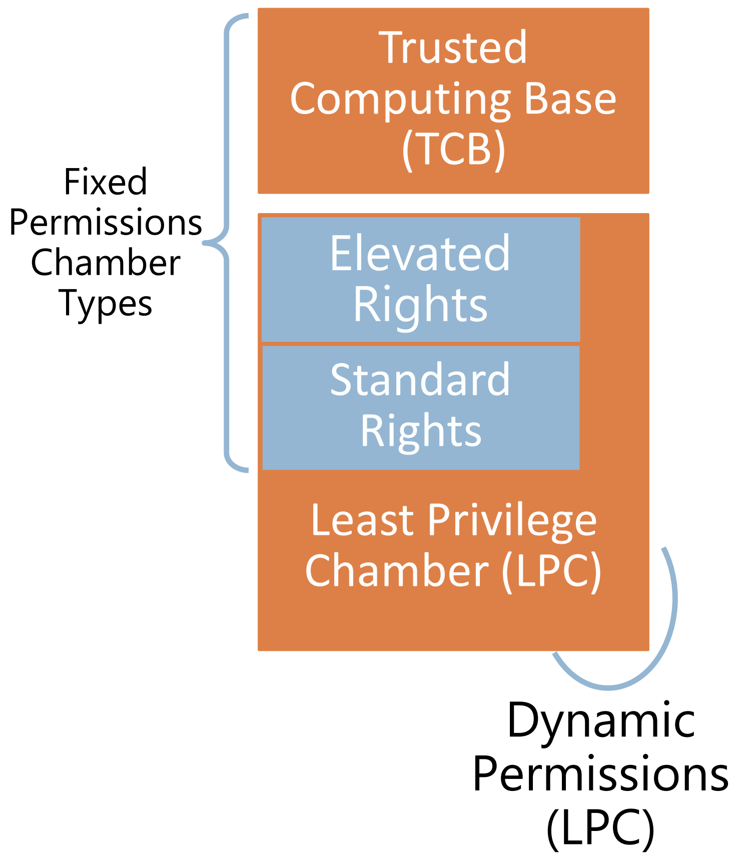
Outline

- Introduction: platforms and attacks
- Apple iOS security model
- Android security model
- Windows Phone 7/8 security model

Windows Phone 7/8 Security

- Secure boot
- All binaries are signed
- Device encryption
- Security model with isolation, capabilities
- Support for enterprise policies
- Distributing LOB apps (for a specific enterprise)

Windows Phone 7/8 Security Model



Trusted Computing Base (TCB)
Elevated Rights
Standard Rights
Least Privilege Chamber (LPC)
Fixed Permissions Chamber Types
Dynamic Permissions (LPC)

Policy system

Central repository of rules
3-tuple {Principal, Right, Resource}

Chamber Model

Chamber boundary is security boundary
Chambers defined using policy rules
4 chamber types, 3 fixed size, one can be expanded with capabilities (LPC)

Capabilities

Expressed in application manifest
Disclosed on Marketplace
Defines app's security boundary on phone

Overview of Four Chambers

- Elevated Rights Chamber (**ERC**)
  - Can access all resources except security policy
  - Intended for services and user-mode drivers
- Standard Rights Chamber (**SRC**)
  - Default for pre-installed applications that do not provide device-wide services
  - Outlook Mobile is an example that runs in the SRC
- Least Privileged Chamber (**LPC**)
  - Default chamber for all non-Microsoft applications
  - LPCs configured using capabilities (see next slide)

Overview of Four Chambers

- Trusted Computing Base (TCB) chamber
  - unrestricted access to most resources
  - can modify policy and enforce the security model.
  - kernel and kernel-mode drivers run in the TCB
  - Minimizing the amount of software that runs in the TCB is essential for minimizing the Windows Phone 7, 8 attack surface

Granting Privileges to Applications

- Goal: Least Privilege
  - Application gets capabilities needed to perform all its use cases, but no more
- Developers
  - Use the **capability detection tool** to create the capability list
  - The capability list is included in the application manifest
- Each application discloses its capabilities to the user
  - Listed on Windows Phone Marketplace
  - Explicit prompt upon application purchase
  - Disclosure within the application, when the user is about to use the location capability for the first time

WP7 Capabilities

- Video and Still capture; Video and Still capture ISV; Microphone; Location
- Services; Sensors; Media Library; Push Notifications; Web Browser
- Component; Add Ringtone; Place Phone Calls; Owner Identity; Phone
- Identity; Xbox LIVE; Interop Services; Networking; File Viewer; Appointments;
- Contacts; Debug; Networking Admin

Example: Code Requires Permission

```
class NativeMethods
{
    // This is a call to unmanaged code. Executing this method
    // requires the UnmanagedCode security permission. Without
    // this permission, an attempt to call this method will throw a
    // SecurityException:
    [DllImport("msvcrt.dll")]
    public static extern int puts(string str);
    [DllImport("msvcrt.dll")]
    internal static extern int _flushall();
}
```

Example: Code Denies Permission Not Needed

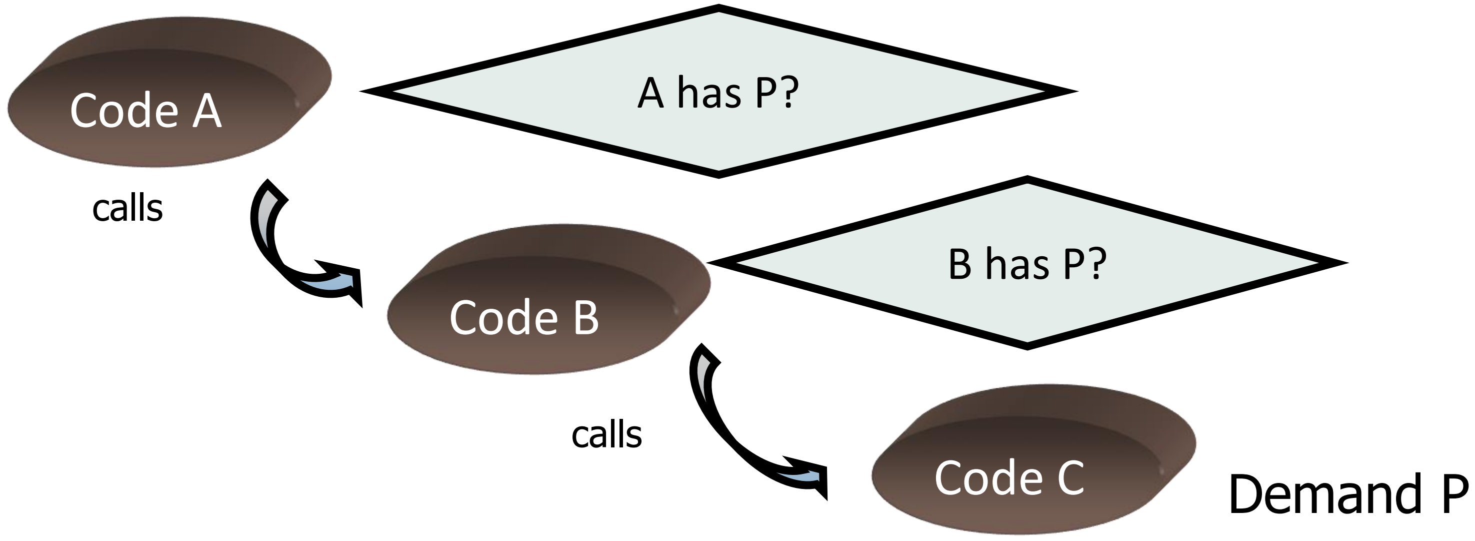
```
[SecurityPermission(SecurityAction.Deny,
        Flags = SecurityPermissionFlag.UnmanagedCode)]
private static void MethodToDoSomething()
{   try
    {
        Console.WriteLine("…");
        SomeOtherClass.method();
     }
    catch (SecurityException)
    {
        …
    }
}
```

