알약
월간 보안동향
보고서.

2014년 10월

알약 10월 보안동향보고서

CONTENTS

9월의 총평

9월에는 리눅스의 배시(Bash) 버그의 보안 취약점이 발견된 이슈가 가장 주목을 받았다. Bash(Bourne Again Shell)란 리눅스, 유닉스, Mac OS X 시스템에 내장된 명령어 Shell 프로그램인데, 이 Shell의 취약점을 이용하여 서버로부터 다양한 정보를 추출할 수 있다. 또한 단순 정보 추출뿐만 아니라 공격자가 원격에서 손쉽게 임의의 코드를 서버에서 실행시킬 수 있는 문제 때문에 더욱 높은 위험성을 가졌다고 볼 수 있다.

이외에도 애플 아이클라우드(iCloud)에 저장된 해외 유명 연예인들의 사진이 대거 유출되는 이슈도 발생했다. 해당 유출은 애플의 아이클라우드가 직접적으로 해킹된 것이 아니라, 나의 아이폰 찾기(Find My iPhone) 기능의 취약점을 노렸거나, 비밀번호 무차별 대입 공격(Brute Force Attack)으로 인해 로그인 계정 정보가 유출되었을 가능성이 큰 것으로 확인되고 있다.

Part1. 9월의 악성코드 통계

악성코드 통계

허니팟/트래픽 분석

스팸메일 및 악성코드가 포함된 메일 분석

스미싱 분석

1.악성코드 통계

감염 악성코드 TOP15

감염 악성코드 Top 15는 사용자 PC에서 탐지된 악성코드를 기반으로 산출한 통계이다.
2014년 9월의 감염 악성코드 TOP 15에서는 8월에 이어 두 달 연속으로 Keygen 악성코드가 1위를 차지했다. 2위를 차지한 Trojan.Generic.11469137의 경우 역시 지난 8월에 이어 연속으로 2위를 차지했다. 3위는 Gen:Variant.Adware.Symmi.42600가 차지했는데 이 악성코드의 경우, 일단 설치되면 사용자 모르게 사용자PC에 지속적으로 광고팝업을 띄우거나 정상적인 웹서핑이 이뤄지지 않도록 다양한 행위를 하며, 사용자의 웹 서핑 기록을 수집하는 애드웨어이다.

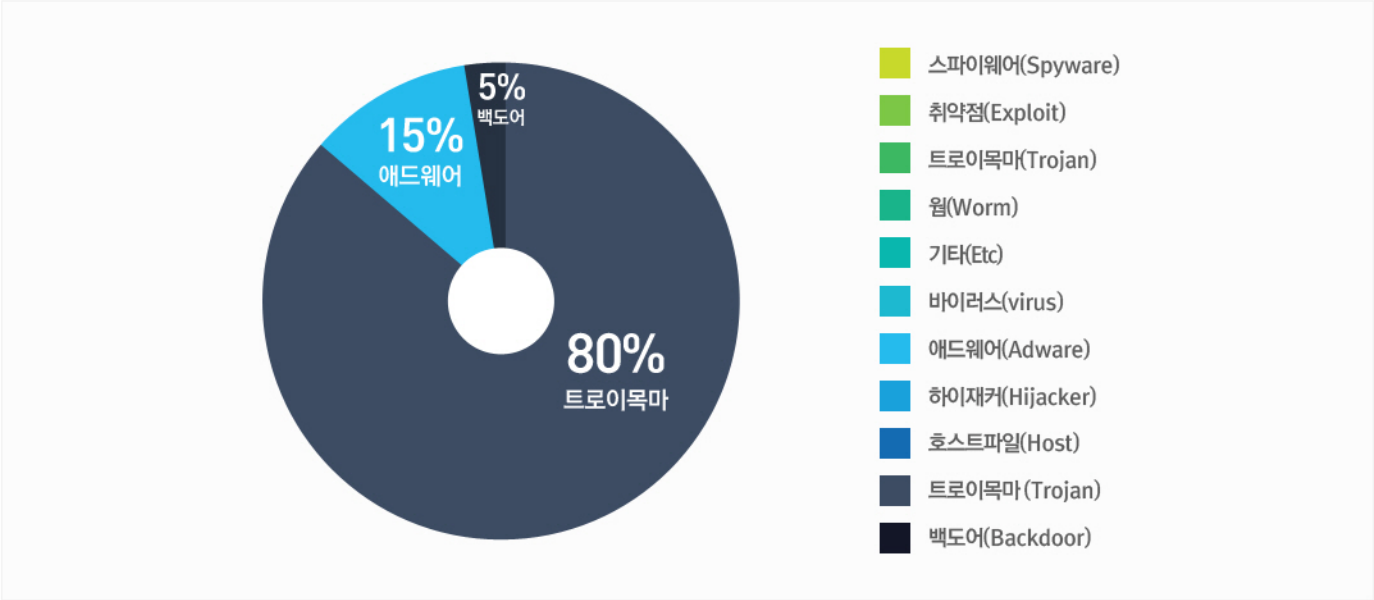
순위	등락	악성코드 진단명	카테고리	합계(감염자수)
1	-	Misc.Keygen	Trojan	1,236
2	-	Trojan.Generic.11469137	Trojan	1,174
3	NEW	Gen:Variant.Adware.Symmi.42600	Adware	1,040
4	↓ 3	Trojan.GenericKD.1826552	Trojan	898
5	↓ 1	Misc.Agent.126672	Trojan	875
6	-	Gen:Variant.Zusy.103708	Trojan	740
7	NEW	Gen:Trojan.Heur.4yWavLjkRxpGn	Trojan	617
8	NEW	Gen:Variant.Adware.Symmi.42529	Adware	606
9	NEW	Trojan.GenericKD.1719819	Trojan	597
10	-	Trojan.Heur.TP.Mr2@b0XV1LjO	Trojan	562
11	↓ 3	Backdoor.Xtreme.gen	Backdoor	493
12	↓ 1	Trojan.GenericKD.1697656	Trojan	480
13	NEW	Gen:Variant.Graftor.154453	Trojan	470
14	NEW	Gen:Variant.Strictor.57762	Trojan	464
15	NEW	Trojan.GenericKD.1832017	Trojan	461

*자체 수집, 신고된 사용자의 감염통계를 합산하여 산출한 순위임

2014년 09월 01일 ~ 2014년 09월 30일

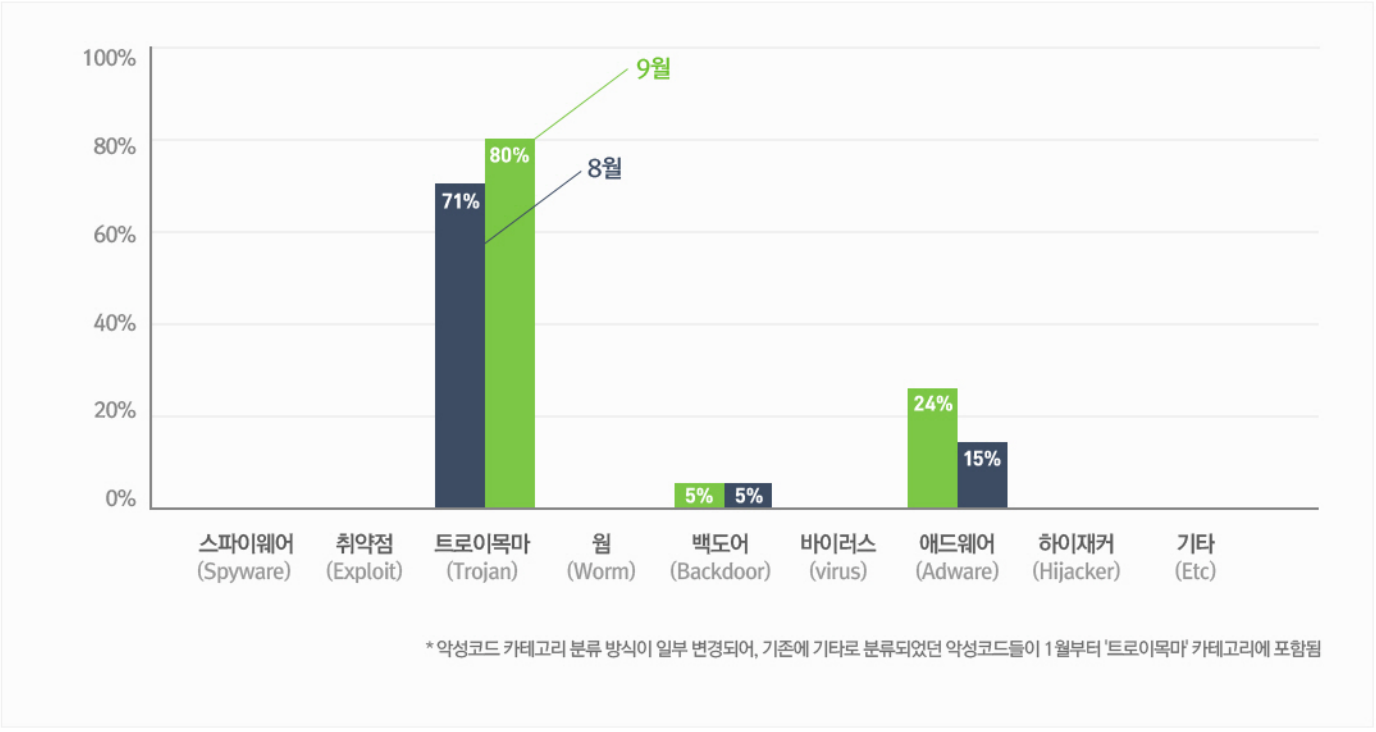
악성코드 유형별 비율

악성코드 유형별 비율에서 트로이목마(Trojan) 유형이 가장 많은 80%를 차지했으며, 애드웨어(Adware) 유형이 15%로 그 뒤를 이었다.



카테고리별 악성코드 비율 전월 비교

9월에는 지난 8월과 비교하여 트로이목마(Trojan) 유형 악성코드 비율이 소폭 감소하였고, 애드웨어(Adware)유형 및 백도어(Backdoor) 악성코드의 비중은 크게 증가했다.

