Exploiting Embedded Systems

Blackhat Amsterdam 2006



E

Barnaby Jack



Introduction

Overview

- Embedded Systems Basics
- Real Time OS
- The ARM Architecture
- The JTAG Interface
- The UART Interface
- Introduction To the ICE (In-Circuit Emulator)
- Interfacing With The Embedded System
- Reverse Engineering And Debugging
- Defeating The Watchdog
- Exploiting The Vulnerability
- Shell-code Example







Embedded Systems Are Everywhere

Automobiles, Cell-phones, Routers, Microwaves – Embedded devices are an integral part of our daily lives.

The popularity of internet-connected and wireless devices is rapidly increasing

Where there is code – There are flaws!





The Target



DI-604 Broadband Router

A popular home router – ARM9E 150Mhz processor, 1MB FLASH, 8MB SDRAM, 6 ETH PORTS, ThreadX RTOS

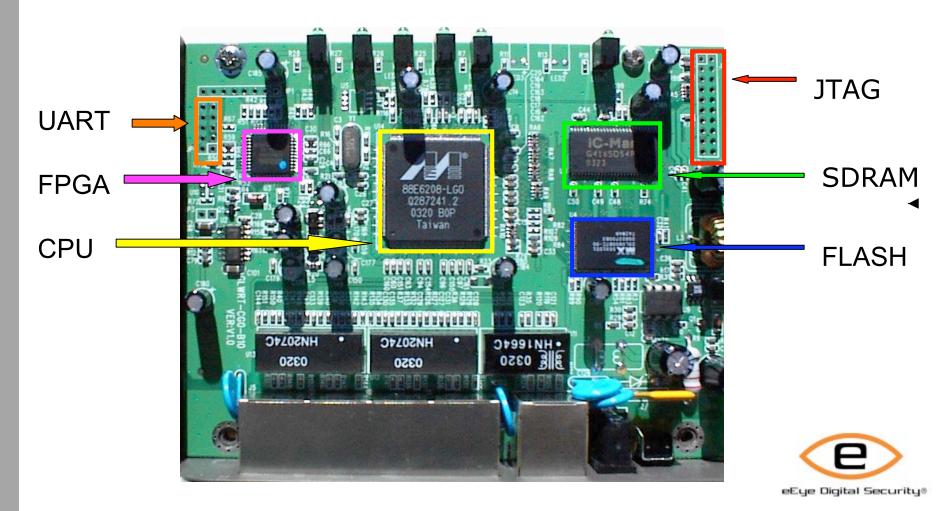




Embedded Systems Basics



The Internals – D-Link 604 Router



Embedded System OS

Thread-Based:

VxWorks, ThreadX, Integrity, etc

Process-Based:

RTLinux, OS-9

Thread-based RTOS' are widely used due to fast performance and low overhead.







The ARM Architecture



The ARM9E-S Processor

The ARM9E processor in the DLINK is a synthesizable version of the ARM9TDMI core.

Implements an extended version of the ARM instruction set

Supports full ARM architecture v5TE.



The ARM Architecture



ARM Assembly

The ARM is a typical RISC architecture with some additions:

- Control of the ALU and shifter in every data processing instruction
- Auto-increment and auto-decrement addressing modes
- Load and Store multiple instructions
- Conditional execution on all instructions

ARM has 31 general purpose 32 bit registers – 16 registers are visible at any one time.

R13 and R14 have special roles – the link register and program counter.



The ARM Architecture



ARM exception vectors

Mode	Normal address	High vector address
Supervisor	0x00000000	0xFFFF0000
Undefined	0x00000004	0xFFFF0004
Supervisor	0x0000008	0xFFFF0008
Abort	0x0000000C	0xFFFF000C
Abort	0x00000010	0xFFFF0010
IRQ	0x0000018	0xFFFF0018
FIQ	0x0000001C	0xFFFF001C
	Supervisor Undefined Supervisor Abort Abort IRQ	ModeaddressSupervisor0x00000000Undefined0x00000004Supervisor0x00000008Abort0x0000000CAbort0x00000010IRQ0x00000018



The Marvell Chipset



The Marvell 88E6218



The 88E6218 chipset is specifically designed for Router applications.