.NET Code Access Security

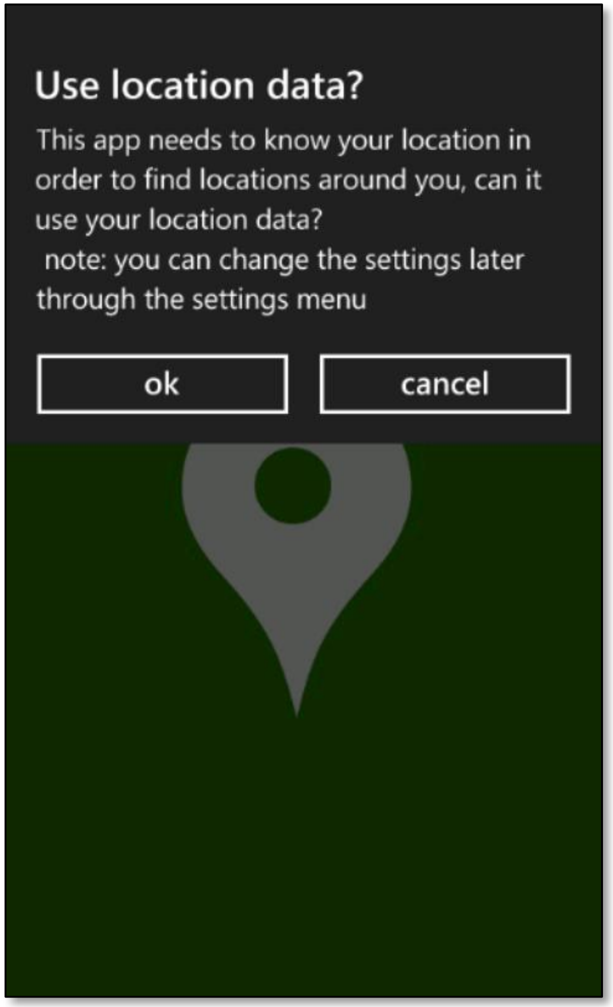
- Default Security Policy is part of the .NET Framework
  - Default permission for code access to protected resources
- Permissions can limit access to system resources.
  - Use `EnvironmentPermission` class for environment variables access permission.
  - The constructor defines the level of permission (read, write,…)
- Deny and Revert
  - The Deny method of the permission class denies access to the associated resource
  - The `RevertDeny` method will cause the effects of any previous Deny to be cancelled

.NET Stackwalk

- Demand must be satisfied by all callers
  - Ensures all code in causal chain is authorized
  - Cannot exploit other code with more privilege



AroundMe



Use location data?

This app needs to know your location in order to find locations around you, can it use your location data?
 note: you can change the settings later through the settings menu

```
public static bool AroundMe.App.CheckOptin() {
 if (((Option)Enum.Parse(typeof(Option),Config.
  (SettingConstants.UseMyLocation),true)) == Op
  return GetCurrentCoordinates();
 }
 if (MessageBox.Show("This app needs ...",
          "Use location data?", MessageBoxButto
        == MessageBoxResult.OK)
 {
  Config.UpdateSetting(new KeyValuePair<string,
  (SettingConstants.UseMyLocation,Option.Yes.To

  return GetCurrentCoordinates();
 }
 ...
}
```

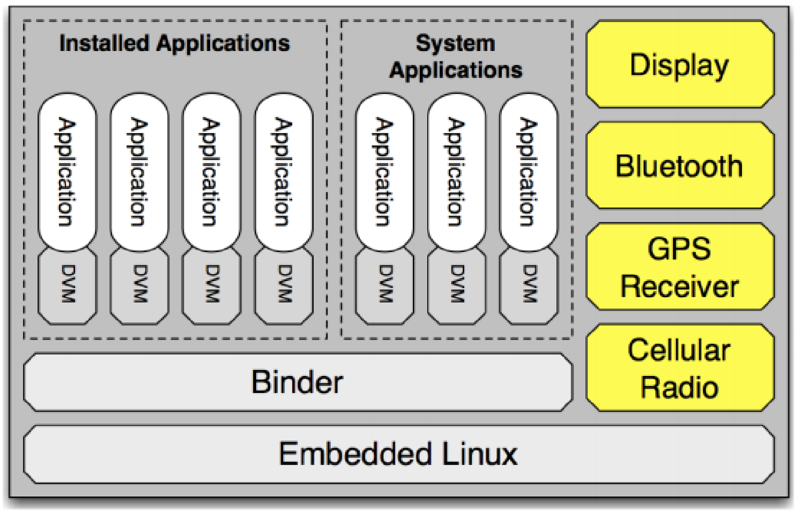
The Problem of Over-permissioning

- Flashlight XT (version 3.3.0.0)
  - video and still capture
  - camera
  - HD720P (720x1280)
  - WVGA (480x800)
  - WXGA (768x1280)
  - photo, music, and video libraries
  - microphone
  - camera
- Flashlight-X (6.6.0.0)
  - phone identity
  - owner identity
  - video and still capture
  - media playback
  - microphone
  - data services
  - movement and directional sensor
  - HD720P (720x1280)
  - WVGA (480x800)
  - WXGA (768x1280)
  - photo, music, and video libraries
  - camera
  - compass

Comparison

	iOS	Android	Windows
Unix	x	x	
Windows			x
Open market		x	
Closed market	x		x
Vendor signed	x		
Self-signed		x	x
User approval of permissions		x	x
Managed code		x	x
Native code	x		
Runtime prompts	x		

Android Security and Privacy



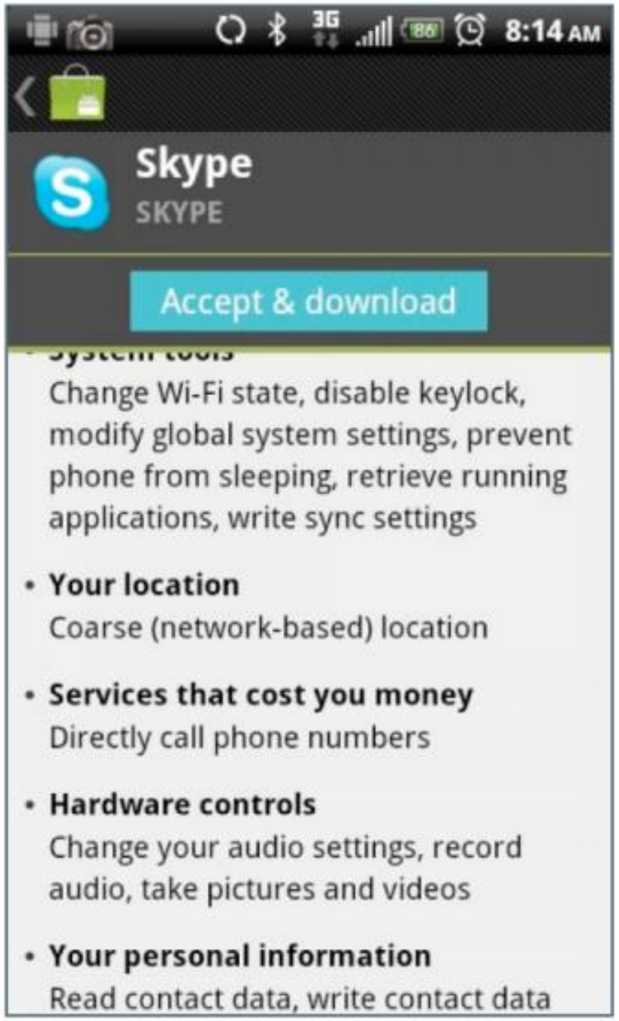
- Each app runs with its own user ID
- This gives apps a level of isolation
- But this doesn't prevent app attacks

[From Enck et al.,
"A Study of Android Application Security",
USENIX Security 2011.]