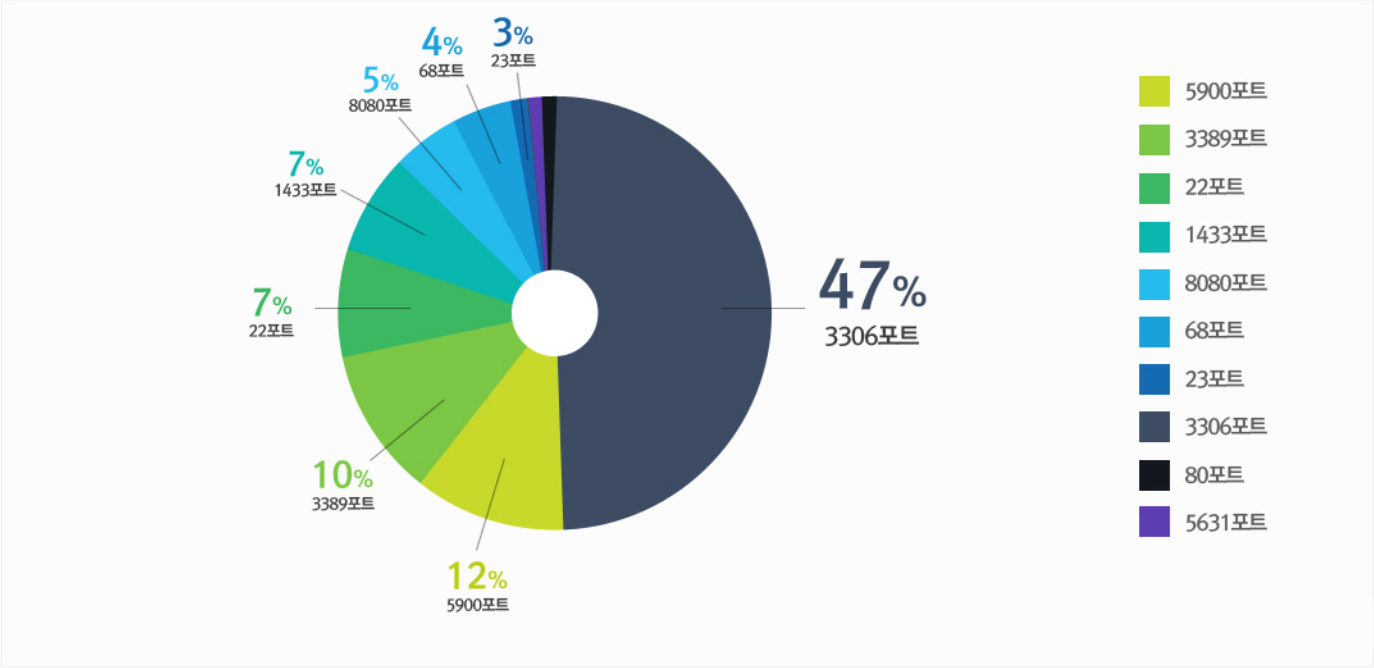


* 악성코드 카테고리 분류 방식이 일부 변경되어, 기존에 기타로 분류되었던 악성코드들이 1월부터 '트로이목마' 카테고리에 포함됨

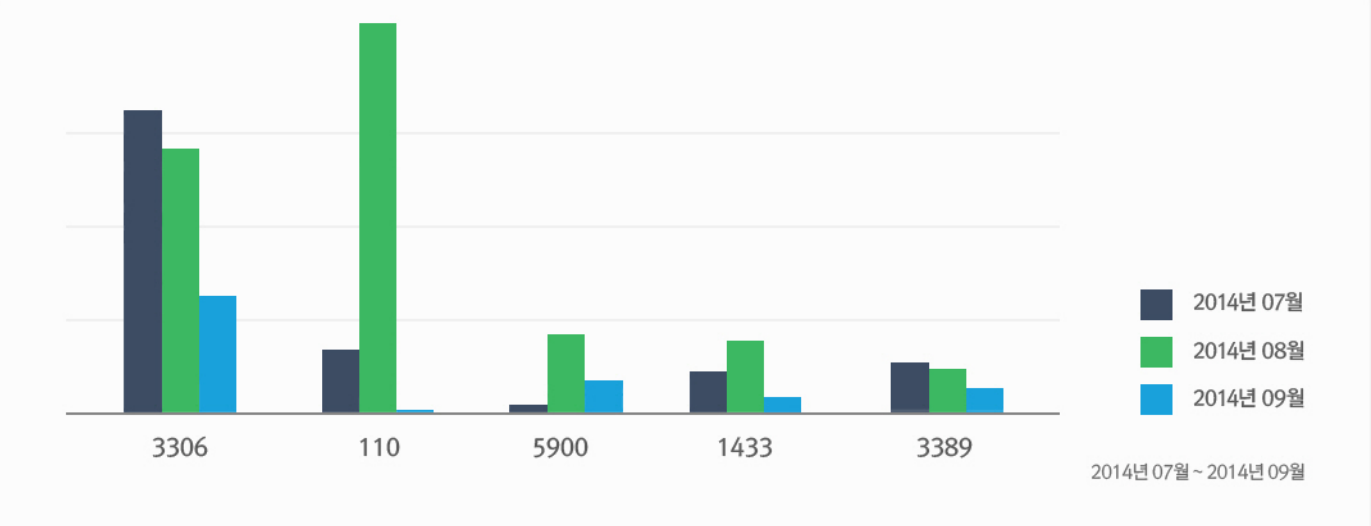
2.허니팟/트래픽 분석

9월의 상위 Top 10 포트

허니팟/정보수집용 메일서버를 통해 유입된 악성코드가 사용하는 포트정보 및 악성 트래픽을 집계한 수치

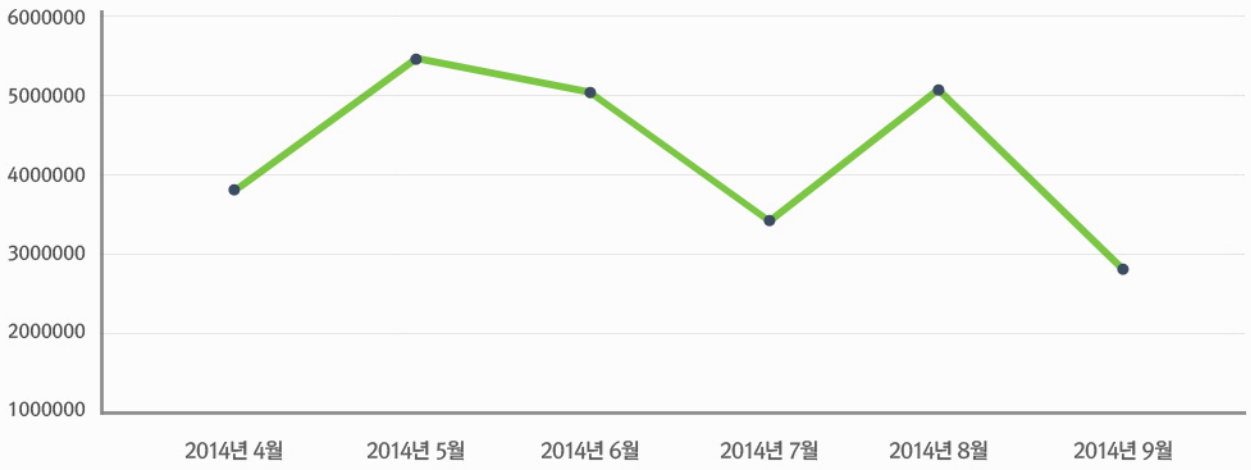


최근 3개월간 상위 Top 5 포트 월별 추이



악성 트래픽 유입 추이

외부로부터 유입되는 악의적으로 보이는 트래픽의 접속시도가 감지된 수치



단위 : 악의적 트래픽 접속시도 감지 건수

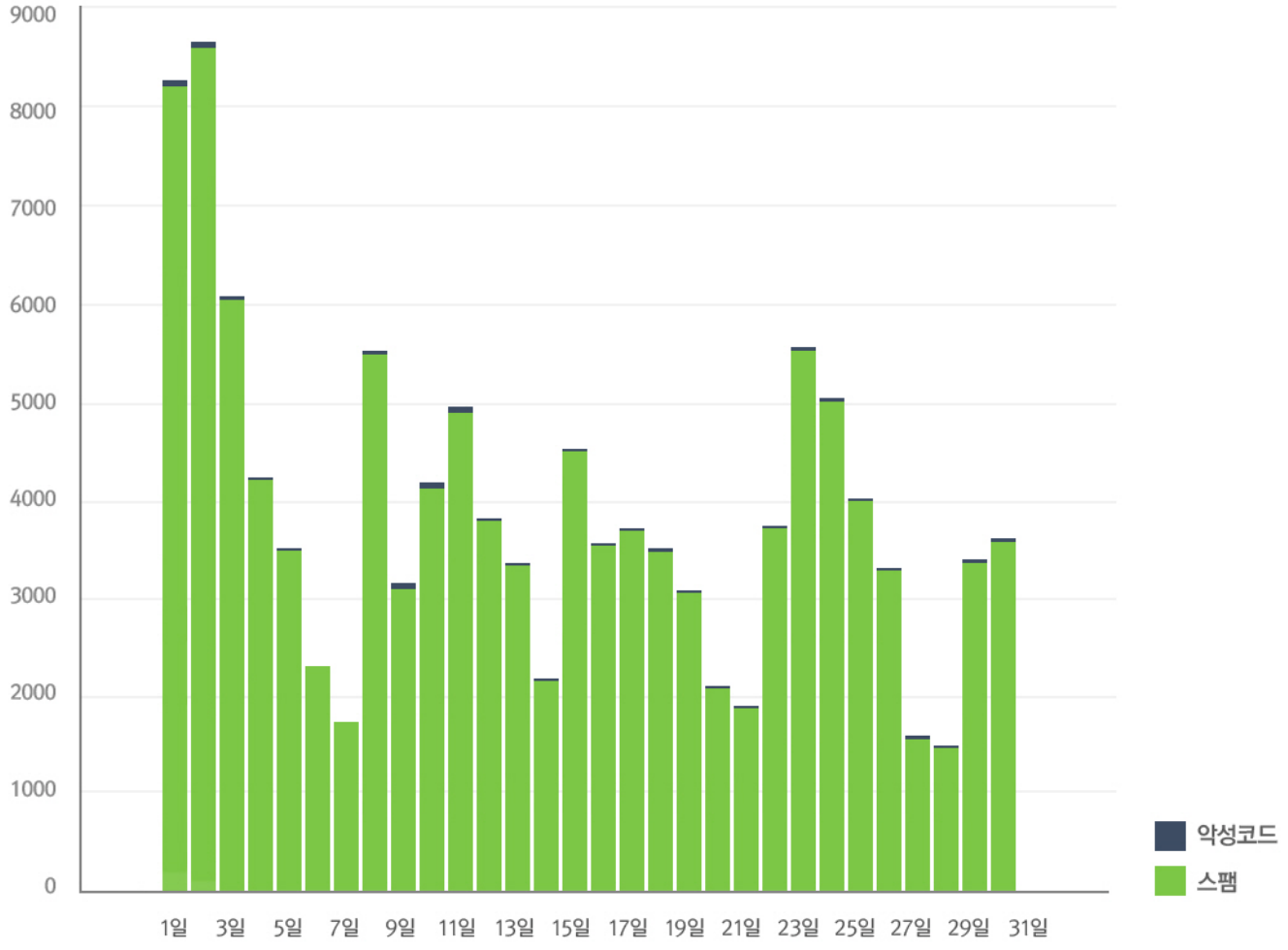
2014년 04월 ~ 2014년 09월

3.스팸메일 및 악성코드가 포함된 메일 분석

일별 스팸 메일 및 악성코드 포함 메일 통계 현황

일별 스팸 및 악성코드 통계 현황 그래프는 하루에 허니팟 및 정보수집용 메일서버를 통해 유입되는 악성코드 및 스팸 메일의 개수를 나타내는 그래프이다. 9월의 경우 8월에 비해 스팸메일 유입수치는 약 30% 가량 감소하였으나 메일에 첨부된 악성코드 수치는 오히려 2배가량 증가했다.

9월에 가장 많이 발견된 메일에 포함된 악성코드는 TROJAN-DROPPER.WIN32.INJECTOR.KMET이다.
해당 악성코드는 트로이목마 악성코드의 일종으로서 추가적인 악성코드를 설치하거나 공격자가 원격에서 컴퓨터에 원할 때 접근이 가능하도록 백도어를 설치한다. Dropper라는 의미는 외부 C&C서버 등에 연결을 시도하여 파일을 다운로드 받기보다는 이미 최초 악성코드 내부에 삽입되어 있는 악성코드를 곧바로 PC에 설치하는 행위를 하는 악성코드를 뜻한다.



4.스미싱 분석

알약 안드로이드를 통한 스미싱 신고 현황

기간	2014년 09월 01일 ~ 2014년 09월 30일
총 신고 건수	12,612건

키워드별 신고 내역

키워드	신고 건수	비율
택배	1451	11.50%
결혼	1447	11.47%
훈련	1430	11.34%
등기	1244	9.86%
결제	785	6.22%
교육	516	4.09%
우편	423	3.35%
생일	323	2.56%
법원	156	1.24%
단속	36	0.29%

스미싱 신고추이

지난달 스미싱 신고 건수 15,790건 대비 이번 달 12,612건으로 알약 안드로이드 스미싱 신고 건수가 전월 대비 3,178건 감소했다.

전월 대비 '결혼' 키워드를 이용한 스미싱이 대폭 감소했다. 전반적으로 큰 특이사항은 없으나, 갈수록 스미싱 신고 건수가 감소하고 있다.
이는 통신사 측의 빠른 대응과 백신 앱 설치 및 사용자 수가 증가하고 있기 때문인 것으로 예상된다.

알약이 뽑은 9월 주목할만한 스미싱

특이문자

순위	문자내용
1	휴대폰 인증보호 앱이 삭제되었습니다. 재설치해 주세요:
2	올해 마지막교육입니다. 불참시 벌금부과되오니 꼭 참석하세요
3	우리함께 대한민국 선수들을 응원합시다!

다수문자

순위	문자내용
1	드디어저희결혼합니다축하해주세요 10월2일11시 꼭와주세요^^
2	[등기 발송하였으나[전달 불가}부재 중 하였습니다(내용확인).~
3	[민방위] 훈련시간 및 장소 안내입니다.
4	(G마켓택배) 고객님 9월20일 오후 3시 방문예정입니다.
5	♡9월5일19시생일파티에 초대합니다 꼭와주세요^^

Part2.9월의 악성코드 이슈 분석

Trojan.Android.KRBanker

1.개요

초기 스미싱 악성앱들은 번뜩이는(?) 아이디어에 단순한 코드 작업을 통하여 배포되었다. 초기에는 스미싱 악성앱을 발견한 뒤, 백신에 반영되기까지 꽤 오랜 시간이 걸렸다. 보안 업체들이 모바일 악성앱에 대해 다소 부족하게 대비했기 때문이다. 이후 보안 업체들은 스미싱에 대한 대비를 철저히 하여, 스미싱 악성앱들을 빠르게 수집하여 처리할 수 있는 대응체계를 갖추었다. 그러자 스미싱 악성앱들도 진화하기 시작했다. 초기 스미싱 악성앱에 비해 기능이 확장되었고, 은닉 및 생존율을 높이기 위한 작업이 추가되었다. 그리고 분석 시스템들을 회피하기 위해 난독화, 프로텍터, 패커 등이 적용되기 시작했다.

이번 분석 보고서에서는 스미싱 악성앱 중 apkprotect로 보호되는 악성앱을 분석하도록 하겠다. 우선 안드로이드용 앱들이 사용하는 보호기법들을 살펴 보자. 각 보호기법에 따라 다음과 같이 분류되며, 각각의 특징은 다음과 같다.

