취약점이 해커의 공격 대상이 되고 있다. 특히 상용 소프트웨어는 취약점 분석가에게 소스 코드가 주어지지 않는 경우가 많으며 이 때문에 취약점 분석이 어려웠으며, 분석은 주로 역공학 (reverse-engineering)에 의존해 왔다. 그런데 역공학은 특성상 기술 진입 장벽이 높고, 상용 소프트웨어 실행 코드가 방대해서 이를 분석하는데 많은 시간과 노력이 소요되는 실정이다. 따라서, 시간을 줄이면서도 정확하게 취약점을 찾아낼 수 있는 역공학 기술에 기반한 취약점 분석 기술 개발이 필요하다.

본 연구의 목표는 상용 소프트웨어의 보안 취약점의 다양한 유형을 분석하고, 취약할 수 있는 부분을 효과적으로 찾으며, 취약할 수 있는 부분이 실제 취약점으로 나타나는지 검증할 수 있는 취약성 분석 프로세스를 개발하는 것이다.

*정회원, 홍익대학교 컴퓨터정보통신공학과
접수일자: 2012년 8월 30일, 수정완료 : 2012년 9월 30일
게재확정일자: 2012년 10월 12일

Received: 30 August 2012 / Revised: 30 September 2012 / Accepted: 12 October 2012
*Corresponding Author: sohwang@hongik.ac.kr
Dept. of Computer & Information Communications Engineering, Hongik University, Korea

II. 취약점 분석 프로세스

1. 개관

소프트웨어 취약점(software vulnerability)이란 일반적으로 소프트웨어 개발자가 개발 또는 유지보수 단계에서 의도하지 않은 형태로 사용되어 보안 문제를 발생하는 경우를 말한다. 일반적으로 상용 소프트웨어는 소스 코드를 외부에 공개하지 않고 개발자 그룹에 국한하는 경우가 많으며 소프트웨어 보안성 분석은 회사 내외부 보안 테스팅 전문가에게 맡기는 것이 보통이다. 때문에 상용 소프트웨어 보안성 분석의 경우 대부분 바이너리 코드 분석에 의존하여 취약점을 분석하게 된다.

그러나 바이너리 코드 분석은 소스 코드 분석에 비해 어셈블리 코드를 대상으로 하며, 분석 범위가 방대하며, 여러 사람이 팀을 이뤄 분석을 하는 경우 작업 할당에 있어서 중복 문제가 발생할 수 있으며, 접근 방식에 따라 분석에 소요되는 시간 및 비용이 많이 차이가 발생할 수 있기 때문에, 체계적인 취약점 분석 프로세스가 필요하다. 그림 1은 본 연구에서 제안하는 취약점 분석 프로세스를 나타낸 것이다. 다음에서는 분석 프로세스를 구성하는 각 부분들을 자세히 설명한다. 또한 각 프로세스에 대한 이해를 돕기 위해 본 연구에서 사용된 취약점 분석 대상 소프트웨어 A, B를 대상으로 중심으로 관련 사례를 제시하였다. 참고로 A, B는 각각 세계적으로 유명한 메신저 프로그램 및 압축 프로그램이다.

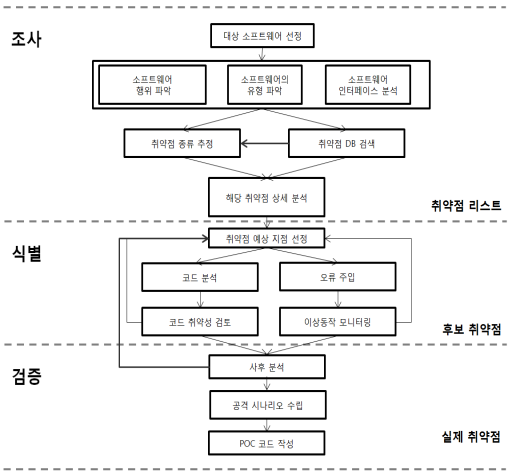

그림 1. 취약점 분석 프로세스
Fig 1. Security Vulnerability Analysis Process

2. 전반적인 기능 및 동작 파악

세부 분석에 앞서 분석 대상 소프트웨어의 전반적인 기능 및 동작을 파악하는 것이 중요하다. 이것은 크게 행위 및 명세 분석, 유형 파악, 인터페이스 분석 등의 과정을 통해 이루어진다.