Processor is ARM9E based, running at 150MHZ, 7 Switch ports, 1MB embedded memory, 16 GPIO ports, 1 UART

Supports both the ARM and THUMB instruction set.



The JTAG Interface



Introduction to JTAG

The JTAG interface is supported on-chip.

Five dedicated signals must be provided on each chip that supports the JTAG standard.

TRST – Test-Reset: initializes and disables the test interface

- TCK Test Clock: independent timing control
- TMS Test Mode Select: controls state transitions
- TDI Test Data Input: supplies data to the JTAG registers
- TDO Test Data Output: outputs data from the JTAG registers.



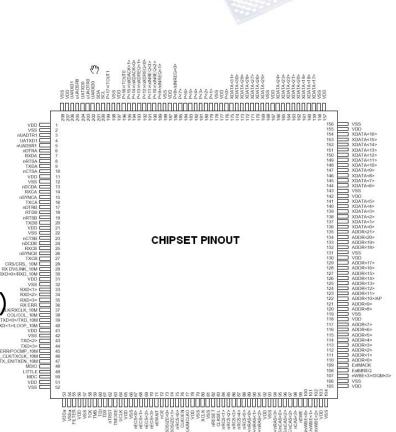
The JTAG Interface

Locating the JTAG test points

Acquire vendor chip pin-out. Use voltmeter to verify connections. Possibilities:

- Full JTAG connector on PCB
- Individual JTAG points
- No JTAG points (solder directly to chip)







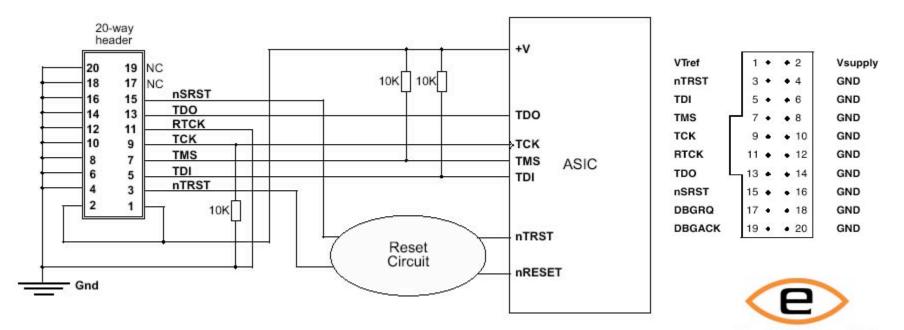
The JTAG interface



Building a JTAG Connector

If a pre-installed JTAG connection is not available, a JTAG connector must be built.

This schematic is for the ARM Multi-ICE.



eEye Digital Security®

The ICE (In-Circuit Emulator)

The ARM Multi-ICE

The Multi-ICE supports all current ARM processors: ARM7, ARM9, XSCALE, etc Fast and reliable, supported by most debuggers via RDI interface.

MuttilCE













The Serial Interface

Most chipsets will support a UART interface.

The serial interface can be used to view program output over a serial connection.

A level-shifter must be incorporated into the adapter. The USBMOD3 level shifts and outputs over USB.





```
The UART Interface
```



Simulating the UART with macros

Serial output can be viewed by setting breakpoint macros.