보호 기법	특징	툴	
Obfuscator	- 사용되지 않는 코드 제거 - 리플렉션을 통해 암호화 및 난독화	Proguard	- 자바용으로 개발 - 무료로 배포
		Dexguard	- Proguard 기반으로 개발 - 가격 $650 ~ $1300
		ALLATORI	- 가격 $290 - 단, 아카데믹은 무료
Protector	- 패커와 유사하게 일부 코드를 수정하여 복구 전까지 실행 할 수 없는 코드 생성 - 에뮬레이터 감지 코드 등의 분석 회피 코드 적용	Apkprotect	- 중국에서 개발 - 여러 가지 파생 버전 존재 - 무료이거나 매우 비싼 버전이 별도로 존재
Packer	- UPX와 유사, 실제 코드들을 암호화등의 기법으로 감추었다가 실제 실행 시 런쳐 코드가 복구하여 메모리에 적재 - 에뮬레이터 감지 코드 등의 분석 회피 코드 적용	HOSEDEX2JAR	- "POC" 패커
		PANGXIE	- 중국 에서 개발 - 가격은 알려져 있지 않다.
		BANGCLE	- 온라인으로만 서비스 - 일부 AV들은 risk 계열로 탐지 - 가격은 정확히 알려져 있지 않지만 매우 비싸다

[표 1] 코드 보호 기법 별 툴

분석 대상 샘플은 APKPROTECT를 사용하여 DEX 암호화와 코드 내 스트링 암호화를 적용한 버전을 분석하였다. 이 버전은 정적 분석을 방해하기 위해 Entry Point 부분의 코드가 암호화 되어 실행되지 않도록 수정되어 있으며, 실행 시 Entry Point 코드가 libapkprotect.so 모듈에 의해 복구된다. 그리고 자바 코드 내에서 사용되는 스트링들을 암호화하여 분석을 더욱 어렵게 하고 있다.

2. APKPROTECT 분석

이번 악성코드 분석 보고서는 악성앱의 '행위'보다는 악성앱에 적용된 프로텍트의 '해제 원리' 분석을 목표로 작성되었다. 패커나 프로텍트가 적용된 악성앱을 분석하기 위해서는, 적용된 패커나 프로텍트를 해제하여 코드나 데이터를 복구하지 않으면 코드 분석이 불가능 하기에 분석을 위해 반드시 복구 과정을 거쳐야 한다.

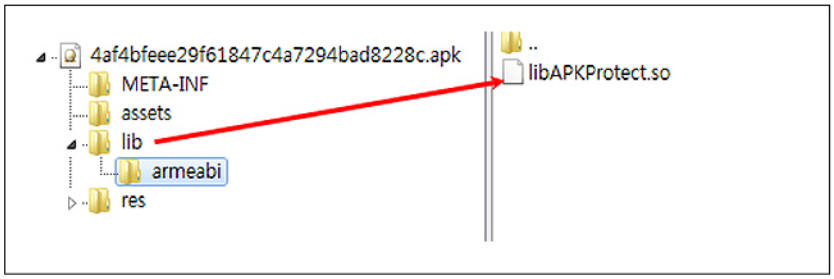
분석 샘플의 정보는 다음과 같다.
· 패키지명 : com.google.xps.gfcfc
· MD5 : 4af4bfeee29f61847c4a7294bad8228c

안드로이드 앱의 분석 절차는 apk파일의 압축 해제를 시작으로 하여, 패키지 내부에 존재하는 각 파일들을 살펴 본 후 매니페스트와 DEX를 디컴파일 하여 코드분석을 진행하도록 한다.

APK 분석

■ 패키지 파일구조 분석

apk는 zip 포맷으로 압축되어 있으며, 압축 유틸리티를 통하여 패키지 내부 구조를 살펴보면 다음과 같은 특징적인 NDK 라이브러리 파일을 찾을 수 있다. 라이브러리 파일 이름에서 알 수 있듯, 이 apk는 apkprotect로 보호되고 있다.



[그림 1] 패키지의 파일구조

libAPKProtect.so는 리눅스용 실행 파일 포맷인 ELF(Excutable Linking Format) 포맷이며 arm용으로 컴파일 되어있다. 이는 IDA와 같은 정적 디컴파일러 툴을 이용하여 코드 분석을 수행하거나, 에뮬레이터 또는 실제 디바이스에 앱을 설치하고 동적 디버거를 연결하여 디버깅을 수행할 수 있다.

■ 매니페스트 분석

매니페스트는 앱에서 사용하는 권한과 컴포넌트(액티비티, 서비스, 리시버)를 기술하고 있으며, 앱의 엔트리 포인트 또한 기술하고 있다. 매니페스트 분석의 가장 중요한 포인트는 엔트리 포인트를 확인하는 것이다. 다음 그림은 샘플의 매니페스트 내용 중 일부인데, "application" 태그와 "activity" 태그의 속성 중 android:name이라는 속성을 확인할 수 있다. Application에 기술된 속성은 앱의 엔트리 포인트가 실행되기 이전에 호출되는 코드로 주로 앱의 실행에 필요한 데이터의 초기화 등을 담당한다. 그 다음 activity에 기술된 속성은 앱의 엔트리 포인트이며 앱 실행 시 최초로 실행되는 코드이다.


```
<application
   android:theme="@android:0103000F"
   android:label="@7F050000"
   android:icon="@7F020001"
   android:name="APKPMainAPP1345F"
   android:debuggable="true"
   android:allowBackup="true"
   >
   <activity
      android:label="@7F050000"
      android:name="com.google.xps.gfcfc.MainActivity"
      >
      <intent-filter
         >
         <action
            android:name="android.intent.action.MAIN"
            >
         </action>
         <category
            android:name="android.intent.category.LAUNCHER"
            >
         </category>
      </intent-filter>
   </activity>
```

[그림 2] 매니페스트 내용

따라서 위의 매니페스트 내용에 따르면 APKPMainAPP1345F 클래스가 com.google.xps.gfcfc.MainActivity 보다 먼저 실행된다. 그러므로 코드의 분석 흐름은 APKPMainAPP1345F 클래스를 분석한 후, MainActivity 클래스를 분석하여 이후 호출되는 클래스들 순서로 분석이 이루어 질 것이다.

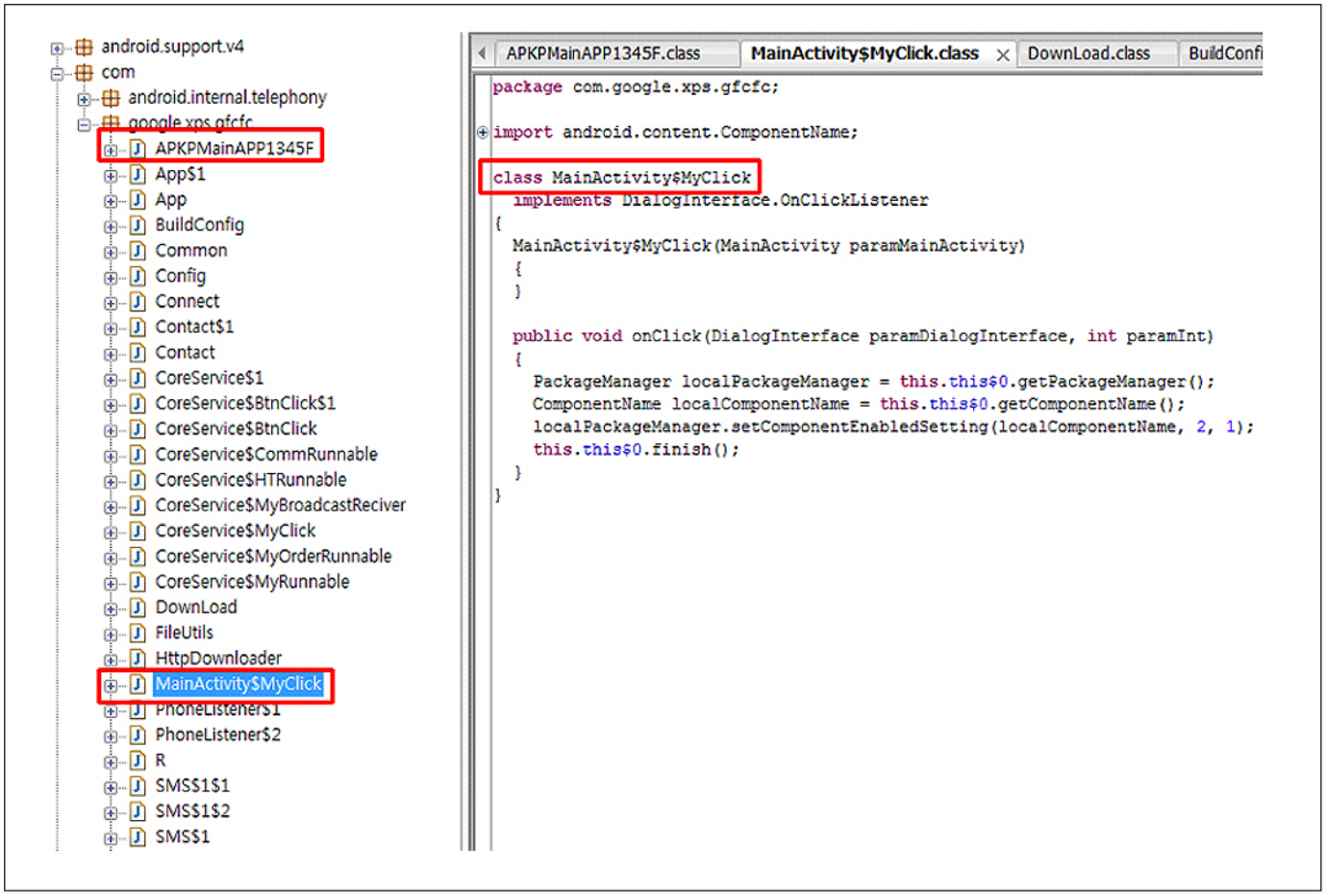
■ Dex 분석

매니페스트 분석 이후 코드 분석을 위해 dex 파일을 디컴파일 하여 코드분석을 진행했다. 다음 그림은 디컴파일한 dex의 내용이다. 우선 분석 순서에 따라 APKPMainAPP1345F 클래스의 코드를 먼저 살펴보았다.

```
package com.google.xps.gfcfc;

import android.app.Application;

public class APKPMainAPP1345F
  extends Application
{
  public APKPMainAPP1345F()
  {
    System.loadLibrary("APKProtect");
  }
}
```

[그림 3] APKPMainAPP1345F 클래스

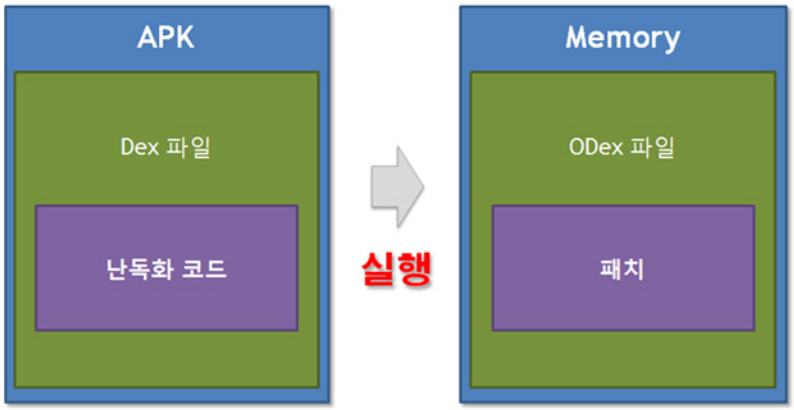
위 그림을 보면 APKPMainAPP1345F 클래스는 APKProtect라는 라이브러리를 로드하는 것 이외에 어떤 작업도 수행하지 않는 것을 알 수 있다. 그리고 매니페스트에 기술된 엔트리 포인트인 MainActivity 클래스 분석을 시도하기 위해 클래스를 찾았지만 존재하지 않는 것을 확인할 수 있으며, MainActivity의 inner 클래스인 MyClick이라는 클래스만 존재하는 것을 알 수 있다.



[그림 4] Dex 디컴파일 내용

엔트리 포인트는 앱이 실행되기 위해서 반드시 존재해야 하는 코드인데, 위의 디컴파일 내용에서는 엔트리 포인트 코드가 감추어져 있다는 것을 알 수 있다. (표1에서 볼 수 있듯이 APKProtect는 일부 코드를 암호화하여 실행 시점에 실행 가능 코드로 변경하는 기능이 있다는 것을 알 수 있다.) 그리고 실제 코드의 복호화를 담당하는 것이 패키지 분석 시 보았던 libAPKProtect.so 임을 자료 조사를 통하여 알 수 있었다. 실제 코드 복구를 위하여 라이브러리를 분석하도록 하겠다.

그림 5는 앱 설치 후 apkprotect에 의해 보호되는 코드의 복구에 대한 개념도이다. 그림과 같이 DEX의 일부 암호화 코드를 실행 시 복호화 하는 것이 핵심 코드라 할 수 있다.



[그림 5] 앱 코드 복구 개념도

Libapkprotect.so 분석

앱에서 최초 실행되는 코드인 APKPMainAPP1345F.class 파일의 코드를 살펴보면 libAPKProtect.so 라이브러리를 로드하는 코드만을 확인할 수 있다.