Application Permissions

- Apps must request permissions to access sensitive resources

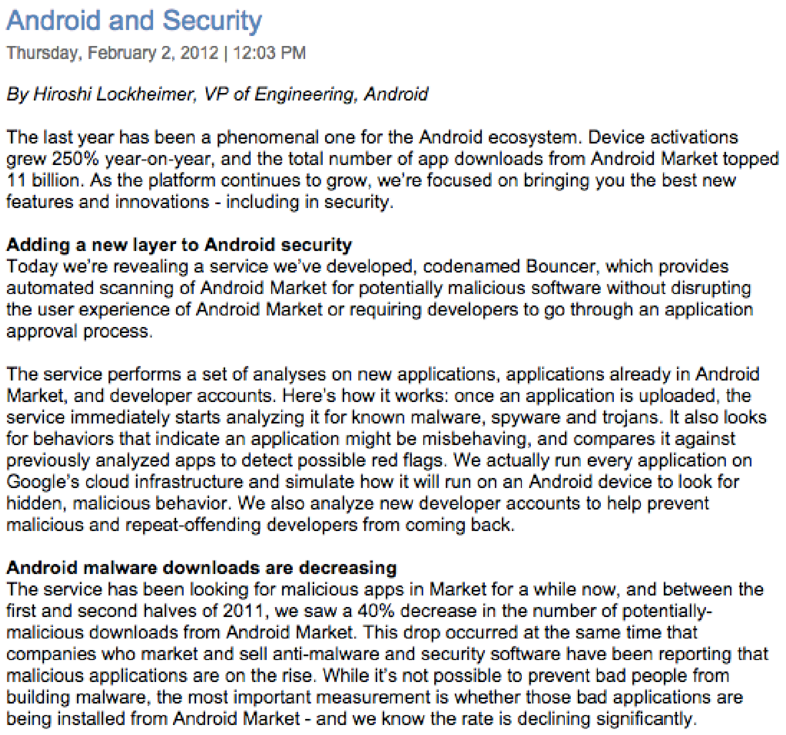
INTERNET, ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, CAMERA, CALL_PHONE, READ_CALENDAR, READ_PHONE_STATE, SEND_SMS, REBOOT, and many more.



System tools
Change Wi-Fi state, disable keylock, modify global system settings, prevent phone from sleeping, retrieve running applications, write sync settings

- **Your location**
  Coarse (network-based) location
- **Services that cost you money**
  Directly call phone numbers
- **Hardware controls**
  Change your audio settings, record audio, take pictures and videos
- **Your personal information**
  Read contact data, write contact data

Android Malware and Privacy



Android and Security

Thursday, February 2, 2012 | 12:03 PM

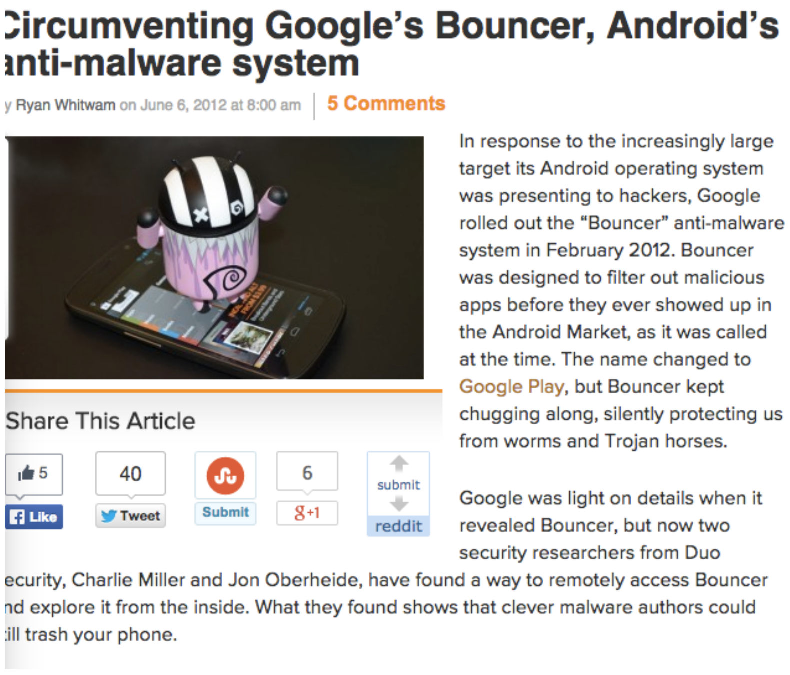
By Hiroshi Lockheimer, VP of Engineering, Android

The last year has been a phenomenal one for the Android ecosystem. Device activations grew 250% year-on-year, and the total number of app downloads from Android Market topped 11 billion. As the platform continues to grow, we're focused on bringing you the best new features and innovations - including in security.

Adding a new layer to Android security
Today we're revealing a service we've developed, codenamed Bouncer, which provides automated scanning of Android Market for potentially malicious software without disrupting the user experience of Android Market or requiring developers to go through an application approval process.

The service performs a set of analyses on new applications, applications already in Android Market, and developer accounts. Here's how it works: once an application is uploaded, the service immediately starts analyzing it for known malware, spyware and trojans. It also looks for behaviors that indicate an application might be misbehaving, and compares it against previously analyzed apps to detect possible red flags. We actually run every application on Google's cloud infrastructure and simulate how it will run on an Android device to look for hidden, malicious behavior. We also analyze new developer accounts to help prevent malicious and repeat-offending developers from coming back.

Android malware downloads are decreasing
The service has been looking for malicious apps in Market for a while now, and between the first and second halves of 2011, we saw a 40% decrease in the number of potentially-malicious downloads from Android Market. This drop occurred at the same time that companies who market and sell anti-malware and security software have been reporting that malicious applications are on the rise. While it's not possible to prevent bad people from building malware, the most important measurement is whether those bad applications are being installed from Android Market - and we know the rate is declining significantly.



Circumventing Google's Bouncer, Android's anti-malware system

y Ryan Whitwam on June 6, 2012 at 8:00 am | 5 Comments

Share This Article

In response to the increasingly large target its Android operating system was presenting to hackers, Google rolled out the "Bouncer" anti-malware system in February 2012. Bouncer was designed to filter out malicious apps before they ever showed up in the Android Market, as it was called at the time. The name changed to Google Play, but Bouncer kept chugging along, silently protecting us from worms and Trojan horses.

Google was light on details when it revealed Bouncer, but now two security researchers from Duo ecurity, Charlie Miller and Jon Oberheide, have found a way to remotely access Bouncer nd explore it from the inside. What they found shows that clever malware authors could ill trash your phone.

Android Malware Examples

- **Fake Banking Apps**
  - In 2009, while the Android Market was still in its infancy, a user known as **Droid09** uploaded several phony online banking apps to lure customers of major banking institutions into entering their online account logins.
  - "Informed of this, Google quickly removed them," said Robert Vamosi, senior analyst at Mocana and author of When Gadgets Betray Us.

- **Android.PjappsM**
  - Early in 2010, sly attackers downloaded legitimate programs from the Android Market, infected them with the Android.Pjapps malware, and then redistributed the modified versions on third-party Android marketplaces.
  - Goal: steal information from infected devices and enroll the device in a botnet that then launched attacks on websites to steal additional data and infect more devices. Send costly SMS messages.