Set macro to execute on dbgprint, and use macro to print serial data to console

```
// UART simulator
___var sbuf;
uartsim()
{
    sbuf = __readMemory32(0x00,"Register");
    __message "UART output:", (char *)sbuf, "\n";
return 0;
}
```



Connected JTAG



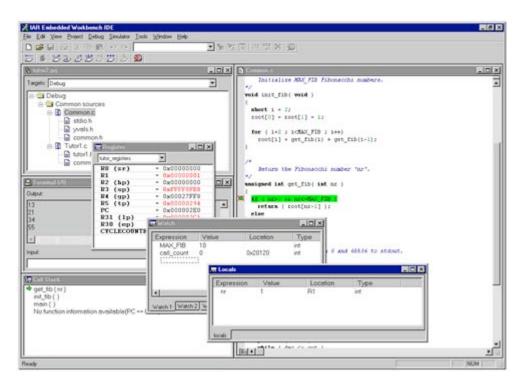




Embedded Debuggers



Unfortunately, no excellent embedded debuggers – each have their shortcomings.



We use IAR Embedded Workbench

Unfortunately problems exist: breakpoints sometimes flaky, occasional incorrect values.



Embedded Debuggers - DEMO



DEBUGGING DEMO

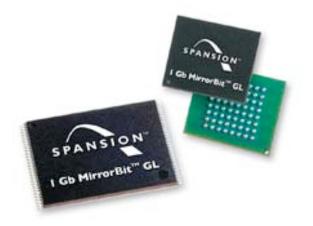


Retrieving the firmware image

Retrieve the firmware image to disassemble within IDA. Useful for locating vulnerabilities, mapping API calls, etc.

3 options:

- Rip image from flash chip via JTAG
- Dump memory to file from within debugger
 Download firmware image from Vendor (usually encoded)







Retrieving The Firmware Image

There are many standalone products that offer flash reading/writing. Debuggers often include a flash read/write option.

Most common flash memory chips are supported.

<mark>se</mark> Project - De	fault _ 🗆 🗙	<mark>, J</mark> Target n	nemo	ry (E	ntire	flash	ı chip)*														_	
Name	Value	Address: Oxi)	_	×1	x2	×4																
Connection	USB			-		_			1				1										
		Address		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCI		-		
nit JTAG speed	10 kHz	00000	10			EA	60	FØ	9F	E5	60	FØ	9F	E5	60	FØ	9F	E5				1	-
TAG speed	Auto	00010	60							E5			9F		60				1				1
AP number	<not used=""></not>	00020	47				54		43	54		2D		31	2E	30	20		GLWR				
R len	<not used=""></not>	00030	20				20		30	30		30			30	2E	30				00.0		
	500 (2010-0209/1-07).	00040	00		22	33	44		00	00		ØB		E3		00	80		"31	<i>w.</i> .	e	x	•
hip	Generic ARM7/ARM9	00050	10				9A		07		00			E3		ØF	07	EE			• • • •	• • •	•
Clock speed	<don't care=""></don't>	00060	16						2F	E1		DØ		E5		FØ		E5					- 3
Indian	Little	00070	30							00		AØ		00	84		09	00	0.0.	.0			-
heck core Id	No	00080	88				8C		09	00		AØ			94		09	00					•
RM core Id	0x0	00090	A8	0.00		00		00	00	00	00			00	00	00	00	00					•
Jse target RAM	No	000A0	00			00	00	00	00	00		00		00	00	00	00						-
AM address	0x0	000B0	00				00	00	00	00		00			00	00		00					÷.
AM size	4 KB	00000	00				00			00		00			00	00	00						•• I
se DCC mode	Yes	000D0	00			00	00	00	00	00	00			00	00	00	00	00					
		000E0	00			00	00	00	00	00		00		00	00	00	00	00					-
lanufacturer	AMD	000F0	00				00	00	00	00		00		00	00	00	00	00					-
)evice	Am29LV800BB	00100	00	00		00		00	00	00		00		00	00	00	00	00					÷.
ize	1024 KB	00110	00			00	00	00	00	00		00		00	00	00	00	00					•
lash Id	0x1225B	00120	00			00	00	00	00	00		00		00	00	00	00	00					
Check flash Id	No	00130	00			00	00	00	00	00		00		00	00	00	00	00					.
ase address	0x0	00140	00				00	00	00	00		00		00	00	00	00	00					•• I
Irganization	16 bits x 1 chip	00150	00	00				00	00	00		00		00	00	00	00	00					
-		00160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					. 1
		00170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					-
LOG																						_	
Project opened nnectina	successfully																						
	d error. Expected 0x1225E	found: 0xEOE	NEANC	i -																			
RROR: Failed	to connect																						
nnecting																							
	sh chip (19 sectors, 1 rang																						
ading entire fla:			and C	omple	ted aff	ter 37.	.354 :	sec															
ading entire fla: Target memory r	ead successfully. (104857	5 bytes, 2 rang	es) · C	ompio																			
	ead successfully. (104857	5 bytes, 2 rang	esj-c	ompio																			