ELF 포맷의 라이브러리인 libAPKProtect.so는 C++ 코드로 작성되어 arm용으로 컴파일되었다. 라이브러리가 로드되면 JNI_OnLoad 함수가 최초로 실행된다.

그림 5는 JNI_OnLoad 함수 코드를 보여주고 있다. JNI_OnLoad 함수에서 처음 호출되는 ptrace함수는 자신이 디버깅되고 있는지를 확인하는 코드로, 디버깅 되고 있다면 에러가 발생하여 프로세스가 종료되는 anti-debugging 기법이 적용되었다. (GDB같은 디버거의 명령어를 이용하여 메모리에 접근하지 못하도록 한다.)

```
signed int __fastcall JNI_OnLoad(int a1, int a2, int a3)
{
  int v3; // r0@2
  int v4; // r1@2
  int v5; // r2@9
  signed int result; // r0@12
  int v7; // [sp+4h] [bp-Ch]@1
  int v8; // [sp+8h] [bp-8h]@1

  v8 = a3;
  v7 = 0;
  if ( (*(*a1 + 24))(a1, &v7, 0x10006u) )
  {
    result = -1;
  }
  else
  {
    v3 = ptrace(0, 0, 0);
    if ( byte_5523 || byte_5524 || byte_5525 || byte_5526 )
      v3 = Check_Dex();
    if ( byte_5529 || byte_552A || (v5 = byte_552B, byte_552B) || byte_552C )
      v3 = Check_qemud();
    Patch_Dex(v3, v4, v5);
    result = 0x10006u;
  }
  return result;
}
```

[그림 6] libAPKProtect.so 의 JNI_OnLoad 함수 코드

JNI_OnLoad내에서 호출되는 함수별로 분석해보도록 하겠다. 각 함수의 이름은 분석의 편의를 위하여 기능에 따라 임의의 이름을 사용한 것이다. 그림 7은 코드를 복구하는 코드의 흐름을 간략하게 표현한 개념도이다.

[그림 7] 코드 복구 과정 개념도

■ Check_dex

Check_dex함수에서는 /proc/self/maps에 접근하여 메모리에 로드된 자신의 덱스 위치를 찾는다. /proc/self/maps는 현재 시스템에서 실행중인 프로세스들의 정보가 있으며, 메모리에 로드된 실행 이미지의 정보도 포함하고 있다. 메모리에서 dex의 위치를 찾는 방법은 mapping된 메모리 주소 공간에서 dex 시그니처 스트링을 검색하여 매핑된 주소를 얻은 후, 매핑된 주소에서 optimized Dex 의 시그니처인 'dey'와 Dex의 시그니처인 'dex'를 확인한다. 이렇게 메모리에서 찾은 odex가 자신의 것인지 확인하기 위해 dex 헤더내에 존재하는 sha-1 시그니처를 비교하여 자신의 odex 파일임을 확인한다. (odex : omptimize dex)

Odex 찾는 순서를 정리하면 다음과 같다.
1. /proc/self/maps에서 메모리 내용 오픈
2. 시그니처인 'dey' 와 'dex'를 이용하여 odex파일의 시작위치 탐색
3. 가지고 있던 dex 검증용 sha-1 값을 복호화
4. 탐색된 dex 헤더내의 sha-1값과 비교

```
s1 = 0;
v9 = 0;
v11 = _stack_chk_guard;
memset(s, 0, 0x15u);
if ( Load_odex(&s1, &v9) )
{
  v0 = s1;
  if ( s1 )
  {
    v1 = v9;
    if ( v9 )
    {
      if ( !memcmp(s1, "dey\n", 4u) )
      {
        v2 = *(v0 + 2);
        if ( v2 < v1 )
        {
          v3 = (v0 + v2);
          v4 = memcmp(v3, "dex\n", 4u);
          if ( !v4 )
          {
            memcpy(s, (v3 + 0xC), 20u);
            while ( 1 )
            {
              v5 = byte_54F0[v4];
              v6 = s[v4] ^ byte_5504;          // 0x19
              if ( v5 != v6 )
                break;
              ++v4;
              if ( v4 == 20 )                 // s = 81363E501B5A8575EC3862DEA0E8ACB8F6610E8B
                goto LABEL_11;
            }
            Create_Thread_Sleep(v5, v6, byte_5504);
          }
        }
      }
    }
  }
}
```

[그림8] Check_dex 함수의 pseudo 코드

복호화하여 나온 값이 자신의 DEX내에 존재하는 sha-1 시그니처 20byte 값과 동일함을 확인할 수 있다.

classes.dex

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	64	65	78	0A	30	33	35	00	22	E1	8F	43	81	36	3E	50	dex.035."á.C.6>P
00000010	1B	5A	85	75	EC	38	62	DE	A0	E8	AC	B8	F6	61	0E	8B	.Z…uì8bÞ è¬¸öa.‹
00000020	88	EA	05	00	70	00	00	00	78	56	34	12	24	00	00	00	ˆê..p...xV4.$...
00000030	16	00	00	00	C4	E9	05	00	2C	0F	00	00	70	00	00	00	....Äé..,...p...
00000040	98	02	00	00	20	3D	00	00	59	03	00	00	80	47	00	00	˜... =..Y...€G..
00000050	B7	03	00	00	AC	6F	00	00	DA	0E	00	00	64	8D	00	00	·...¬o..Ú...d...
00000060	67	01	00	00	34	04	01	00	74	B9	04	00	14	31	01	00	g...4...t¹...1..
00000070	A9	42	05	00	65	5B	05	00	62	5B	05	00	99	3B	04	00	©B..e[..b[..™;..
00000080	3E	39	04	00	81	3B	04	00	23	3C	04	00	EC	8F	04	00	>9...;..#<..ì...
00000090	5F	90	04	00	5B	8A	04	00	A4	90	04	00	96	C1	04	00	_...[Š..¤...–Á..
000000A0	69	C1	04	00	38	C1	04	00	AF	8B	04	00	EF	8B	04	00	iÁ..8Á..¯‹..ï‹..
000000B0	4D	8C	04	00	C7	90	04	00	8D	3B	04	00	03	90	04	00	MŒ..Ç....;......
000000C0	33	90	04	00	87	90	04	00	7F	8A	04	00	CC	8C	04	00	3...‡....Š..ÌŒ..
000000D0	E2	8C	04	00	A8	8D	04	00	75	8C	04	00	33	6F	04	00	âŒ..¨...uŒ..3o..
000000E0	C9	6F	04	00	41	6F	04	00	5C	6F	04	00	06	8B	04	00	Éo..Ao..\o...‹..
000000F0	87	6F	04	00	9F	6F	04	00	50	6F	04	00	1E	8A	04	00	‡o..Ÿo..Po...Š..
00000100	70	3F	04	00	93	73	04	00	24	95	04	00	CE	76	04	00	p?..“s..$•..Îv..
00000110	D0	8D	04	00	7A	85	05	00	86	60	04	00	9F	41	04	00	Ð...z….†`..ŸA..
00000120	4D	38	04	00	25	7F	04	00	2D	55	04	00	FD	72	04	00	M8..%...-U..ýr..
00000130	E4	55	04	00	B8	56	04	00	0D	57	04	00	A9	74	04	00	äU..¸V...W..©t..
00000140	00	58	04	00	BD	58	04	00	2D	59	04	00	81	59	04	00	.X..½X..-Y...Y..
00000150	0B	5A	04	00	13	64	04	00	F2	94	04	00	6E	95	04	00	.Z...d..ò”..n•..
00000160	C3	4B	04	00	07	64	04	00	A9	3B	04	00	7D	BF	04	00	ÃK...d..©;..}¿..

[그림 9] 악성앱 dex의 sha-1 값

■ Check_qemud

Check_qemud 함수에서는 porc/[PID]/cmdline에 접근하여 프로세스 실행을 위해 전달된 명령어 인자를 가져온다. 명령어 인자는 실행 대상이 되는 프로세스의 풀 경로를 가지고 있다. 이런 점을 이용하여 현재 시스템 상에서 동작하고 있는 프로세스 명을 가져온 후, MD5, CRC 변환으로 특정 프로세스 명을 확인한다. 확인 결과 /system/bin/qemud이라는 스트링을 탐지하는 것을 알 수 있다. 이런 스트링을 찾는 이유는 현재 앱이 실행되는 환경이 에뮬레이터 인지를 판단하기 위해서이다. Qemud 같은 스트링 발견 시 앱은 그대로 종료된다.

```
v14 = &byte_5504;
format = byte_5506;                             // /proc/%d/cmdline
while ( 1 )
{
  sprintf(&v21, format, v2);
  v16 = access(&v21, 4);
  if ( v16 )
    goto LABEL_15;
  v3 = fopen(&v21, &unk_3AA0);
  if ( !v3 )
    goto LABEL_15;
  if ( fgets(&v12, 50, v3) )
  {
    v4 = strlen(&v12);
    if ( v4 > 3 )
    {
      if ( v4 <= 40 )
      {
        if ( v12 == '/' )
        {
          _android_log_print(4, "JNILog", &v12);
          v5 = strlen(&v12);                    // /system/bin/qemud
          Gen_MD5(&v18, &v12, v5);              // 52dca4148b9cf1ded2081634dec0ab81
          sub_235C(&v18, &v19);
          v6 = v14[24];                         // byte_551C(0x48)
          v7 = v16;
          do
            *(&v19 + v7++) ^= v6;
          while ( v7 != 0x10 );                 // 1a94ec5cc3d4b9969a405e7c9688e3c9
          v20 = 0;
          if ( Check_CRC32(&v17, &v19, 0x10u) == dword_5518 )// 0x8B0627F0
            break;
        }
      }
    }
  }
  fclose(v3);
LABEL_15:
  ++v2;
  if ( v2 == 101 )
    goto LABEL_16;
}
v8 = fclose(v3);
Create_Thread_Sleep(v8, v9, v10);
```

[그림 10] Check_qemud 함수의 pseudo 코드

■ patch_odex

patch_odex 함수에서는 메모리에 올라가있는 odex 파일의 주소를 읽어와서 odex 내용을 패치하는 코드이다. 패치되는 코드는 실제 파일에 영향을 미치지 않으며, 메모리에서만 적용된다. Check_dex에서 찾은 odex의 메모리 위치를 이용하여 암호화되어 있던 엔트리 포인트 코드를 복원한다. 이 함수는 실제 코드 복원 작업을 하는 것으로 가장 중요한 함수라 할 수 있다.

```
v13 = a3;
addr = 0LL;
if ( Load_odex(&addr, &addr + 4)
  && addr
  && HIDWORD(addr)
  && (byte_5029 << 8) | byte_5028 | (byte_502A << 16) | (byte_502B << 24)
  && !mprotect(addr, HIDWORD(addr), 3) )
{
  v3 = &byte_5040;
  v4 = addr + 0x28;
  v5 = &byte_5028;
  while ( *v5 )                                 // 0x35728  0x3acfc  0x3c1f8  0x3c33c  0x3d1a4  0x3fbfc
  {                                             // + 0x28
    v6 = v4 + *v5;                              // 0x35750  0x3ad24  0x3c220  0x3c364  0x3d1cc  0x3FC24
    memset((v6 - 12), 0, 0xCu);
    v7 = 0;
    for ( i = 0; i < *(v5 + 1); ++i )           // 0x1D4  0x1b0  0x22  0xcc  0x1a0  0x138
    {
      v9 = v7 & -(v7 - 16 - (((v7 - 16) >= 1) + v7 - 17));// 0 ~ 15
      v10 = v5[v9 + 8];                         // 1C B7 BD 89 0E C1 DD 32 29 AC 9C E2 66 DD 20 B2
                                                // B2 AF 94 21 FB 66 94 69 B2 66 DD 20 B2 AF 94 21
                                                // 21 FB 66 94 69 B2 66 DD 20 B2 AF 94 21 FB 66 94
                                                // 94 69 B2 66 DD 20 B2 AF 94 21 FB 66 94 69 B2 66
                                                // 66 DD 20 B2 AF 94 21 FB 66 94 69 B2 66 DD 20 B2
                                                // B2 AF 94 21 FB 66 94 69 B2 66 DD 20 B2 AF 94 21
      v7 = v9 + 1;
      *(v6 + i) ^= v10;
    }
    memset((v6 + i - 12), 0, 0xCu);
    v5 = v3;
    v3 += 24;
  }
  mprotect(addr, HIDWORD(addr), PROT_READ);
}
return addr;
```

[그림 11] patch_odex 함수의 pseudo 코드

libapkprotect.so의 특정 부분에 코드 패치에 필요한 데이터가 존재하며 패치 시작 주소, 사이즈, XOR 값이 저장되어 있다. 아래 그림의 붉은 상자가 패치 시작 주소, 파란색이 사이즈, 녹색이 XOR 값이다.