Even More Android Malware…

- **DroidDream** (aka, Android.Rootcager)
  - One of the most nefarious malware campaigns addressed in Lookout's Mobile Threat Report, DroidDream infected roughly 60 different legitimate apps in the Android Market and infected 100Ks of users in 2011.
  - The malware added infected devices to a botnet, breached the Android security sandbox, installed additional software, and stole data.
- **Android.Bgserv**
  - Shortly after Google deployed a tool for users to clean up devices infected with DroidDream, malware authors got clever
  - Attackers capitalized on the hype and released a malicious fake version of the cleanup tool.
  - Known as Android.Bgserv, this malware stole device data, such as the phone's IMEI number and phone number, and uploaded it to a server in China.

From the 2014 McAffee Report…

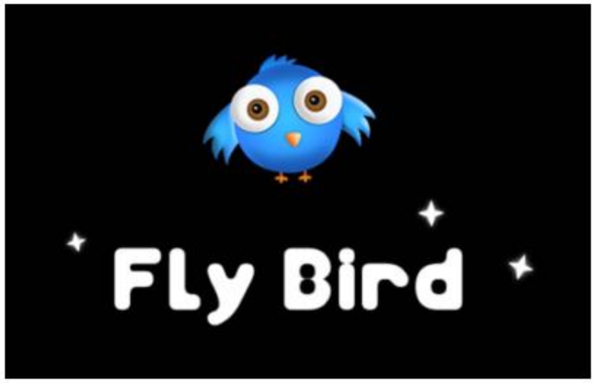
Attack of the Flappy Bird clones

Once again we see that social engineering, combined with the latest "hot game," leads to plentiful malware. The current infestation is a flock of malevolent "Flappy Bird" clones.

The original "Flappy Bird" game was released in mid-2013 on Apple iOS and early this year on Android. The game was a huge success, with more than 50 million downloads, and brought a great deal of notoriety to developer Dong Nguyen before he pulled the app from the marketplace in February.

During the last several *McAfee Labs Threats Reports,* we have reported on the steep rise in mobile malware. The Flappy Bird craze and subsequent malware sweep is a prime example of malware authors taking full advantage of over-the-top user enthusiasm for legitimate apps or games. Malicious Flappy Bird clones existed prior its removal from online marketplaces, but the demand for Flappy Bird–like games only rose after the app was pulled. During the first quarter of 2014, we saw hundreds of Flappy Bird clones emerge, the majority of which were malicious.

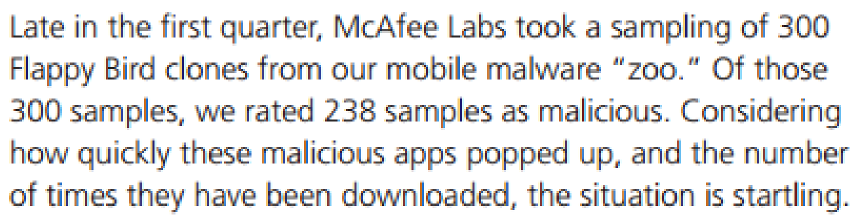
The original Flappy Bird game.



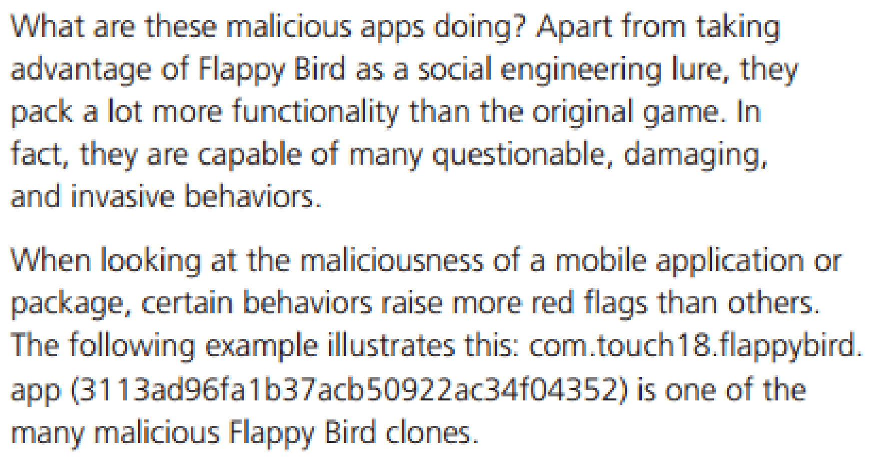
A malicious Flappy Bird clone.

http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2014.pdf

More Flappy Birds…

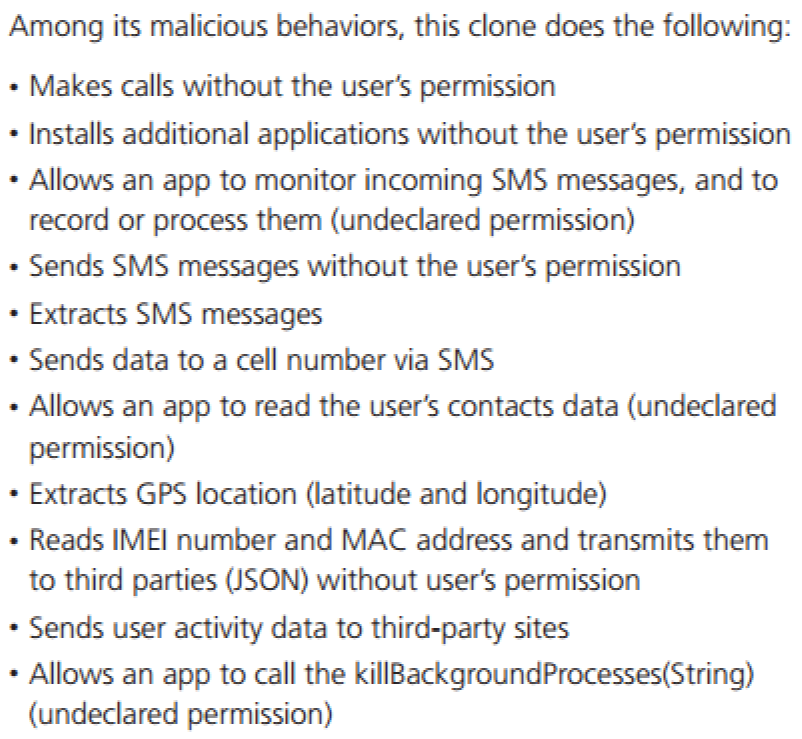


Late in the first quarter, McAfee Labs took a sampling of 300 Flappy Bird clones from our mobile malware "zoo." Of those 300 samples, we rated 238 samples as malicious. Considering how quickly these malicious apps popped up, and the number of times they have been downloaded, the situation is startling.



What are these malicious apps doing? Apart from taking advantage of Flappy Bird as a social engineering lure, they pack a lot more functionality than the original game. In fact, they are capable of many questionable, damaging, and invasive behaviors.

When looking at the maliciousness of a mobile application or package, certain behaviors raise more red flags than others. The following example illustrates this: com.touch18.flappybird.app (3113ad96fa1b37acb50922ac34f04352) is one of the many malicious Flappy Bird clones.



Among its malicious behaviors, this clone does the following:

- Makes calls without the user's permission
- Installs additional applications without the user's permission
- Allows an app to monitor incoming SMS messages, and to record or process them (undeclared permission)
- Sends SMS messages without the user's permission
- Extracts SMS messages
- Sends data to a cell number via SMS
- Allows an app to read the user's contacts data (undeclared permission)
- Extracts GPS location (latitude and longitude)
- Reads IMEI number and MAC address and transmits them to third parties (JSON) without user's permission
- Sends user activity data to third-party sites
- Allows an app to call the killBackgroundProcesses(String) (undeclared permission)

How About Malware on a Larger Scale?

Financial gain has been the primary motive behind the botnet malware industry for many years. There is money to be made for the authors of malware, kits, and exploits, as well as for those who buy them and create their own botnets.