Firmware Reversing

Most firmware updates are encoded/check-summed.

Decoder routine can be found by reversing the memory dump or live debugging.

The DI-604 firmware is compressed and check-summed to verify original firmware.

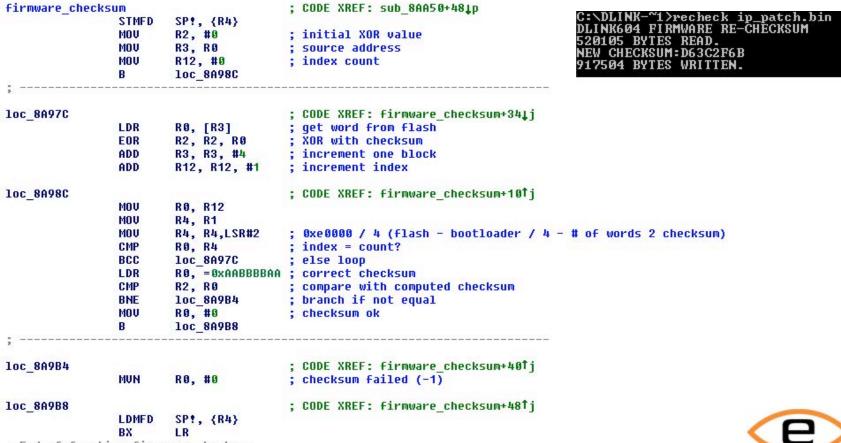
Checksum routine is simple to find by reversing "upload firmware" code snippet.

A small tool was written to patch firmware after modification – any hacked firmware may be uploaded.



Analyzing Firmware





; End of function firmware_checksum



The Watchdog Timer



Many embedded systems implement a watchdog timer.

The watchdog can be a dedicated hardware register, or may be implemented with external circuitry.

Watchdog will reset embedded system when

a debugger is attached.



The Watchdog Timer



Defeating the Watchdog

- Write to watchdog register
- Locate watchdog reference and patch
- Trap exception vector 0 (reset)