행위 분석 단계에서는 특정한 문서나 소스 없이 프로그램을 동작시키면서 어떠한 기능들이 실행되는지 전체적인 기능을 대략적으로 파악한다. 예를 들어, 메신저 프로그램의 경우 분석가는 이를 실행하여 메시지 기능, 이모티콘 기능, 원격 접속 기능 등 각 기능들을 파악하고 기능들 사이의 관계를 파악한다. 압축 프로그램의 경우에도 사용될 압축 알고리듬이 다양하며 다양한 길이의 암호화 옵션을 선택하는 등 사용자에게 선택의 여지가 많다. 따라서, 이러한 기능들을 실행해 봄으로써 전체 기능을 파악하는 것이 중요하다. 명세 분석 단계에서는 프로그램 명세서, 사용자 매뉴얼 등 부가적인 정보를 참조하여 기능을 파악한다. 명세 분석은 단순한 행위 분석 단계에서 파악하지 못한 정보를 파악할 때 도움이 된다.

소프트웨어 유형은 소프트웨어가 실행되는 환경 및 소프트웨어가 가지는 고유한 특성에 따라 분류할 수 있다. 분석 대상 소프트웨어가 Web 환경에서 실행되는지 Non-web 환경에서 실행되는지에 따라 발생하는 취약점 종류에 차이가 있을 수 있다. Web 환경에서는 XSS, injection 종류의 취약점이 많이 발생하고 Non-web 환경에서는 버퍼 오버플로우, 에러 핸들링 종류의 취약점이 많이 발생한다. 취약점 발생 양상에 차이가 있기 때문에 Web 어플리케이션인지 Non-web 어플리케이션인지 파악하는 단계가 필요하다. 예를 들어, 압축 프로그램의 경우 Non-web 환경에서 실행되며, Web 환경과 연관된 취약점이 발견되기 어렵다. 하지만, 메신저 프로그램의 경우에는 Non-web 어플리케이션이지만, 메신저 프로그램의 특성상 Web과 관련된 취약점이 나타날 수도 있어 향후에 취약점 우선순위를 Non-web 취약점과 Web 취약점을 모두 고려하여 우선순위를 결정해야 한다.

취약점은 대부분 외부 입력에 의해 발생한다 (물론 내부 입력에 의해서도 발생가능하지만 본 연구에서와 같이 실행코드만 주어진 상황에서 이를 발견하기란 현실적으로 매우 어렵다). 따라서 외부에서 어떠한 인터페이스를 받아들이는지에 대한 파악이 필요하다. 여기서 인터페이스란 외부 파일 입력, 네트워크 사용 여부, 외부 데이터 입력 부분, 사용자 인터페이스 (GUI, CLI 등)가 될 수 있

으며 이러한 인터페이스의 사용 빈도에 따라 취약점 발생 양상에 차이가 있을 수 있다.

3. 취약점 종류 추정

소프트웨어 인터페이스에 근거해서 취약점을 추정할 수 있다. 예를 들어 외부 파일의 입력을 받는 오피스 프로그램이나 멀티미디어 프로그램, 압축 프로그램은 파일을 읽어 들이는 부분에서 오버플로우 형태의 취약점이 발생할 수 있다. 이와 달리, 메신저 프로그램은 외부에서 파일을 첨부하여 전송할 때 파일 입력이 일어난다.

또한 소프트웨어 실행환경 (Web/Non-Web)에 근거해서 추정할 수도 있다. Web 어플리케이션에서는 XSS, SQL Injection등 웹과 관련된 취약점이 다수 발생하는 것을 볼 수 있다. 그리고 Non-web 어플리케이션에서는 버퍼 오버플로우나 에러 핸들링 등 메모리 관리 문제로 인해 취약점이 많이 발생한다.

발생 가능한 취약점 우선순위 리스트는 먼저 프로그램 동작 환경에 따라 Web과 Non-Web으로 분류 (예: 표 1) 할 수 있고 프로그램 특성에 따라 분류 (예: 표 2)할 수 있다. 따라서 표 1, 표 2의 취약점 우선순위를 고려하여 분석가 자의적인 판단 하에 표 3을 얻을 수 있다. 표 1은 [1] 에서 제시한 취약점 우선순위표를 참고하여 수정하였다.