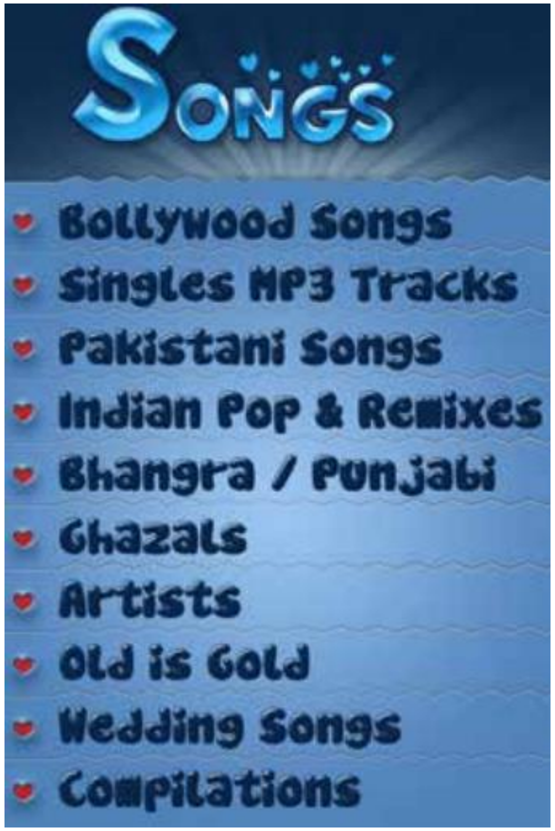
Recently, an additional factor has come into the picture: the commoditization of virtual currency mining as a core botnet function. We see this functionality being adopted across popular platforms, including mobile. This emergence is very similar to past innovations in bots and malware, such as the rise of distributed denial of service (DDoS) attacks, the persistence of installations, private update mechanisms, and active detection evasion.

Spend some time digging around any underground security forum or marketplace and you will find a myriad of SHA-256 and SCRYPT miner botnets, builders, and cracked versions of commercial builders and kits, along with the usual assortment of DDoS bots, cryptors, and other nefarious services and tools. Some recent examples include EnvyMiner, DeadCow, SovietMiner, JHTTP, Black Puppet, and Aura. These are just a tiny fraction of what exists.

Bitcoin Mining: Computationally Heavy Activity

- Bitcoin mining
  - In traditional fiat money systems, governments simply print more money when they need to
  - But in bitcoin, money isn't printed at all – it is discovered
  - Computers around the world 'mine' for coins by competing with each other
- Currently, more than 12 million are in circulation.
- A little less than 9 million bitcoins are waiting to be discovered (capped at 21M)
- "Mining" is lingo for the discovery of new bitcoins—just like finding gold.
- In reality, it's simply the verification of bitcoin transactions – a computationally heavy activity.

Some Mining Apps



Songs

- Bollywood Songs
- Singles MP3 Tracks
- Pakistani Songs
- Indian Pop & Remixes
- Bhangra / Punjabi
- Ghazals
- Artists
- Old is Gold
- Wedding Songs
- Compilations

Finding and Fixing Vulnerabilities with AppScan

Automates Application Security Testing

Same process for whitebox & blackbox

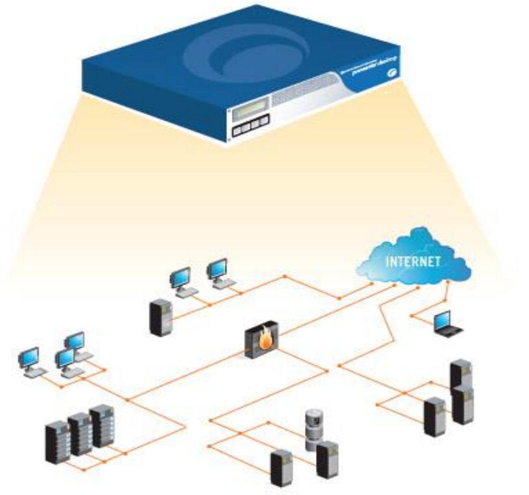
Scan applications

Analyze

(identify issues)

Report

(detailed & actionable)

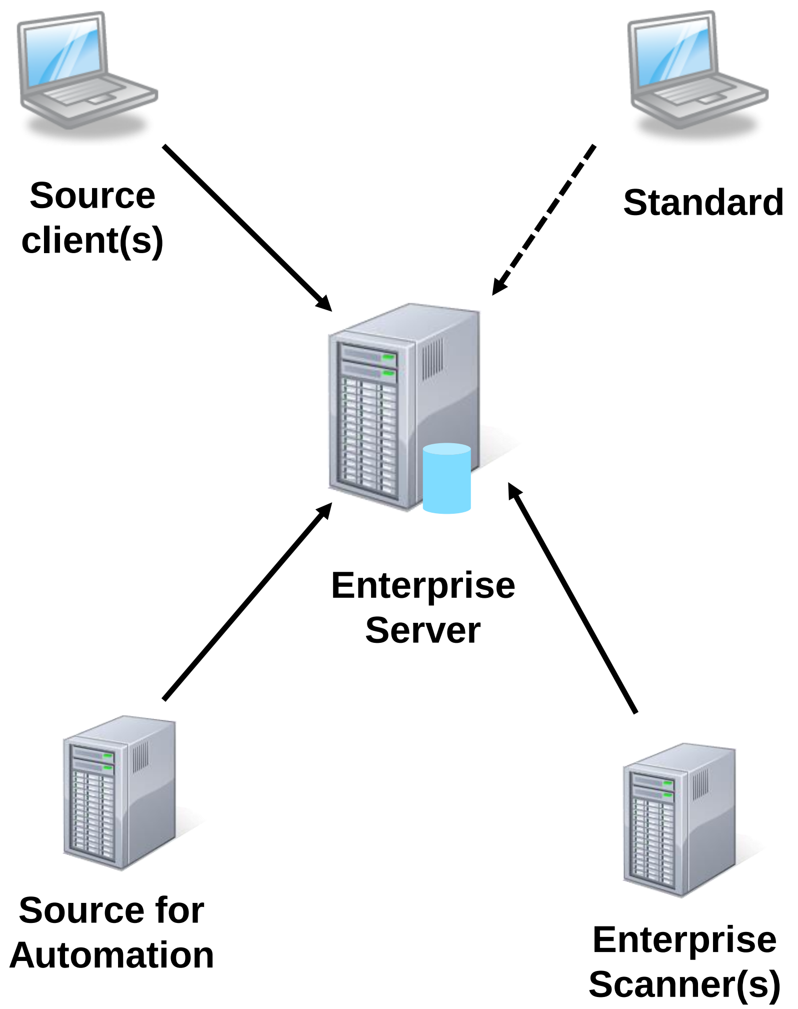


Scanning Techniques 101

	Static Analysis SAST - Static Application Security Testing	Dynamic Analysis DAST – Dynamic Application Security Testing
Scan input	**Source code**	**“Running” web application**
Assessment Techniques	**Taint analysis & pattern matching “code auditing”**	**Tampering with HTTP messages**
Who uses it	Application developers	Development, QA, Security, Deployment
Results and output	**Results are presented by line of code**	**Results are presented as HTTP messages (exploit requests)**

No other vendor provides a broader set of scanning techniques

AppScan Portfolio Overview



Portfolio Overview

AppScan Standard

Base price 36k – Avg. 50k

•Desktop tool for Dynamic. One scan and assessment at a time

AppScan Enterprise

Base price 120k – Avg. 200k

•Server solution for Dynamic. Scanners on servers for parallel scans. Server stores all assessments for centralized reporting and web access.