AR Embedded Workbench IDE	
Edit View Project Debug Disassembly 3-Link Tools Window Help	
■	
issembly	Quick Watch
o to Memory 💌 🗈	CPU Registers
78387844 00000000 ANDEC R0, R0, R0 R0 78387848 00000000 ANDEC R0, R0 R0 R0 78787848 00000000 ANDEC R0, R0 R0 R0 R0 78787848 00000000 ANDEC R0, R0 R0 R0 R0 78787848 00000000 ANDEC R0, R0 R0 R0 R0 78787850 00000000 ANDEC R0, R0 R0 R0 R0 78787850 00000000 ANDEC R0, R0 R0 R0 R0 78787850 00000000 ANDEC R0, R0 R0 R0 R0 78787860 00000000 ANDEC R0, R0 R0 R0 R0 78787860 00000000 ANDEC R0, R0 R0 R0 R0 7878786 00000000 ANDEC R0, R0 R0 R0 R0 7878786 00000000 ANDEC R0, R0 R0 R0 </td <td>R0 = 0xFFFFFFF R14_fiq Expression R1 = 0x0000000 R13 svc R3 R3 svc R2 = 0x0000000 R13 svc R3 R3 svc R4 = 0x54534F48 ESPSR_svc R4 R4 R6 = 0x087878003A R13_abt R6 R14_abt R7 = 0x0000000 ESPSR_svc R4 R4 R6 = 0x005250F8 R14_abt R7 R0 R7 = 0x0005000 ESPSR_std R13_irq R4 R9 = 0x0015AC00 R14_irq R14_abt R14_abt R10 = 0x00000002 ESPSR_irq R14_irq R10 = 0x00000002 ESPSR_irq R14_irq R10 = 0x00000002 ESPSR_irq R14_irq R11 = 0x0000002 ESPSR_und R14_irq R12 = 0x0000002A R14_usr R14_usr ECPSR = 0x6000013 R10_usr R0_usr PC = 0x6000013 R10_usr R1_usr R9_fiq 0x82037E5 R13_usr R10_fiq</td>	R0 = 0xFFFFFFF R14_fiq Expression R1 = 0x0000000 R13 svc R3 R3 svc R2 = 0x0000000 R13 svc R3 R3 svc R4 = 0x54534F48 ESPSR_svc R4 R4 R6 = 0x087878003A R13_abt R6 R14_abt R7 = 0x0000000 ESPSR_svc R4 R4 R6 = 0x005250F8 R14_abt R7 R0 R7 = 0x0005000 ESPSR_std R13_irq R4 R9 = 0x0015AC00 R14_irq R14_abt R14_abt R10 = 0x00000002 ESPSR_irq R14_irq R10 = 0x00000002 ESPSR_irq R14_irq R10 = 0x00000002 ESPSR_irq R14_irq R11 = 0x0000002 ESPSR_und R14_irq R12 = 0x0000002A R14_usr R14_usr ECPSR = 0x6000013 R10_usr R0_usr PC = 0x6000013 R10_usr R1_usr R9_fiq 0x82037E5 R13_usr R10_fiq
787878B0 0000000 ANDEQ R0, R0, R0 787878D4 0000000 ANDEC D0 D0 Debug Log [Build	
Go to Memory 💌 💌	
00524070 ef e	R. 9AR. 6PR. bBR. XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXX

Stack Overflow

(upnp)





Shellcode Creation

ARM9E processor supports switching to thumb state. Thumb state is very helpful for shellcode creation.

ARM:

• 32 bit instructions, word aligned

THUMB:

• 16 bit instructions, half-word aligned. (smaller code size, easier to avoid NULL bytes)

Achieve thumb state by executing BX with state bit cleared in register.



Basic Instructions

ADC	Add with carry	LDSH	Load sign-extended half-word
ADD	Add	LSR	Logical shift right
AND	AND	MOV	Move register
ASR	Arithmetic shift right	MUL	Multiply
в	Unconditional branch	MVN	Move negative register
Bxx	Conditional branch	NEG	Negate
BIC	Bit clear	ORR	OR
BL	Branch and link	POP	Pop registers
BX	Branch and exchange	PUSH	Push registers
CMN	Compare negative	ROR	Rotate right
CMP	Compare	SBC	Subtract with carry
EOR	EOR	STMIA	Store multiple
LDMIA	Load multiple	STR	Store word
LDR	Load word	STRB	Store byte
LDRB	Load byte	STRH	Store half-word
LDRH	Load half-word	SWI	Software interrupt
LSL	Logical shift left	SUB	Subtract
LDSB	Load sign-extended byte	TST	Test bits



Shellcode Possibilities

- Program the hardware directly (ref: Yuji Ukai, PacSec 2005)
- Reverse Firmware and map out API calls (dump password over port, embedded sniffer.)
- Memory patch to allow debug access to router

But we'll take a different route.





How to Own Everyone

Two phase exploitation.

- 1. Send initial exploit
- Disable admin password
- Enable remote configuration
- 2. Upload hacked firmware
- Modify firmware, re-checksum
- Modified code monitors data and injects hostile code





Initial exploit

Debug/reverse code that opens remote configuration.