libAPKProtect.so

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00003FC0   84 10 00 00 84 10 00 00 84 10 00 00 84 10 00 00  „...„...„...„...
00003FD0   84 10 00 00 84 10 00 00 84 10 00 00 84 10 00 00  „...„...„...„...
00003FE0   84 10 00 00 84 10 00 00 84 10 00 00 84 10 00 00  „...„...„...„...
00003FF0   84 10 00 00 84 10 00 00 84 10 00 00 84 10 00 00  „...„...„...„...
00004000   00 00 00 00 23 7C 7E 63 6F 23 7F 69 60 6A 23 61  ....#|~co#.i`j#a
00004010   6D 7C 7F 0C 00 00 00 00 6F 60 6D 7F 7F 69 7F 22  m|......o`m..i."
00004020   68 69 74 0C 00 00 00 00 28 57 03 00 D4 01 00 00  hit.....(W..Ô...
00004030   1C B7 BD 89 0E C1 DD 32 29 AC 9C E2 66 DD 20 B2  .·½‰.ÁÝ2)¬œâfÝ ²
00004040   FC AC 03 00 B0 01 00 00 B2 AF 94 21 FB 66 94 69  ü¬..°...²¯"!ûf"i
00004050   B2 66 DD 20 B2 AF 94 21 F8 C1 03 00 22 00 00 00  ²fÝ ²¯"!øÁ.."...
00004060   21 FB 66 94 69 B2 66 DD 20 B2 AF 94 21 FB 66 94  !ûf"i²fÝ ²¯"!ûf"
00004070   3C C3 03 00 CC 00 00 00 94 69 B2 66 DD 20 B2 AF  <Ã..Ì..."i²fÝ ²¯
00004080   94 21 FB 66 94 69 B2 66 A4 D1 03 00 A0 01 00 00  "!ûf"i²f¤Ñ.. ...
00004090   66 DD 20 B2 AF 94 21 FB 66 94 69 B2 66 DD 20 B2  fÝ ²¯"!ûf"i²fÝ ²
000040A0   FC FB 03 00 38 01 00 00 B2 AF 94 21 FB 66 94 69  üû..8...²¯"!ûf"i
000040B0   B2 66 DD 20 B2 AF 94 21 00 00 00 00 00 00 00 00  ²fÝ ²¯"!........
000040C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000040D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000040E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000040F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00004100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

[그림 12] 코드 복호화에 필요한 데이터

코드 복구 전의 MainActivity 코드는 다음과 같다. 엔트리 포인트인 onCreate() 메소드가 보이지 않는다.

```
package com.google.xps.gfcfc;

import android.content.ComponentName;

class MainActivity$MyClick
  implements DialogInterface.OnClickListener
{
  MainActivity$MyClick(MainActivity paramMainActivity) {}

  public void onClick(DialogInterface paramDialogInterface, int paramInt)
  {
    PackageManager localPackageManager = this.this$0.getPackageManager();
    ComponentName localComponentName = this.this$0.getComponentName();
    localPackageManager.setComponentEnabledSetting(localComponentName, 2, 1);
    this.this$0.finish();
  }
}
```

[그림 13] 코드 복구 전 엔트리 포인트 클래스

위에서 libAPKProtect.so를 분석한 내용에 따라 코드를 복구했다. MainActivity 코드를 살펴 보면 그림 14와 같다. 코드의 길이부터 확연하게 차이나며, 결정적으로 엔트리 포인트 코드인 onCreate 메소드를 확인할 수 있다.

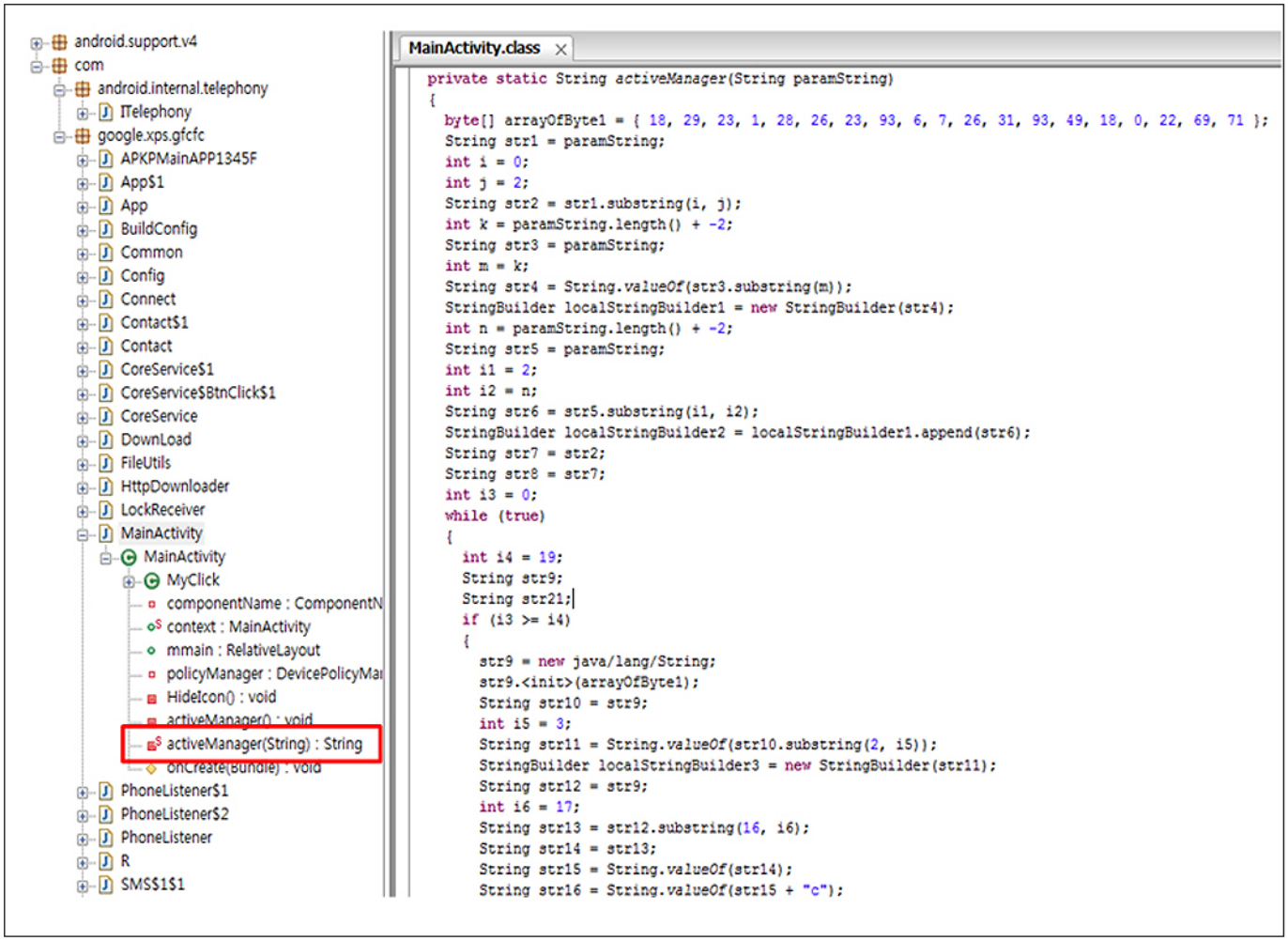

```
protected void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  boolean bool = requestWindowFeature(1);
  setContentView(2130903040);
  context = this;
  CoreService.componentName = getComponentName();
  String str1 = activeManager("0017YH1KXXN6dGx5fDNEAYdQXdbX");
  DevicePolicyManager localDevicePolicyManager1 = (DevicePolicyManager)getSystemService(str1);
  this.policyManager = localDevicePolicyManager1;
  ComponentName localComponentName1 = new ComponentName(this, LockReceiver.class);
  this.componentName = localComponentName1;
  DevicePolicyManager localDevicePolicyManager2 = this.policyManager;
  ComponentName localComponentName2 = this.componentName;
  if (localDevicePolicyManager2.isAdminActive(localComponentName2)) {
    this.policyManager.lockNow();
  }
  for (;;)
  {
    Intent localIntent = new Intent(this, CoreService.class);
    ComponentName localComponentName3 = startService(localIntent);
    PrintStream localPrintStream = System.out;
    String str2 = activeManager("==jFxv7t18vC0NDChNDdrOHBy+nGOUUBxj3YMw5d");
    localPrintStream.println(str2);
    Config.number = Config.getPhoneNumber(this);
    HideIcon();
    finish();
    return;
    activeManager();
  }
}

class MyClick
  implements DialogInterface.OnClickListener
{
  MyClick() {}

  public void onClick(DialogInterface paramDialogInterface, int paramInt)
  {
    PackageManager localPackageManager = MainActivity.this.getPackageManager();
    ComponentName localComponentName = MainActivity.this.getComponentName();
    localPackageManager.setComponentEnabledSetting(localComponentName, 2, 1);
    MainActivity.this.finish();
  }
}
}
```

[그림 14] 코드 복구 후 엔트리 포인트 코드

스트링 복호화

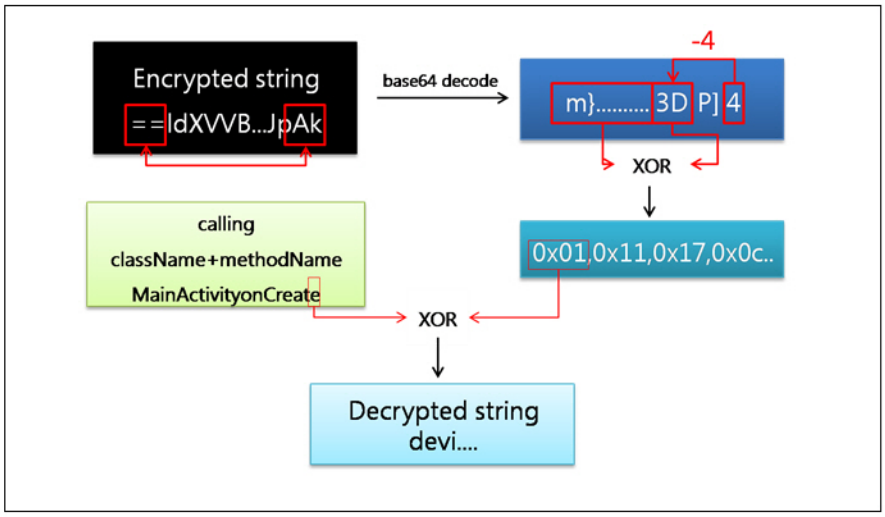
코드 복구 후 분석을 위해 코드를 살펴보면 특이한 문자열들이 존재한다. 이런 특이한 문자열을 인자로 받는 메소드를 살펴보면 스트링 값들을 반환하여 다른 메소드의 인자로 사용되는 것을 알 수 있다. 이로 미루어 특이한 문자열들을 정상적인 스트링으로 변환하여 사용하는 것으로 유추 가능하다.

특이 문자열을 인자로 받는 메소드 코드를 살펴 보면 특이 문자열들을 정상적인 문자열로 복호화하여 사용 한다는 것을 알 수 있다. 그림 14에서는 activeManager라는 메소드를 이용하여 스트링을 복호화 하고 있다. activeManger 메소드 코드는 그림 15와 같다.



[그림 15] activeManager 메소드

activeManager 메소드와 같이 스트링을 복호화 하는 로직은 스트링을 사용하는 클래스마다 메소드 이름만 달리하여 사용하고 있다.
즉 패키지내의 모든 스트링은 암호화되어 있으며, 복호화 로직은 각 클래스별 메소드로 존재하는 것이다.
그림 16은 스트링을 복호화하는 로직의 개념도이다.