AppScan Source

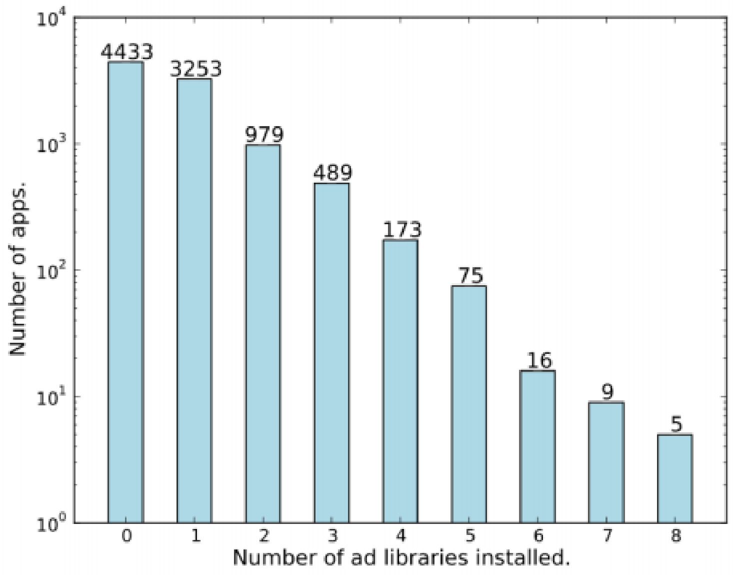
Base price 100k – Avg. 180k

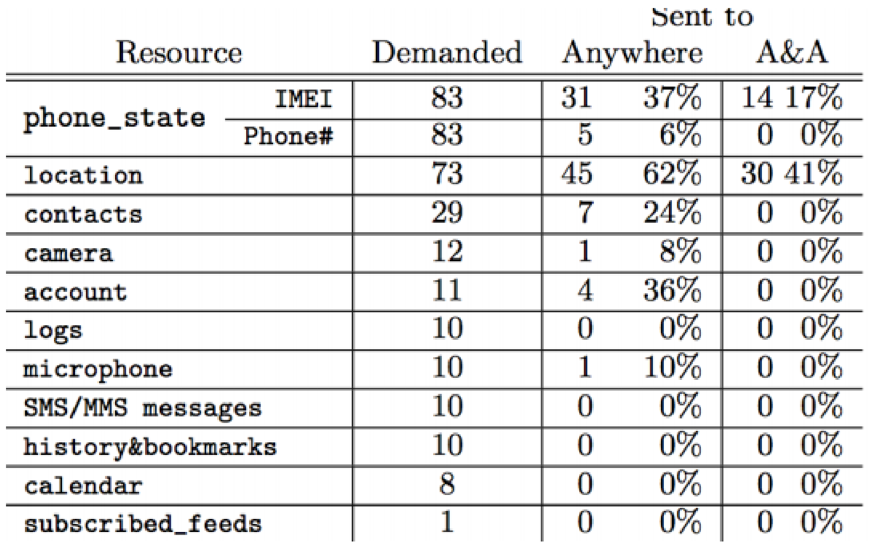
- Client/Server based solution for Static. Scans at client or build server and assessments stored centrally

A lot of academic interest…

Information Leaks

- Many apps include advertising or analytics **libraries**
- These **libraries** tend to leak user data



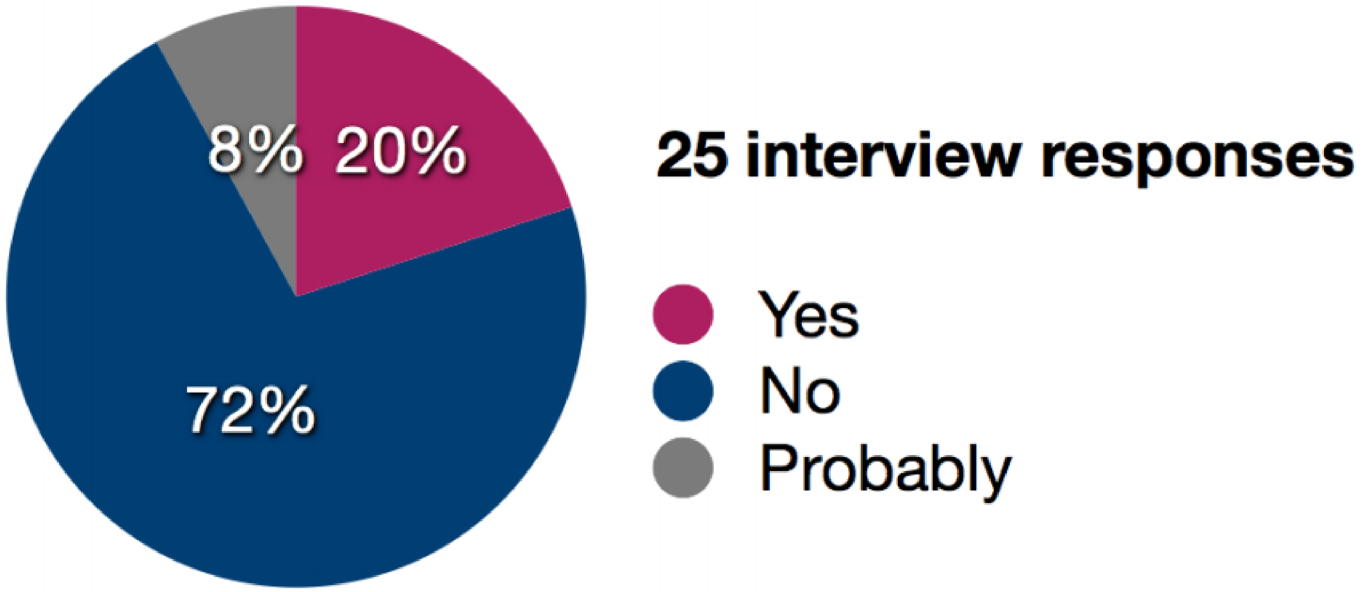


Resource		Demanded	Sent to Anywhere		A&A	
phone_state	IMEI	83	31	37%	14	17%
	Phone#	83	5	6%	0	0%
location		73	45	62%	30	41%
contacts		29	7	24%	0	0%
camera		12	1	8%	0	0%
account		11	4	36%	0	0%
logs		10	0	0%	0	0%
microphone		10	1	10%	0	0%
SMS/MMS messages		10	0	0%	0	0%
history&bookmarks		10	0	0%	0	0%
calendar		8	0	0%	0	0%
subscribed_feeds		1	0	0%	0	0%

[From Shekhar et al., “AdSplit: Separating smartphone advertising from applications”, USENIX Security 2012.]

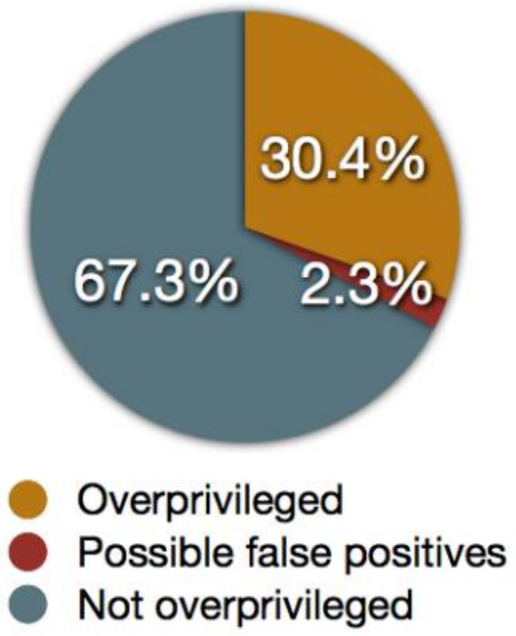
Do Users Understand Android Permissions?