표 1. 프로그램 환경별 취약점 우선순위 리스트
Table 1. Vulnerability Priority List by Program Environments

우선순위	Web	Non-Web
1	XSS	Buffer Overflow
2	Information Leakage	Error Handling
3	CRLF Injection	Potential Backdoor
4	Cryptographic Issues	Directory Traversal
5	SQL Injection	Cryptographic Issues

표 2. 프로그램 특성별 취약점 우선순위 리스트
Table 2. Vulnerability Priority List by Program Characteristics

우선 순위	A 소프트웨어	B 소프트웨어
1	Buffer overflow	Error handling
2	Error handling	Buffer overflow
3	Information Leakage	Buffer Management Errors
4	Cryptographic Issues	Numeric Errors
5	Untrusted Search Path	Information Leakage

표 3. 환경 및 프로그램 특성을 고려한 취약점 우선순위 리스트
Table 3. Vulnerability Priority List by Program Environments and Characteristics

우선 순위	A 소프트웨어	B 소프트웨어
1	Buffer overflow	Buffer overflow
2	Error handling	Error handling
3	Information Leakage	Buffer Management Errors
4	Cryptographic Issues	Numeric Errors
5	Directory Traversal	Cryptographic Issues

4. 취약점 DB 검색

취약점은 기존에 발생했던 유사한 취약점이 또 발생할 수 있으며 이전에 취약점이 발생하여 패치한 부분 주변에서 발생하는 경우도 존재한다. 따라서 취약점 DB에서 해당 프로그램을 검색하여 기존 취약점을 분석하는 것은 취약점을 분석하는 것에 도움을 줄 수 있다. 해당 프로그램을 취약점 DB에서 검색했으나 직접적인 정보가 별로 없다면 분석 대상 프로그램과 유사한 프로그램의 취약점 정보를 검색하는 것이 도움을 줄 수 있다. 실제 본 연구에서도 실험 대상 관련 취약점 분석 정보가 많지 않아서 유사 프로그램 취약점 분석 정보를 조사하고 활용하였다.

5. 해당 취약점 상세 분석

기존 DB [2-7]에 있는 취약점 및 해당 Exploit code가 존재 하는 경우 Exploit code를 통하여 취약점을 발현 시킬 수 있고 Exploit code를 분석하여 어떻게 취약점이 발현되는지에 대한 정보도 얻을 수 있으므로 취약점을 분석하는데 용이하다. 기존 DB에서 관련 정보를 찾을 수 없을 경우는 취약점 유형에 관한 자료조사(예: CWE)를 바탕으로 해당 취약점에 관한 특성을 파악 후 이 특성들로 취약점 예상 지점 선정 단계로 진행한다.

6. 취약점 예상 지점 선정

앞에서 분석한 기존 취약점 분석 내용이 있는 경우 분석 내용을 기반으로 취약점 예상 지점을 선정해야 한다. 예를 들어 앞에서 분석한 내용에서 외부에서 파일을 받아들이는 부분에서 발생한 취약점이 있었다면 분석 내용을 바탕으로 분석 대상 프로그램의 파일을 받아들이는 부분을 예상 지점으로 삼아야 할 것이다. 또한 이미 패치가 된 부분이라도 패치가 된 부분에서 취약점이 발생 하

는 경우가 있으므로 패치가 된 부분을 예상 지점으로 선정할 수도 있다. 또한 앞에서 취약점 분석 내용이 없는 경우 분석가의 경험과 파악한 내용(입력 인터페이스 등)으로 취약점 예상 지점을 선정해야 한다. 이 경우 분석가의 경험과 분석 능력에 영향을 많이 받게 된다.

7. 코드 분석

역공학을 통한 바이너리 분석은 코드의 크기가 방대하여 지나치게 시간이 많이 걸릴 수 있는데 프로그램 명세서, Map File 등의 부가 정보가 있을 경우 취약점 예상 지점으로 분석의 범위를 좁힐 수 있어 시간 낭비를 줄이면서 취약점 예상 지점을 효과적으로 분석할 수 있다. 부가 정보가 전혀 없을 경우 역공학만을 이용하여 바이너리 분석을 해야 한다. 이 경우 시간이 많이 소요될 수 있으므로 정밀조준(Pin Point) 등의 분석방법을 활용해야 하며 분석가의 실력과 경험에 의지하게 된다.