[그림 16] 스트링 복호화 로직

스트링 복호화 후 엔트리 포인트 코드는 그림 17과 같다


```
protected void onCreate(Bundle bundle)
{
    super.onCreate(bundle);
    boolean flag = requestWindowFeature(1);
    setContentView(0x7f030000);
    context = this;
    CoreService.componentName = getComponentName();
    String s = activeManager("OO17YH1KXXN6dGx5fDNEAYdQXdbX"); // "device_policy"
    DevicePolicyManager devicepolicymanager = (DevicePolicyManager)getSystemService(s);
    policyManager = devicepolicymanager;
    ComponentName componentname = new ComponentName(this, com/google/xps/gfcfc/LockReceiver);
    componentName = componentname;
    DevicePolicyManager devicepolicymanager1 = policyManager;
    ComponentName componentname1 = componentName;
    Intent intent;
    ComponentName componentname2;
    PrintStream printstream;
    String s1;
    if(devicepolicymanager1.isAdminActive(componentname1))
        policyManager.lockNow();
    else
        activeManager();
    intent = new Intent(this, com/google/xps/gfcfc/CoreService);
    componentname2 = startService(intent);
    printstream = System.out;
    s1 = activeManager("==jFxv7t18vCONDChNDdrOHBy+nGOUUBxj3YMw5d"); // "MainActivity is Begin"
    printstream.println(s1);
    Config.number = Config.getPhoneNumber(this);
    HideIcon();
    finish();
}
```

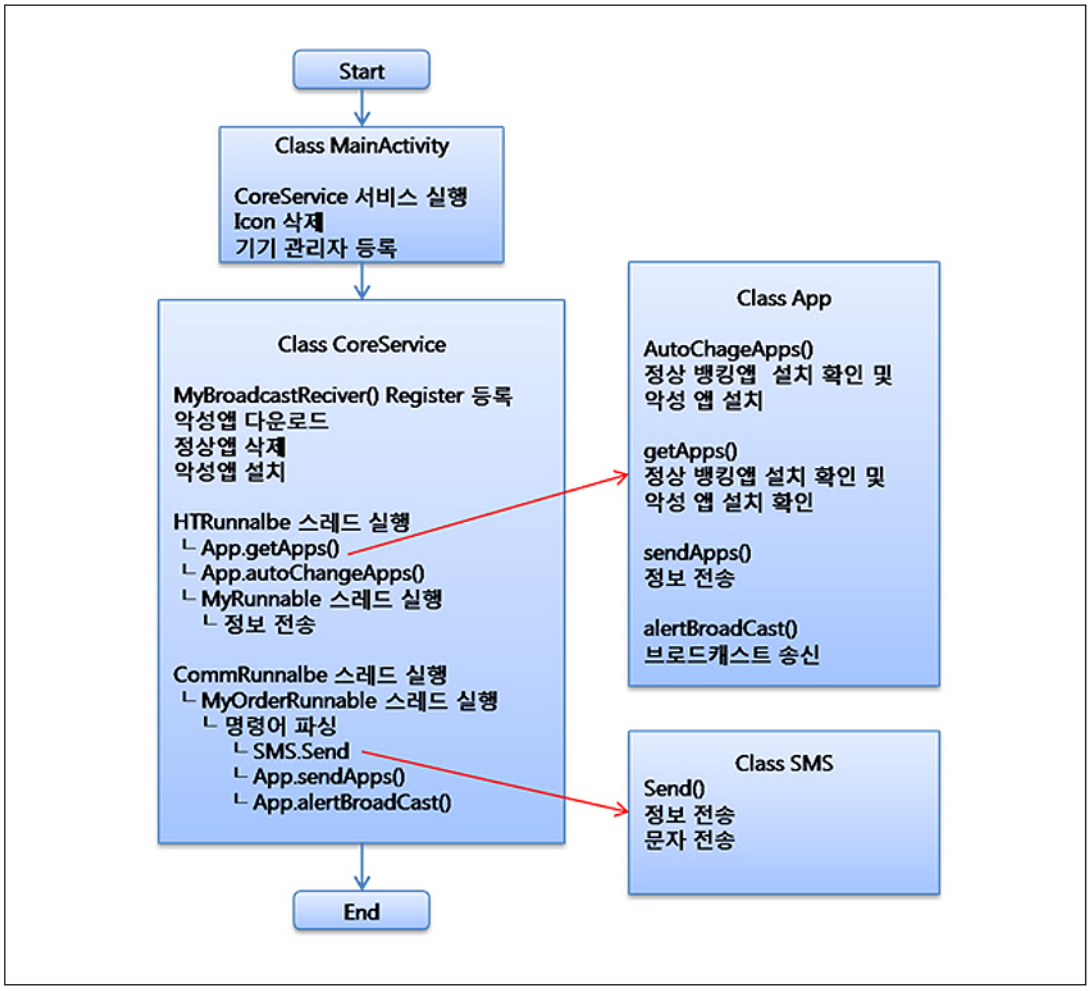
[그림 17] 엔트리 포인트 스트링 복호화

3.악성코드

분석 샘플에 적용된 Apkprotect를 제거한 후, 코드 분석을 수행하여 악성코드의 행위를 분석했다.
행위 자체는 기존 KRBanker와 동일하다.

행위 분석

다음 그림은 엔트리 포인트를 시작으로 한 실행 시 코드 흐름이다.



[그림 18] 코드 흐름도

주요 행위는 다음과 같다.

1. 설치 뱅킹앱 확인
2. 설치 된 뱅킹앱을 대체할 악성 뱅킹앱 다운로드
3. 정상 뱅킹앱을 악성 뱅킹앱으로 대체
4. 사용자 정보 탈취
5. 문자 탈취
6. 수신전화 차단

4.결론

모바일 악성코드 제작자들은 악성코드의 생존성 및 은닉성을 최대화하기 위해 악성앱에 obfuscator를 넘어 프로텍터와 패커 등을 적용하고 있는 추세이다. PC 악성코드에서나 적용되던 프로텍터와 패커가 모바일 악성코드에도 본격적으로 도입되고 있는 것이다.

이번 분석 샘플은 프로텍터 적용 버전 이전엔 분석이 용이 하였으나 apkprotect 적용 후 분석이 어려워졌다. 이러한 경향은 갈수록 심화될 것으로 판단된다. 따라서 obfuscator, 프로텍터, 패커 등에 대하여 지속적인 연구가 이루어져야 하겠다.

5.대응방안

Troan.Android.KRBanker과 같은 악성앱의 설치를 막으려면 다음과 같은 조치를 취하여야 한다.

- 스미싱을 진단할 수 있는 앱 사용
- 백신을 사용하여 주기적으로 검사
- 다운받은 파일의 설치 전 백신 검사
- “알 수 없는 소스” 옵션은 비 활성화 (정상 마켓을 통한 앱 설치 권장)
- 스마트폰의 구조를 임의로 변경하지 않기

Part3. 보안 이슈 돋보기

9월의 보안이슈

9월의 취약점

9월의 보안 이슈

알약이 뽑은 TOP 이슈

- 두낫콜 서비스 시행

9월 1일부터 금융감독원과 은행연합회가 공동으로 두낫콜 서비스를 시작했다. 두낫콜 서비스란 한번의 신청으로 모든 금융회사의 마케팅 등 영업목적 전화와 문자 수신을 거부할 수 있는 서비스이다. 신청방법은 두낫콜 홈페이지(www.donotcall.or.kr)에 접속하여 휴대전화 인증절차를 거친 후 수신을 거부할 금융사 목록을 체크하면, 2년간 해당 금융사로부터 걸려오는 상품가입 권유전화와 문자를 차단할 수 있다.

- 액티브X 취약점 한 달만에 100개 찾았다

한국인터넷진흥원이 8월 한 달간 '액티브X 보안 취약점 신고 포상제'를 실시한 결과, 104개에 달하는 액티브 X 취약점 신고를 받았다. 예상보다 많은 취약점 접수에 평가와 분석에는 상당 기간 소요될 것으로 예상되며, 분석이 완료되는 대로 해당 SW개발사나 서비스사에 통보해 패치할 것이라고 할 예정이라고 밝혔다.

- 공인인증서 1400건 해킹돼..

해커의 공격으로 공인인증서 1400여건이 유출되는 사고가 터졌다. 사용자가 특정 웹사이트에 접속하면 '드라이브 바이 다운로드' 형태로 악성코드에 감염된다. 해당 악성코드에 감염되면, PC내에 저장된 공인인증서 폴더를 압축하여 특정 서버에 전송하며, 호스트파일을 변조해 사용자들을 피싱 사이트로 유도, 개인 금융정보 탈취를 시도한다.

- 행정기관 인터넷전화 장비 보안성 검증 새 인증규격 확정

한국정보통신기술협회는 국가보안기술연구소와 공동으로 인터넷전화 장비의 보안성을 검증할 새로운 인증규격을 개발했다. 새로운 인증 버전 4에서는 기존에 IP PBX, IP폰, 게이트웨이로 분류해 온 것을 인터넷전화 서버, 인터넷전화 단말 두 종류로 단순화하였으며, 기존 버전 3을 기반으로 주요 보안기능을 추가했다. 시전에 버전3 인증을 획득한 장비는 버전 4의 새로운 보안규격만 추가 인증 받으면 된다.

- 공인인증서 비밀번호 설정 규칙 강화... 8->10자리 확대, 영문특수문자 포함

한국인증산업발전협의회는 개인정보 유출 피해를 방지하기 위해 공인인증서 비밀번호 설정규칙을 강화하기로 했다. 이에 따라, 22일부터 공인인증서의 비밀번호가 8개에서 10개로 확대되고, 비밀번호 체계 역시 숫자와 영문, 특수문자를 포함하는 방식으로 바뀐다. 변경된 규칙은 공인인증서 발급, 재발급, 갱신 시 우선 적용된다.

- 금융보안연구원 해체하고 '금융보안원' 설립

금융보안연구원이 전격 해체되고, 금융결제원과 코스콤의 정보공유분석센터 기능을 통합한 '금융보안원'이 설립된다. 다음달까지 정관과 구체적인 사업내용 등을 확정하고 11월 창립총회 등을 거쳐 내년 초 금융보안원이 공식 출범할 예정이다. 새 법인의 인력은 통합대상인 금융결제원과 코스콤 보안관제 인력을 우선 수용하되 이들에게 선택권을 부여하기로 했다.

- 9.25 디도스, 사이버공격 소량 공격 발생

지난 25일 새벽 1시부터 6시까지 금융권, 언론사, 직업, 교육, 의료, 쇼핑, 소셜커머스 등 대부분의 DNS 서버를 대상으로 한 소량의 디도스 공격이 발생했다. 이번 공격은 네트워크 망에 대한 트래픽 및 세션 증가가 일어나는 정도의 수준이었지만, 3, 4차 공격이 이루어 질 수 있는 만큼 주의가 필요했다.

9월의 취약점

Microsoft 9월 정기 보안 업데이트

- Internet Explorer 누적 보안 업데이트(2977629)

이 보안 업데이트는 Internet Explorer의 공개된 취약점 1건과 비공개로 보고된 취약점 36건을 해결합니다. 가장 심각한 취약점은 사용자가 Internet Explorer를 사용하여 특수하게 조작된 웹 페이지를 볼 경우 원격 코드 실행을 허용할 수 있습니다. 취약점 악용에 성공한 공격자는 현재 사용자와 동일한 권한을 얻을 수 있습니다. 시스템에 대한 사용자 권한이 적게 구성된 계정의 고객은 관리자 권한으로 작업하는 고객에 비해 영향을 적게 받습니다.

- .NET Framework의 취약점으로 인한 서비스 거부 문제점(2990931)

이 보안 업데이트는 비공개적으로 보고된 Microsoft .NET Framework의 취약점 한 가지를 해결합니다. 이 취약점으로 인해 공격자가 소수의 특수하게 조작된 요청을 영향을 받는 .NET 기반 웹 사이트로 보낼 경우 서비스 거부가 발생할 수 있습니다. 지원되는 Microsoft Windows 에디션에 Microsoft .NET Framework가 설치된 경우 ASP.NET는 기본적으로 설치되지 않습니다. 이 취약점의 영향을 받으려면 고객은 수동으로 설치하고 이를 IIS에 등록하여 ASP.NET을 활성화해야 합니다

- Windows 작업 스케줄러의 취약점으로 인한 권한 상승 문제점(2988948)

이 보안 업데이트는 비공개적으로 보고된 Microsoft Windows의 취약점을 해결합니다. 이 취약점으로 인해 공격자가 영향을 받는 시스템에 로그온한 후 특수하게 조작한 응용 프로그램을 실행할 경우 권한 상승이 허용될 수 있습니다. 이 취약점을 악용하려면 공격자가 유효한 로그온 자격 증명을 가지고 로컬로 로그온할 수 있어야 합니다. 익명의 사용자에 의해서나 원격으로는 이 취약점을 악용할 수 없습니다.

- Microsoft Lync Server의 취약점으로 인한 서비스 거부 문제점(2990928)

이 보안 업데이트는 Microsoft Lync Server에서 발견되어 비공개적으로 보고된 취약점 3건을 해결합니다. 이 중 가장 심각한 취약점으로 인해 공격자가 특수하게 조작된 요청을 Lync Server에 보내는 경우 서비스 거부 문제점이 발생할 수 있습니다.

- 해결법

Windows Update를 수행하거나 Microsoft 보안 공지 요약 사이트에서 해당 취약점들의 개별적인 패치 파일을 다운로드 받을 수 있습니다.