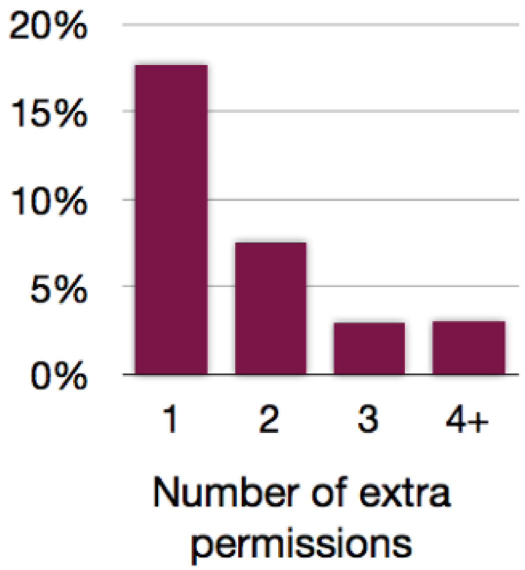
“Have you ever not installed an app because of permissions?”



From Felt et al.,
“Android Permissions: User Attention,
Comprehension, and Behavior”,
SOUPS 2012.

Many Apps are Over-Permissioned





[From Felt et al.,
"Android Permissions Demystified",
CCS 2011.]

CSE484/CSE584

CLOUD SECURITY

Dr. Benjamin Livshits

Who Are the Principals?

- User(s)
- Image/VM provider
- Cloud provider
- Who else?..

- What are the trust relationships?

Analysis of Threats on Amazon’s EC2

- Instantiated and analyzed over **five thousand** Linux and Windows images publicly provided by the Amazon catalog
- Check for a wide-range of security problems such as the prevalence of **malware**, the quantity of **sensitive data** left on such images, and the **privacy risks** of sharing an image on the cloud

1. secure the image against **external** attacks
2. secure the image against a **malicious image provider**
3. sanitize the image to prevent users from extracting and abusing **private information** left on the disk

Experimental Setup

- During our study we had continuous contact with the Amazon Web Services Security Team.
- Even though Amazon is not responsible of what users put into their images, the team has been prompt in addressing the security risks identified and described in this paper.
- Meanwhile, it has published public bulletins and tutorials to train users on how to use Amazon Machine Images (AMIs) in a secure way
- An AMI can be created from a live system, a virtual machine image, or another AMI by copying the file system contents to the Amazon Simple Storage Service (S3) in a process called bundling.
- Public images may be available for free, or may be associated with a product code that allows companies to bill an additional usage cost via the Amazon DevPay payment service.
- Thus, some of these public machines are provided by companies, i.e. SalesForce or Oracle

Images Are Old and Vulnerable

- Software running on each AMIs is often out of date and, therefore, must be immediately updated by the user after the image is instantiated
- From our analysis, 98% of Windows AMIs and 58% of Linux AMIs contain software with critical vulnerabilities.
- This observation was not typically restricted to a single application but involved multiple services: an average of 46 for Windows and 11 for Linux images

Malware in the Image

- Discovered two Windows-based infected images.
- The first machine was infected with a Trojan-Spy malware (variant 50112).
- This Trojan has a wide range of capabilities: key logging, monitoring processes on the computer, and stealing data
- Observed several images that opened connections to various web applications within and outside of Amazon EC2
- discovered two AMIs in which the
- syslog daemon was configured to send the log messages to a remote host, out of the control of the user instantiating the image.

Backdoors

- When a user rents an AMI, she is required to provide the public part of the her ssh key that it is then stored by Amazon in the authorized_keys in the home directory.
- The first problem with this process is that a user who is malicious and does not remove her public key from the image before making it public **could login** into any running instance of the AMI.
- The existence of these kinds of potential backdoors is known by Amazon since the beginning of April 2011
- Leftover ssh keys only allow people with the corresponding private key, to obtain access to the instance
- **ssh** server permits password-based authentication, thus providing a similar backdoor functionality if the AMI provider does not remove her **passwords** from the machine.
- Passwords provide a larger attack vector: anybody can extract the password hashes from an AMI

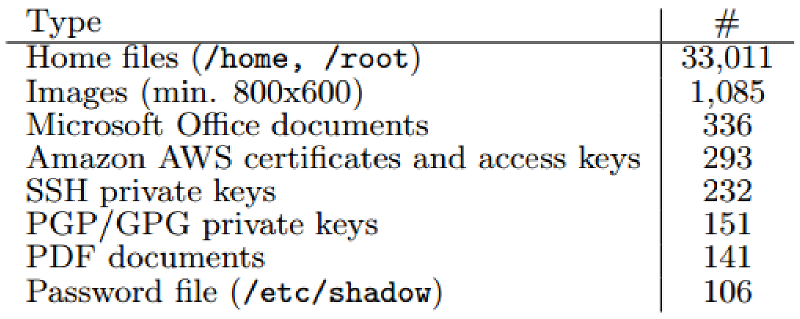
Private Info

- Our system was able to identify 67 Amazon API keys, and 56 private SSH keys that were forgotten.
-
- The API keys were mostly not password protected and, therefore, can immediately be used to start images on the cloud at the expense of the key's owner.

- Browser history found
  - 9 AMIs contained history files
  - Reveal information about image creator
- History files can easily be used to de-anonymize, and reveal information about the image's creator.
- Shell history: ~/.history ~/.bash_history, ~/.sh_history, etc.

Recovering Files

- **extundelete** and **Winundelete** to attempt to recover previously deleted files
- Able to recover files for 98% of the AMIs (from a minimum of 6 to a maximum of more than 40,000 files per AMI).
- In total, 28.3GB of data (i.e., an average of 24MB per AMI)



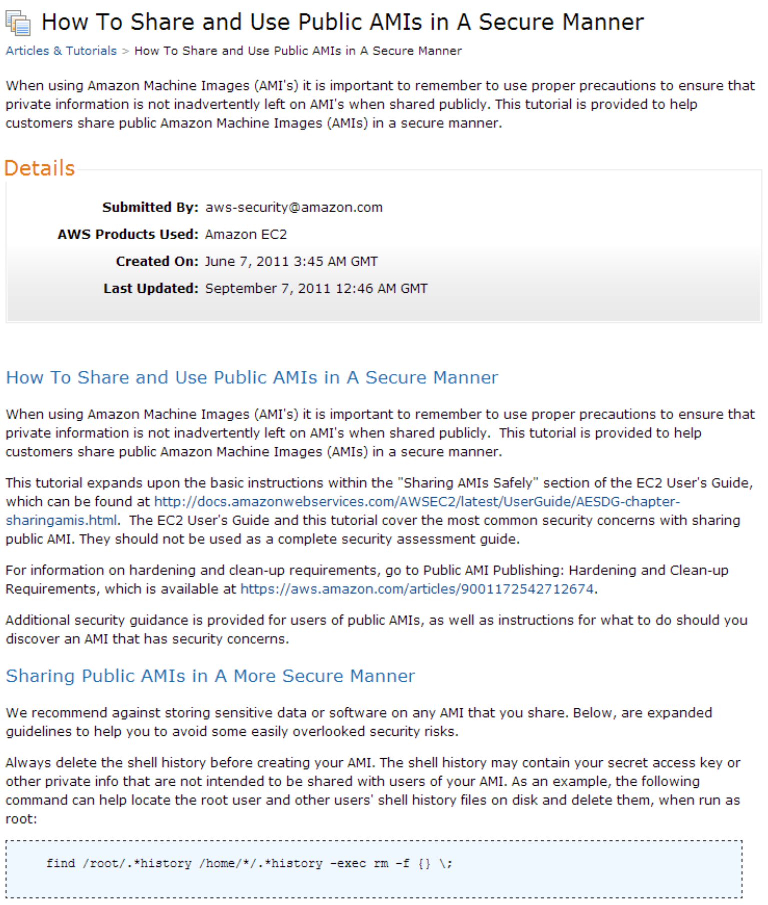
Type	#
Home files (`/home, /root`)	33,011
Images (min. 800x600)	1,085
Microsoft Office documents	336
Amazon AWS certificates and access keys	293
SSH private keys	232
PGP/GPG private keys	151
PDF documents	141
Password file (`/etc/shadow`)	106