8. 코드 취약성 검토

분석가가 해당 지점을 분석하여 가능성을 판별하고 가능성 여부에 따라 다음 두 과정으로 진행할 수 있다. 예를 들어 외부 문자열을 받아들이는 코드에서 문자열의 길이를 검사하는 코드가 없을 때 이런 코드가 버퍼 오버플로우에 취약하다고 판단할 수 있다. 만일 취약성 가능성이 있다면 검증 작업으로 진행하고, 가능성이 없다면 취약점 예상 지점 선정 단계로 진행한다.

9. 오류 주입

코드 취약성 검토를 위해 위와 같이 코드 분석을 행할 수도 있지만 보다 더 적극적인 방법으로 오류 주입을 통해서도 할 수 있다. 오류 주입은 크게 수동적인 방법과 자동화된 방법으로 나뉠 수 있다.

예를 들어 분석가가 수동으로 오류를 주입하여 에러 상황을 모니터링 할 수 있다. 이 경우 오류를 주입하는 방법에 따라 취약점 발현 여부가 결정 될 수 있다. 또한 수동 조작 과정은 분석가의 분석 내용과 경험에 상당부분 의존하게 된다.

취약점 검출을 자동화 할 수 있는 방법은 퍼징(fuzzing), 테인팅 (taint analysis)과 같은 방법들이 있다. 퍼징 [8]은 데이터를 어플리케이션에 전달하여 에러를 유도함으로써 발생되는 예외를 모니터링하는 소프트웨어 테스팅 방법이다. 주로 프로그램의 입력 부분에 테스트 값을 생성하여 입력하고 크래쉬, 메모리 위반 등의 예외 상황을 모니터링 한다. 반면에, 테인팅 [9]은 어떤 프로그램에 외부 입력이 발생했을 때 미치는 영향 (예: 데이터가 어디까지 영향을 미치고 어떤 생명 주기를 갖는지를 파악, 취약점 발현, 외부 입력으로 인한 분기 상황 등)을 파악하는 것이다. 이 기법에서는 외부 입력값을 모두 신뢰할 수 없는 값으로 가정하고 이 값들의 흐름을 추적한다.

이러한 자동화 방법은 수동 방법보다 많은 종류의 경우를 테스트 할 수 있고 시간 낭비를 줄일 수 있다는 장점이 있다. 자동화된 오류 주입 분석 기법에 관한 자세한 실험 내용은 논문 [14]를 참조하기 바란다.

10. 이상 동작 모니터링

앞의 방법을 이용하여 이상동작을 모니터링 하였을 때 예상했던 이상 동작이나 예외가 발생하였는지의 여부에 따라 두 단계로 진행 할 수 있다. 이상 동작을 발견하였으면 검증 단계로 진행하며, 이상동작을 발견하지 못하였으면 취약 예상 지점 선정 단계로 진행한다.

11. 사후 분석

이전 단계에서 했던 퍼징, 테인팅, 수동 분석의 결과 (예: 크래쉬, 이상동작)가 실제 취약점으로 발현되지 않을 수 있다. 따라서 취약점으로 발현될 수 있는지에 대해서 이 단계에서 판단해야 한다. 예를 들어 퍼징의 결과로 오버플로우가 발생하여 레지스터 영역을 침범하는 경우가 발생하였다면 상세 분석을 통해 확실히 취약점으로 발현될 수 있다고 판단할 수 있다. 다른 경우들도 마찬가지로 상세 분석을 통하여 취약점으로 발현될 수 있는지를 판단한다. 리뷰를 하기 전에 자체적으로 취약점 발현 여부에 대한 판단이 가능하나 리뷰를 함으로써 오진을 감소시킬 수 있다. 또한 분석가의 분석 내용을 공유할 수 있고 분석가가 놓칠 수 있는 부분을 줄일 수 있다.