- 한글 : http://technet.microsoft.com/ko-kr/security/bulletin/ms14-Jul
- 영문 : http://technet.microsoft.com/en-us/security/bulletin/ms14-Jul

Adobe Flash Player 신규 취약점 보안 업데이트

- 상세정보

- Adobe Flash Player에서 발생하는 12개의 취약점을 해결하는 보안 업데이트를 발표
- 랜덤 메모리 주소 기능을 우회할 수 있는 메모리 정보 노출 취약점(CVE-2014-0557)
- 보안 기능을 우회할 수 있는 취약점(CVE-2014-0554)
- 임의코드 실행으로 이어질 수 있는 메모리 할당 해제(use-after-free) 취약점(CVE-2014-0553)
- 임의코드 실행으로 이어질 수 있는 메모리 손상 취약점(CVE-2014-0547, CVE-2014-0549, CVE-2014-0550, CVE-2014-0551, CVE-2014-0552, CVE-2014-0555)
- 동일 출처 정책(same origin policy)을 우회할 수 있는 취약점(CVE-2014-0548)
- 임의코드 실행으로 이어질 수 있는 힙 오버플로우 취약점(CVE-2014-0556, CVE-2014-0559)

- 해결법

· 윈도우즈, 맥 환경의 Adobe Flash Player desktop runtime 사용자는 15.0.0.152 버전으로 업데이트 적용
· 윈도우즈, 맥 환경의 Adobe Flash Player Extended Support Release 사용자는 13.0.0.244 버전으로 업데이트 적용
· 리눅스 환경의 Adobe Flash Player 사용자는 11.2.202.406 버전으로 업데이트 적용
· 윈도우즈, 맥, 리눅스 환경의 Adobe Flash Player 보안 업데이트 적용방법 : Adobe Flash Player Download Center (http://www.adobe.com/go/getflash) 에 방문하여 최신 버전을 설치하거나 자동 업데이트를 이용하여 업그레이드
· 구글 크롬, 인터넷 익스플로러 10 및 인터넷 익스플로러 11에 Adobe Flash Player를 설치한 사용자는 자동으로 최신 업데이트 적용
· Adobe AIR desktop runtime, SDK 및 SDK&Compiler 사용자는 15.0.0.249 버전으로 업데이트 적용 : http://www.adobe.com/devnet/air/air-sdk-download.html에 방문하여 Adobe AIR SDK 또는 Adobe AIR SDK&Compiler 최신 버전을 설치
· 안드로이드 환경의 Adobe AIR 사용자는 15.0.0.252 버전으로 업데이트 적용 : Adobe AIR가 설치된 안드로이드 폰에서 '구글 플레이 스토어' 접속 → 메뉴 선택 → 내 애플리케이션 선택 → Adobe AIR 안드로이드 최신 버전으로 업데이트 하거나 자동업데이트를 허용하여 업그레이드

Cisco Unified Computing System SSH 서비스 거부 취약점 업데이트

CISCO社는 Cisco UCS(Unified Computing System) E-Series 블레이드 서버의 IMC(Integrated Management Controller) SSH(Secure Shell) 모듈에서 발생한 서비스 거부 취약점을 해결한 보안 업데이트를 발표(CVE-2014-3348)

- 상세정보

공격자가 특수하게 조작한 패킷을 IMC(Integrated Management Controller)의 SSH 모듈이 탑재된 취약한 장비에 전송할 경우, 서비스 거부 등을 유발할 수 있음
해당 취약점에 영향을 받는 제품은 서비스 거부 공격 등에 영향을 받을 수 있어 최신버전으로 업데이트 권고

- 해결법

해당 취약점에 영향 받는 장비의 운영자는 유지보수 업체를 통하여 패치 적용

- 참고사이트

http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20140908-ucse

Adobe Reader/Acrobat 취약점 보안 업데이트

Adobe社는 Adobe Reader/Acrobat의 취약점 6개에 대한 보안 업데이트를 발표

- 상세정보

· 임의코드 실행으로 이어질 수 있는 메모리 할당 해제(use-after-free) 취약점(CVE-2014-0560)
· 맥 플랫폼에서 발생하는 크로스 사이트 스크립팅(Cross-Site Scripting(XSS)) 취약점(CVE-2014-0562)
· 메모리 손상과 관계된 서비스 거부 취약점(CVE-2014-0563)
· 임의코드 실행으로 이어질 수 있는 힙 오버플로우 취약점(CVE-2014-0567)
· 임의코드 실행으로 이어질 수 있는 메모리 손상 취약점(CVE-2014-0566)
· 윈도우에서 권한 상승이 가능한 샌드박스 우회 취약점(CVE-2014-0568)

- 해결법

· Adobe Reader 사용자
– Adobe Download Center(http://www.adobe.com/support/downloads/product.jsp?product=10&platform=Windows)를 방문하여 최신 버전을 설치하거나 [메뉴]→[도움말]→[업데이트확인]을 이용하여 업그레이드

· Adobe Acrobat 사용자

– 아래의 Adobe Download Center를 방문하여 최신 버전을 설치하거나 [메뉴]→[도움말]→[업데이트확인]을 이용하여 업그레이드

– 윈도우 환경에서 동작하는 Adobe Acrobat Standard/Pro 사용자 :
http://www.adobe.com/support/downloads/product.jsp?product=1&platform=Windows

– 윈도우 환경에서 동작하는 Adobe Acrobat Pro Extended 사용자 :
http://www.adobe.com/support/downloads/product.jsp?product=1&platform=Windows

긴급 홈페이지 보안 점검 요청

– 상세정보

홈페이지 자체 보안 점검 요청

· 대상 : 홈페이지를 운영하는 중소기업

· 목적 : 홈페이지 자체 보안점검을 통하여 해킹 피해 방지

· 내용 : 홈페이지 해킹도구(웹셸) 확인 및 취약점 자체 보안 점검 요청

– 해결법

웹셸 탐지도구 '휘슬' 사용 신청 안내

· 중소기업에서 홈페이지 해킹도구(웹셸)삽입 확인이 어려울 경우, 한국인터넷진흥원에서 제공하는 웹셸 탐지 도구인 '휘슬'을 이용하시기 바랍니다.

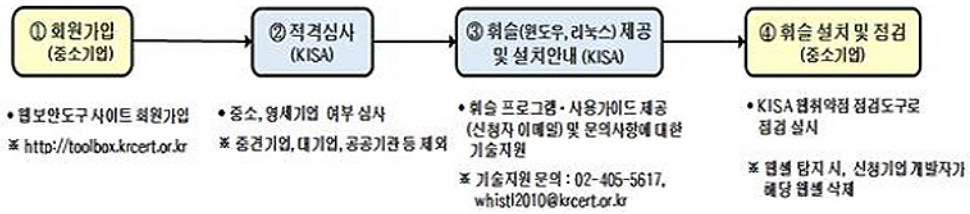
※ 웹셸(Web Shell) : 보안이 취약한 사이트 게시판 등에 악성코드가 포함된 파일을 업로드하여 시스템 관리자 권한 획득 후 개인정보를 수집하는 방식

· 문의 연락처 : 02-405-5617, whistl2010@krcert.or.kr

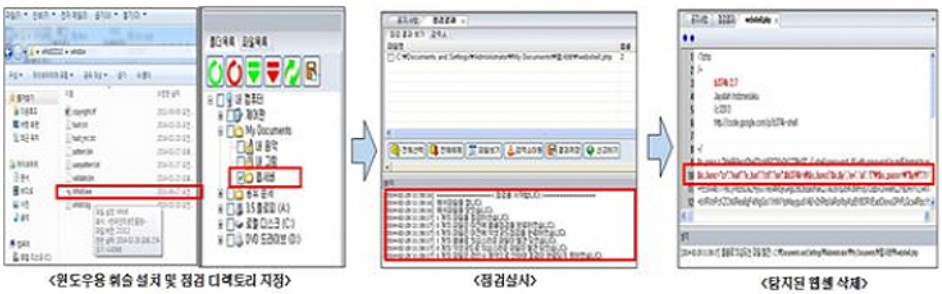
· 휘슬 사용 신청 안내사이트 : http://toolbox.krcert.or.kr/MMVF/MMVFView_V.aspx?MENU_CODE=6&PAGE_NUMBER=15

※ 휘슬 : 관리자가 운영하는 웹서버에 해커가 설치한 웹셸이 있는지 쉽게 탐지할 수 있도록 KISA에서 개발한 웹 보안도구

[휘슬 제공 절차]



[휘슬(윈도우, 리눅스) 설치 및 점검 절차]



원격 웹취약점 점검서비스 안내

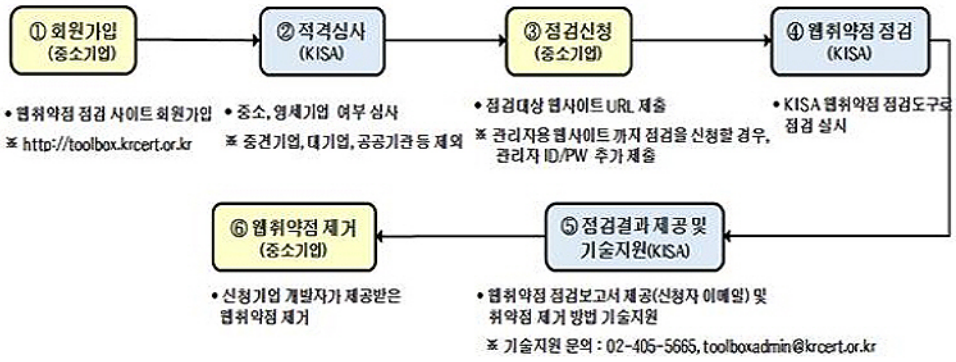
· 홈페이지를 운영하는 중소기업에서 자체적으로 취약점(SQL injection 취약점 등) 확인이 dl려울 경우 한국인터넷진흥원에서 제공하는 원격 웹취약점 점검 서비스를 받으시기 바랍니다.

※ SQL 인젝션 : 사이트 운영 데이터 베이스의 취약점을 이용, 직접 데이터 베이스 명령을 통해 관리자 권한을 획득 후 개인정보를 수집하는 방식

· 문의 연락처 : 02-405-5665, toolboxadmin@krcert.or.kr

· 취약점 점검 요청 안내사이트 : http://toolbox.krcert.or.kr/MMVF/MMVFView_V.aspx?MENU_CODE=40&PAGE_NUMBER=17

[원격 웹취약점 점검서비스 제공 절차]



아래한글 임의코드 실행 취약점 보안 업데이트

- 상세정보

공격자는 특수하게 조작한 웹페이지 방문 유도 또는 웹 게시물, 메일, 메신저의 링크 등을 통해 특수하게 조작된 문서를 열어보도록 유도하여 임의코드를 실행시킬 수 있음

영향 받는 버전의 사용자는 악성코드 감염에 취약할 수 있으므로 해결방안에 따라 보안 업데이트를 권고함

[영향을 받는 소프트웨어]

제품	영향 받는 버전 (보안#21)	영향받지 않는 버전 (보안#22)
한/글 2014	9.0.0.1397	9.0.0.1677
한/셀 2014	9.0.0.1388	9.0.0.1675
한/쇼 2014	9.0.0.1440	9.0.0.1742
한/글 2010	8.5.8.1422	8.5.8.1424
한/셀 2010	8.5.8.1336	8.5.8.1338
한/쇼 2010	8.5.8.1480	8.5.8.1482
한/글 2007	7.5.12.699	7.5.12.704
넥/셀 2007	7.5.12.757	7.5.12.762
한/슬 2007	7.5.12.900	7.5.12.905
한/글 2005	6.7.10.1107	6.7.10.1120
한/글 2004	6.0.5.805	6.0.5.806
한/글 2002	5.7.9.3079	5.7.9.3080

- 해결법

· 한글과컴퓨터 홈페이지에서 보안업데이트 파일을 직접 다운로드 받아 설치하여 영향 받지 않는 버전(보안#22)으로 업데이트

– 다운로드 경로 : http://www.hancom.co.kr/downLoad.downPU.do?mcd=001

· 한글과컴퓨터 자동 업데이트를 통해 최신버전으로 업데이트

– 시작 → 모든 프로그램 → 한글과컴퓨터 → 한글과컴퓨터 자동 업데이트

- 참고사이트

http://www.hancom.co.kr/downLoad.downPU.do?mcd=001

Bourne Again Shell (Bash) 임의코드 실행 취약점 보안 업데이트

리눅스 계열 등 운영체제에서 사용중인 GNU Bash에서 발생하는 임의코드 실행 취약점 (CVE-2014-6271)을 해결하는 일부 보안 업데이트 발표

- 상세정보

공격자는 Bash를 사용하여 구현된 기능 등을 악용하여 임의의 코드를 실행시킬 수 있으므로 해결방안 적용 권고

- 해결법

해당 취약점에 대한 보안업데이트가 공개된 OS를 운영하고 있을 경우, 참고사이트의 내용을 참조하여 보안업데이트 수행

· CentOS (http://lists.centos.org/pipermail/centos/2014-September/146099.html)

· Debian (https://www.debian.org/security/2014/dsa-3032)

· Redhat (https://access.redhat.com/solutions/1207723)

· Ubuntu (http://www.ubuntu.com/usn/usn-2362-1/)

- 참고사이트

· http://lists.centos.org/pipermail/centos/2014-September/146099.html

· https://www.debian.org/security/2014/dsa-3032

· https://access.redhat.com/solutions/1207723

· http://www.ubuntu.com/usn/usn-2362-1/

(2차) Bourne Again Shell (Bash) 임의코드 실행 취약점 보안 업데이트

리눅스 계열 등 운영체제에서 사용중인 GNU Bash에서 발생하는 임의코드 실행 취약점 (CVE-2014-6271) 보안 업데이트를 우회하여 악용할 수 있는 취약점 발생 (CVE-2014-7169)

- 상세정보

공격자는 해당 취약점이 존재하는 시스템을 대상으로 원격에서 악의적인 시스템 명령을 실행시킬 수 있으므로 해결방안에 따라 보안 업데이트 적용 권고

- 해결법

해당 취약점에 대한 보안업데이트가 공개된 OS를 운영하고 있을 경우, 참고사이트의 내용을 참조하여 보안업데이트 수행

· CentOS (http://lists.centos.org/pipermail/centos/2014-September/146154.html)

· Debian (https://www.debian.org/security/2014/dsa-3035)

· Redhat (https://rhn.redhat.com/errata/RHSA-2014-1306.html)

· Ubuntu (http://www.ubuntu.com/usn/usn-2363-2/)

- 참고사이트

- http://lists.centos.org/pipermail/centos/2014-September/146154.html
- https://www.debian.org/security/2014/dsa-3035
- https://rhn.redhat.com/errata/RHSA-2014-1306.html
- http://www.ubuntu.com/usn/usn-2363-2/

Part4. 해외 보안 동향

영미권

중국

일본

1. 영미권

중대한 Bash 취약점이 Linux, UNIX, Mac OS X에 영향을 미친다

Major Bash Vulnerability Affects Linux, UNIX, Mac OS X

대부분의 리눅스, 유닉스, 애플 맥 OS X의 셸 프로그램 'Bash'에 중대한 보안 취약점이 발견되었다. 관리자들은 이를 즉시 패치하기를 권고한다. 이 결점은 공격자로 하여금 원격으로 Bash가 적용 될 때 실행되는 변수에 악성 실행 명령(malicious executable)을 부여 할 수 있게 한다. 레드햇의 시큐리티 매니저는, 모든 버전의 Bash가 이에 취약하고 매우 심각한 취약점이지만, 다행스럽게도 공격 가능한 환경을 조성하기는 다소 까다롭다고 밝혔다.