Recovered data from deleted files

Amazon's Response

- Amazon has a dedicated group dealing with the security issues of their cloud computing infrastructure: the AWS (Amazon Web Services) Security Team.
- The security team reacted quickly, and released a tutorial within five days to help customers share public images in a secure manner.
- Contacted again Amazon on June 24th about the possibility of recovering deleted data from the public Amazon AMIs
- Amazon immediately verified and acknowledged the problem, and contacted all the affected customers as summarized by a public bulletin released on June 4th

Tutorial



How To Share and Use Public AMIs in A Secure Manner

Articles & Tutorials > How To Share and Use Public AMIs in A Secure Manner

When using Amazon Machine Images (AMI's) it is important to remember to use proper precautions to ensure that private information is not inadvertently left on AMI's when shared publicly. This tutorial is provided to help customers share public Amazon Machine Images (AMIs) in a secure manner.

Details

Submitted By:	aws-security@amazon.com
AWS Products Used:	Amazon EC2
Created On:	June 7, 2011 3:45 AM GMT
Last Updated:	September 7, 2011 12:46 AM GMT

How To Share and Use Public AMIs in A Secure Manner

When using Amazon Machine Images (AMI's) it is important to remember to use proper precautions to ensure that private information is not inadvertently left on AMI's when shared publicly. This tutorial is provided to help customers share public Amazon Machine Images (AMIs) in a secure manner.

This tutorial expands upon the basic instructions within the "Sharing AMIs Safely" section of the EC2 User's Guide, which can be found at http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/AESDG-chapter-sharingamis.html. The EC2 User's Guide and this tutorial cover the most common security concerns with sharing public AMI. They should not be used as a complete security assessment guide.

For information on hardening and clean-up requirements, go to Public AMI Publishing: Hardening and Clean-up Requirements, which is available at https://aws.amazon.com/articles/9001172542712674.

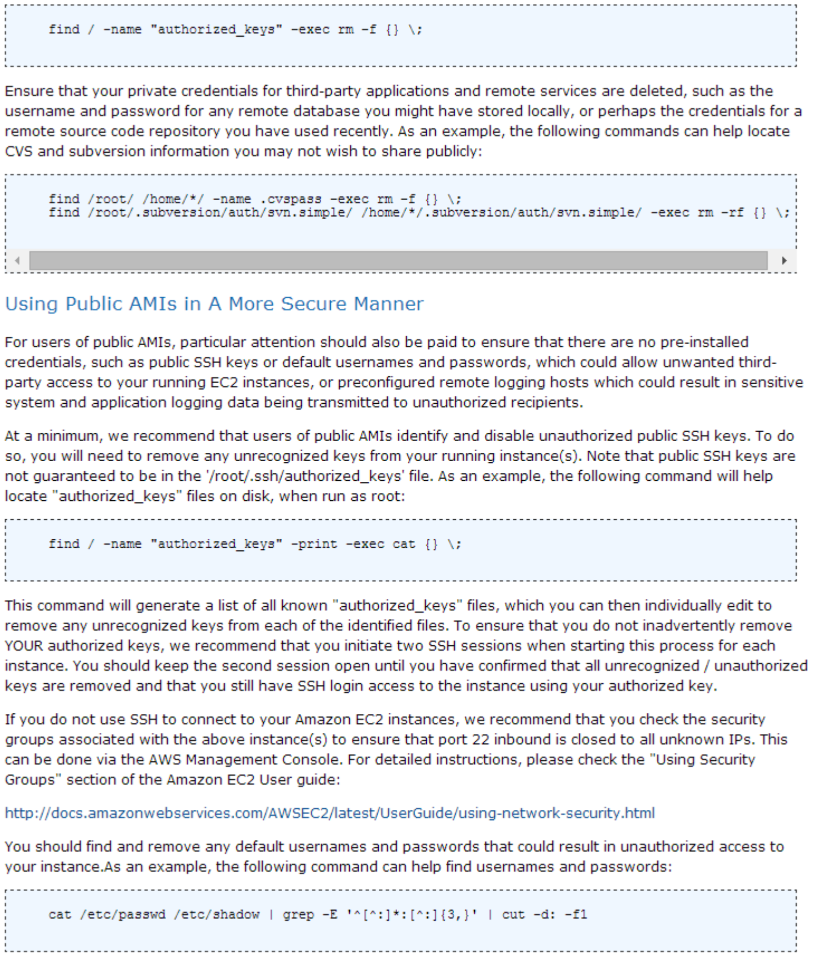
Additional security guidance is provided for users of public AMIs, as well as instructions for what to do should you discover an AMI that has security concerns.

Sharing Public AMIs in A More Secure Manner

We recommend against storing sensitive data or software on any AMI that you share. Below, are expanded guidelines to help you to avoid some easily overlooked security risks.

Always delete the shell history before creating your AMI. The shell history may contain your secret access key or other private info that are not intended to be shared with users of your AMI. As an example, the following command can help locate the root user and other users' shell history files on disk and delete them, when run as root:

```
find /root/.*history /home/*/.*history -exec rm -f {} \;
```



```
find / -name "authorized_keys" -exec rm -f {} \;
```

Ensure that your private credentials for third-party applications and remote services are deleted, such as the username and password for any remote database you might have stored locally, or perhaps the credentials for a remote source code repository you have used recently. As an example, the following commands can help locate CVS and subversion information you may not wish to share publicly:

```
find /root/ /home/*/ -name .cvspass -exec rm -f {} \;
find /root/.subversion/auth/svn.simple/ /home/*/.subversion/auth/svn.simple/ -exec rm -rf {} \;
```

Using Public AMIs in A More Secure Manner

For users of public AMIs, particular attention should also be paid to ensure that there are no pre-installed credentials, such as public SSH keys or default usernames and passwords, which could allow unwanted third-party access to your running EC2 instances, or preconfigured remote logging hosts which could result in sensitive system and application logging data being transmitted to unauthorized recipients.

At a minimum, we recommend that users of public AMIs identify and disable unauthorized public SSH keys. To do so, you will need to remove any unrecognized keys from your running instance(s). Note that public SSH keys are not guaranteed to be in the '/root/.ssh/authorized_keys' file. As an example, the following command will help locate "authorized_keys" files on disk, when run as root:

```
find / -name "authorized_keys" -print -exec cat {} \;
```

This command will generate a list of all known "authorized_keys" files, which you can then individually edit to remove any unrecognized keys from each of the identified files. To ensure that you do not inadvertently remove YOUR authorized keys, we recommend that you initiate two SSH sessions when starting this process for each instance. You should keep the second session open until you have confirmed that all unrecognized / unauthorized keys are removed and that you still have SSH login access to the instance using your authorized key.

If you do not use SSH to connect to your Amazon EC2 instances, we recommend that you check the security groups associated with the above instance(s) to ensure that port 22 inbound is closed to all unknown IPs. This can be done via the AWS Management Console. For detailed instructions, please check the "Using Security Groups" section of the Amazon EC2 User guide:

http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/using-network-security.html

You should find and remove any default usernames and passwords that could result in unauthorized access to your instance.As an example, the following command can help find usernames and passwords:

```
cat /etc/passwd /etc/shadow | grep -E '^[^:]*:[^:]{3,}' | cut -d: -f1
```