12. 공격 시나리오 설정 및 POC 코드 작성

취약점이 발견되었으면 이 취약점으로 어떤 악용이 가능한지에 따라 공격 시나리오를 작성하게 된다. 공격 시나리오에는 공격 대상 운영체제, 공격 대상 버전, 공격 대상에 적용되는 보호 기법(ASLR, GS 등) 등을 고려해야 한다. 이러한 공격 시나리오를 바탕으로 POC 코드를 작성하게 된다. 실제로 본 실험에서 메신저 프로그램의 경우, 공격자가 탈취한 계정으로 접속하여 메신저의 취

약점을 이용하여 버디 목록에 있는 수신자가 모르게 실행 파일을 보낸 다음 이를 다른 취약점 등을 활용하여 실행하는 시나리오를 설정할 수 있었다.

Ⅲ. 취약점 분석 프로세스 적용 실험 결과

1. 발견된 취약점 분석

아래 표는 실험을 통해 발견된 대표적인 취약점을 나타낸 것이다.

표 4. 발견된 취약점 분석 비교
Table 4. Analysis of Security Vulnerabilities

취약점	A 소프트웨어	B 소프트웨어
버퍼오버플로우 취약점 (CWE120)	O	O
정보노출 취약점 (CWE200)	O	O
에러핸들링 취약점 (CWE388)	O	X
업로드/다운로드 취약점 (CWE494)	O	X
ISO 포맷 취약점	X	O

2. 오류 주입 과정에서 수동 방법과 자동화된 방법의 비교

다음에서는 본 실험 대상 중 하나인 A 소프트웨어를 IDA Pro [10] 라는 정적 분석 도구를 이용해서 분석한 결과, 아래 그림과 같이 위와 아래 그림 각각에서 SEH overwrite 및 접근 위반(access violation)이 발생하는 것을 확인하였다 (CWE120). 본 실험에서는 Corelan팀 [11]에서 개발한 Tracer와 In-Memory Fuzzer를 이용하였다.

본 연구에서는 취약 가능성과 발현 가능성이 높은 곳을 발견하기 위해 퍼징을 수행하기 전에 먼저 테인팅 기법을 도입하였다. 여기서 발현 가능성이 높은 기준 중 하나로 코드 커버리지 (coverage) 개념을 사용하였다. 즉, 외부 입력 발생시 취약 가능성이 높은 부분이라 하더라도 코드 커버리지에 속하지 않으면 실행되지 않으므로 취약점이 발현될 수 없다. 따라서 외부 입력시 코드 커버리지에 속하는 부분을 알아내기 위해서 테인팅 기법을 활용하였다. 즉, 실행코드 상에서 외부로부터 입력된 데이터가 어떻게 흘러가는지를 테인팅 기법을 이용해 발현 가능성이 높은 취약점 후보 집합을 찾아내고 이 부분을 대상으로 실제적으로 퍼징을 적용함으로써 취약점을 검증하였다. 즉, 테인팅/퍼징 기법을 통한 자동화된 분석 기법을 적용한 결과 수동 분석에 비해 비교할 수 없을 정도로 빠른 시간 안에 두 배 이상의 취약점을 발견할 수 있음을 확인할 수 있었다. 보다 더 자세한 내용은 논문을 참조하기 바란다.

3. 관련 연구 비교

[12]에서는 멀티미디어 취약점을 퍼징 기법을 이용하여 조사하였다. 멀티미디어 취약점은 멀티미디어 플레이어 취약점과 미디어 포맷 (멀티미디어 파일, 멀티미디어 자막, 재생 목록 등)으로 나뉘는데 주로 미디어 포맷의 취약점 분석을 위해 다량의 램덤 데이터를 미디어 파일에 주입하여 소프트웨어를 실행할 때 변화를 모니터링하여 취약점을 찾아내는 퍼징 기법을 사용하고 있다. 이 실험을 통해 divide by zero, accessed invalid array index와 같은 일부 취약점을 찾아내었지만 스택, 힙 오버플로우 같은 위험성이 높은 취약점을 거의 찾을 수 없었다고 한다. [13]에서는 [12]와 비슷하게 분석 대상을 멀티미디어 플레이어 및 미디어로 국한하며, 분석 방법도 퍼징에 국한하고 있다. 다만 보다 더 다양한 퍼징 기법을 다양한 플레이어 및 미디어에 적용하고 있는 부분이 차이라고 할 수 있다. [13]에서 제시된 분석 방법론은 주로 퍼징 기법을 중심으로 다루고 있으나, 본 제안 방법론은 퍼징 기법과 같은 자동화된 분석 방법 뿐만 아니라 수동 분석도 포함하여 범위가 넓고, 분석 대상도 멀티미디어를 포함하여 일반적인 소프트웨어 환경에 적용한다는 점에서 차이가 있다.