100	_20BF4	; CODE XREF: ROM:00020BD81j
48 91 1F E5	LDR	R9, =dword_293154
B2 13 D9 E5	LDRB	R1, [R9,#0x3B2] ; 0x293506 (remote config flag)
00 00 51 E3	CMP	R1, #0
06 00 00 0A	BEQ	loc_20C20
4C 1C 01 EB	BL	set remote cfg
00 10 A0 E1	MOV	R1, R0
60 81 1F E5	LDR	R8, =dword_293154
A4 13 88 E5	STR	R1, [R8,# <mark>8x3A4</mark>]
00 00 A0 E3	MOV	R0, #0
BA FE FF EB	BL	save_entry
02 00 00 EA	В	loc_20C2C

Remote cfg flag: 0x293506

0 = DISABLED 1 = ENABLED





Initial exploit

Clear Administrator password:

R0, =dword 290190 ; stored password 34 01 9F E5 LDR 93 FD 01 EB sub A0010 BL 08 10 8D E2 ADD R1, SP, #8 R2, =dword 290180 2C 21 9F E5 LDR R2, R2, #0x24 24 20 82 E2 ADD R0, R2, #0x10 10 00 82 E2 ADD 8E FD 01 EB sub A0010 BL R1, R5, #0xFF FF 10 05 E2 AND R8, =dword 293154 D4 80 9F E5 LDR

Password compared at location 0x290190. Overwrite data with NULL bytes.





Initial exploit

Set up password and remote configuration flags, then directly call the save_settings routine.

Mem-patch end of subroutine to perform a soft reset. (mov pc, #0)

Save_settings is called when settings are changed via the dlink web interface.

We also can take advantage of this function $\ensuremath{\textcircled{}}$





Final exploit.

Exploit stack overflow in upnp processing, execute shellcode:

- Uses thumb state to overwrite needed memory locations (password, remote cfg)
- Overwrites the end of the save_settings routine with a branch to the reset vector
- Leaves thumb state, and branches to the save_settings routine
- Router reboots admin password cleared, remote configuration enabled.





The Injector

- Monitors all web traffic
- Waits for an executable download
- Injects custom executable code into download
- \bullet Everyone downloads EXE files $\ensuremath{\textcircled{}}$







The Injector

Considerations:

- Will need a patch location with a pointer to the IP packet
- Will need to modify each packet that meets criteria
- Will need to re-checksum packet and update fields.
- Only process HTTP traffic, and only executable downloads





The Injector

Patch Location:

Router checks for malformed IP packets – lets patch!

Patch location has a pointer to the IP packet, replace branch with a branch to our ROM code.

30 80 9D E5	LDR	R8, [SP,#0x244+var_214]	
	loc_6E6C	; DATA XREF: ROM:off_13F7B8to ; ROM:off 142194to	
00 00 58 E3	CMP	R8, #0 ; check ip length	
10 00 00 1A	BNE	loc_6EB8; jump if not 0	
04 30 A0 E1	MOV	R3, R4	
18 29 9F E5	LDR	R2, =aPacketDropped	
30 19 9F E5	LDR	R1, =aIpWithZeroLeng	
10 00 A0 E3	MOU	R0 #0×10	
3E 8F 01 EB	BL	dbgprint2	PATCH
FC 28 9F E5	LDR	R2, =dword 2900E0	
OC 10 92 E5	LDR	R1, [R2,#0×C]	
01 10 81 E2	ADD	R1, R1, #1	
OC 10 82 E5	STR	R1, [R2,#0xC]	
		(P

eEye Digital Security®



The Injector – Our ROM Code

- Check for port 80 (HTTP)
- Check headers for download
- Check content-length size
- Check for executable headers
- Inject executable code
- Create pseudo-header
- Re-checksum TCP packet
- Return to original code





INJECTOR DEMO



Conclusion



Embedded Systems open a whole new world of possibilities

They still have the "classic" bugs which are near non-existant in the software realm

I hope I've shown that exploiting hardware isn't just a "gimmick" and that the threat is real.

Thanks!.





NO TOASTERS ARE SAFE!!!