레드햇은 해당 코드 인젝션 취약점을 가진 배시의 버전을 테스트하기 위한 진단방법을 공개했다.

출처 : Threat Post(http://threatpost.com/major-bash-vulnerability-affects-linux-unix-mac-os-x/108521)

홈디포의 데이터 유출로 인해 5,600만의 신용카드 정보 노출

Home Depot Breach Exposes 56 Million Credit Cards

홈 디포의 조사 보고서에 따르면, 지난 9월 2일부터, 약 5천6백만개의 신용카드 정보가 홈 디포의 보안 취약점에 노출되어 있었다. 홈 디포는 법무부와 뱅킹 파트너사들이 문제를 제보하여, 데이터 노출 문제를 9월 초에 발견하였다. 홈 디포는 "범죄자들은 그들의 고유한, 커스텀 빌드된 멀웨어를 사용하여 탐지를 피했다. 당사의 보안 파트너사에 따르면, 해당 멀웨어는 이전에 탐지된 적도 없다."고 밝혔다.

아직까지 신용카드의 PIN 넘버까지 유출되었다는 증거는 없다. PIN 넘버는 온라인 결제나 멕시코 점포에만 사용 되었기 때문이다. 또한 홈디포는 지불 정보를 암호화 하기 위한 새로운 기술을 도입하였으며, 모든 미국 점포에 적용했다. 또한 2015년 말까지 모든 미국 점포에 '칩 앤 핀'(Chip and PIN) 표준을 도입하기로 하였다.

홈 디포는 미국, 캐나다, 멕시코 전역에 2,266개의 점포를 가지고 있는 건축, 인테리어 관련 체인이다. 그러므로 홈 디포의 타겟 형식의 멀웨어 감염으로 인한 데이터 유출은 꽤 큰 사건이라고 할 수 있다. 또한 범죄자들은 지난 2013년 Target의 PoS 터미널을 해킹하여 4천만개 이상의 크레딧카드 기록을 탈취한 바 있다. 이는 고작 3주도 채 되지 않는 기간의 기록이었다.

미국 스타일의 신용카드 트랜잭션은, 보통 윈도우 기반의 캐셔 지불 장치로부터 보내진 카드 데이터가 암호화 되지 않은 상태로 USB 케이블을 통하여 키보드 입력과 함께 전달된다. 따라서 캐셔에 설치 된 멀웨어가 메모리에 도착하는 데이터 열람 및 탈취가 가능하다. 이런 종류의 트로이 목마를 '램 스크래퍼(Ram Scrapers)'라고 한다.

크레딧 카드 데이터는 캐셔에 도달한 후에야 암호화 되므로, 이렇게 취약한 암호화 방식으로 설계된 캐셔 장비들을 해킹하는 것은 지불 기기 자체를 해킹하는 것보다 쉽다. 이 중 직불카드 PIN넘버는 지불 장치에 순수한 텍스트 형태로 전송 되지 않으므로 RAM에서 긁어가기 쉽지 않다.

출처 : Naked Security, Hot for Security(http://nakedsecurity.sophos.com/2014/09/09/home-depot-says-er-yes-we-did-have-a-breach-actually%e2%80%8f/ , http://www.hotforsecurity.com/blog/home-depot-breach-exposes-56-million-credit-cards-10172.html)

버라이존의 소비자 프라이버시 문제에 역대 가장 많은 벌금 부여

버라이존이 미 연방통신위원회(FCC: Federal Communications Commission)에 $7.4백만의 벌금을 내게 되었다. 고객의 개인 정보를 개인 맞춤 마케팅에 활용하기 전에 이를 알리고 동의를 받지 않았기 때문이다. 이 벌금은 고객 프라이버시 관련 FCC 역대 최고 금액이다.

벌금과는 별개로 버라이존은 향후 3년간 모든 고객들에게 청구서를 통해서 고객 맞춤 마케팅에 참여하지 않을 권리를 통지해야 한다. 또한 회사의 상급 관리자를 이러한 규정 준수 내용 관리자로 지명해야 하고, 이와 관련하여 일어나는 모든 문제에 대해 규모에 상관 없이 관리자에게 즉시 보고 되어야 하며, 3년간의 법규 준수 계획을 개발하여 적용해야만 한다.

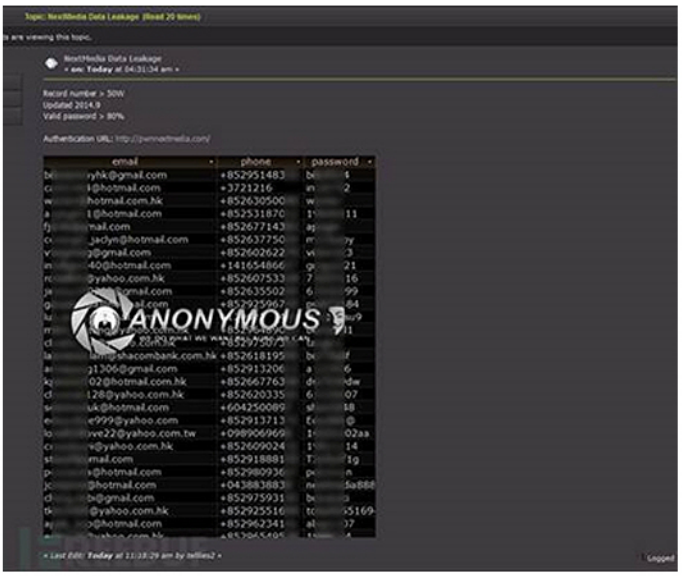
버라이존은 2006년부터 2백만 이상의 고객들에게 보내지는 청구서나 웰컴 레터에 수신거부 공지를 포함하지 않은 것으로 드러났다. 또한 버라이존의 직원들은 2012년 9월까지 이를 알지 못했으며, 발견 후로도 126일간 FCC에 이를 알리지 않았다.

출처 : Threat Post(http://threatpost.com/verizon-to-pay-largest-ever-consumer-privacy-settlement/108096)

2.중국

홍콩 미디어인 nextmedia.com 수십만 회원정보 유출

화요일, 해외 해커 커뮤니티에서 홍콩매체 nextmedia의 회원 수십만 명의 회원정보가 유출되었다. 여기에는 이메일 주소, 암호화되지 않은 비밀번호와 휴대폰 번호가 포함됐다. 이번에 정보 유출 피해를 입은 사용자들은 50만명이 넘으며, 해당 정보의 최근 업데이트 기간은 2014년 9월이다. 해당 비밀번호는 80% 이상으로 일치한 것으로 나타났다. 이번 사건과 관련하여 nextmedia에서는 어떠한 공식 입장도 밝히지 않고 있으며, 홈페이지에도 역시 비밀번호 수정에 대한 어떠한 언급도 보이고 있지 않다.



해당 홈페이지는 Alexa 순위는 전 세계순위 1654등, 홍콩홈페이지 기준 10등을 차지하고 있다.

자신의 정보유출 여부는 해당 페이지(http://www.pwnnextmedia.com)에서 확인할 수 있다.
출처 : http://www.freebuf.com/news/44608.html

컴퓨터 악성코드 감염률이 5년만에 처음으로 상승

중국 컴퓨터 악성코드 긴급대응센터가 발표한 '2013년 중국 인터넷 보안의 현황과 PC.모바일 플랫폼의 악성코드 감염현황조사' 보고서에 따르면, 2013년, 중국의 PC악성코드 감염률은 54.9%로, 전년대비 9.8% 상승했다. 5년동안 지속적으로 감소하던 감염률이 처음으로 상승세로 전환한 것이다. 업계의 분석에 따르면, 각종 매체에서 인터넷 보안의식을 고취시킨 점이 큰 이유로 작용한 것으로 보인다.

해커들은 경제적 이득을 취하기 위해 온라인 뱅킹, 온라인 결제 등을 악성코드의 주요 타겟으로 삼는다. 또한 피싱이나 파밍으로 금전적 탈취를 꾀할 뿐만 아니라, 사용자들의 개인정보 역시 탈취한다. 최근에는 웨이보가 새로운 타겟으로 떠올랐다. 대기업, 주요 은행 등을 타겟으로 하는 악성코드 및 apt공격이 증가했으며, 조사결과, 과반수가 넘는 정부기관 홈페이지에 보안취약점이 존재하는 것으로 확인되었다.

출처 : http://www.ithome.com/html/it/103629.htm

3. 일본

JAL 마일리지 회원 개인정보유출 최대 75만건 사내 PC에 멀웨어, 원격조작으로 데이터유출

JALマイレージ会員の個人情報流出　最大75万件　社内PCにマルウェア、遠隔操作か

일본항공JAL은 9월 24일 사내의 고객관리 시스템이 부정 액세스를 받아, 'JAL 마일리지 뱅크'회원의 개인정보가 최대 75만건이 유출됐을 가능성이 있다고 발표했다. 사내의 일부 PC에 멀웨어가 설치되어 원격조작으로 외부에 데이터가 보내졌을 가능성이 있다고 밝혔다.

유출 가능성이 있는 것은 마일리지 회원의 성명, 생년월일, 주소, 전화번호, 근무지, 메일주소, 회원번호, 가입날짜 등이지만 구체적인 내용에 관해서는 확정되지 않았다. 멀웨어는 7월30일 이후 23대의 PC에 설치되었으며 그 중 12대가 고객관리 시스템이 접근할 수 있는 단말기였고 실제로 데이터를 송신한 PC는 7대였다고 한다.

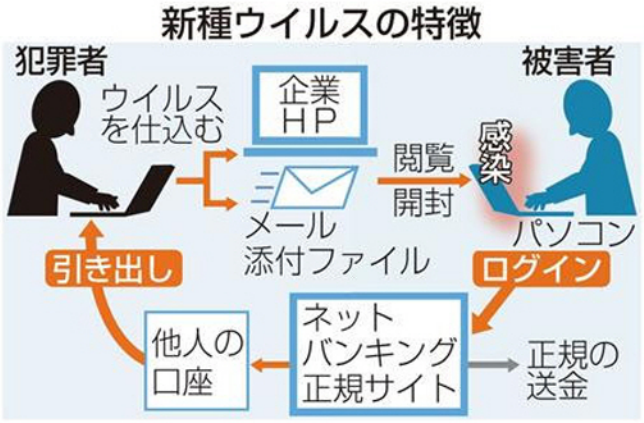
9월 19일 데이터베이스의 액세스가 집중되어 응답이 지연되었다. 특정 프로세스를 삭제하는 것으로 문제를 해결하였으나 22일 재발하여 조사한 결과 평소에 사용하지 않는 PC로 고객관리 시스템에 액세스한 것으로 판명되었다.

출처 : IT media(http://www.itmedia.co.jp/news/articles/1409/24/news165.html)

인터넷 부정 송금, 신종 바이러스 2만건초과 세계의 80%가 일본을 표적으로

新種ウイルス2万件超！ネット不正送金、世界の8割が日本標的

인터넷뱅킹의 정규사이트에 로그인하는 것만으로 범죄자의 계좌로 자동적으로 부정 송금되는 신종 바이러스가 5월이후 일본에서 2만건 이상 검출되었다. 일본에서의 검출 건수는 세계의 80%로 집중적으로 표적이 되고 있는 것이 판명되었다.



트렌드마이크로가 판매하고 있는 백신을 사용하고 있는 컴퓨터로부터 검출하여 최근 검출 건수 등을 경찰에 보고한다. 신종바이러스는 이용자가 공식 인터넷뱅킹 사이트에 로그인하면 감염된 컴퓨터가 탐지하고 자동적으로 다른 명의의 계좌로 부정송금을 명령하는 구조이다.
신종 바이러스는 메일의 첨부파일이나 기업의 공식홈페이지에 설치되어 열람 시 감염될 우려가 있다.

출처 : 産経新聞(http://www.iza.ne.jp/kiji/events/news/140915/evt14091510540005-n1.html)

알약 10월 보안동향보고서

Contact us

㈜이스트소프트 보안대응팀

Tel : 02-3470-2999

E-mail : help@alyac.co.kr

알약 홈페이지 : www.alyac.co.kr