Ⅳ. 결론

상용 소프트웨어의 경우 일반적으로 소스 코드가 분석가에게 제시되지 않으므로 (본 연구에서는 이러한 모델을 따르고 있음), 소스 코드 기반 취약점 분석에 비해 분석 대상에 대한 정보가 부족하여 실제 분석 과정에서 어려운 점이 많다. 또한 분석을 위해서는 기존에 알려진 취약점을 분석해야 하는데 기존 취약점 규모 또한 방대한 것도 연구를 시작하는데 어려움을 준다. 이러한 문제점들을 해결하기 위해, 본 연구진은 기존 취약점을 웹 환

경 유무, 대상 소프트웨어 특성 등을 고려하여 분류하고 여기에 우선순위를 두어 앞으로 연구할 취약점 리스트 및 범위를 결정하는 접근법을 택하였다.

향후 연구 방향으로는 현재 도출된 프로세스를 좀 더 구체화하고 검증하는 과정이 필요하다. 예를 들어, 이 프로세스를 실제 기업 환경에 적용함으로써 현재 프로세스에서 미진한 영역을 찾아 보완하는 작업이 필요할 것이다. 특히 현업에 근무하는 분석가들의 숙련된 경험을 반영하는 작업이 필요할 것이다. 또한 현재 연구가 활발히 이루어지고 있는 퍼징 및 테인팅 분야에 대한 연구를 심화함으로써 취약점 발견 과정을 자동화하는 것이 필요하다. 현재 퍼징 및 테인팅은 일부 기술 및 환경에는 부분적으로 적용 가능하나 아직 범용화 되지 못한 실정이다. 따라서, 범용화를 위한 기술 개발이 필요하며, 높은 오진율, 방대한 출력 결과로부터 취약점으로 발현될 수 있는 데이터 추출 등도 시급히 해결해야 한다.

참 고 문 헌

[1] Veracode, "State of Software Security Report", 2011.

[2] SANS, "CWE/SANS TOP 25 Most Dangerous Software Errors", http://www.sans.org/top25-software-errors, 2011.

[3] CVE List, "http://cve.mitre.org/cve".

[4] OSVDB, "http://www.osvdb.org".

[5] Exploit DB, "http://www.exploit-db.com".

[6] CWE List, "http://cwe.mitre.org/data/index.html".

[7] Metasploit, "www.metasploit.com".

[8] M. Sutton, A. Greene and P. Amini, "Fuzzing Brute Force Vulnerability Discovery", Addison-Wesley, 2008.

[9] B. Edgar, "Taint Analysis", Hackers to Hackers Conference, 2009.

[10] IDA Pro, http://www.hexblog.com.

[11] Corelan, "In Memory Fuzzing", http://www.corelan.be/index.php/2010/10/20/in-memory-fuzzing, 2010.

[12] Colleen Lewis, Barret Rhoden, Cynthia Sturton, “Using Structured Random Data to Precisely Fuzz Media”, http://www.eecs.berkeley.edu/~csturton/classes/cs261/fuzz_media_players.pdf, 2007.

[13] Yong Su Park et al., Window Multimedia Vulnerabilities Analysis Study, KISA, 2009.

[14] Seong Oun Hwang, Finding Vulnerabilities in Binary Codes Using Tainting/Fuzzing Analysis, 6th International conference on Convergence and Hybrid Information Technology (ICHIT), CCIS, vol. 310, 2012.

저자 소개

황 성 운(정회원)

- 1993년 : 서울대학교 수학과 (학사)
- 1998년 : 포항공과대학교 정보통신학과 (석사)
- 2004년 : 한국과학기술원 전자전산학과 (박사)
- 2008년 ~ 현재 : 홍익대학교 컴퓨터정보통신공학과 교수

<주관심분야 : 정보보호, 프로그래머블 로봇>