Practical Attacks against the MSP430 BSL*

[Work in Progress]

Travis Goodspeed
1933 Black Oak Street
Jefferson City, TN, USA
travis@radiantmachines.com

ABSTRACT

This paper presents a side-channel timing attack against the MSP430 serial bootstrap loader (BSL), extending a theoretical attack with the details required for a practical implementation. Also investigated is the use of voltage glitching to attack a disabled BSL.

1. SUMMARY

The Texas Instruments MSP430 low-power microcontroller is used in many medical, industrial, and consumer devices. It may be programmed by JTAG or a serial bootstrap loader (BSL) which resides in masked ROM.

Recent versions of the BSL may be disabled by setting a value in flash memory. When enabled, the BSL is protected by a 32-byte password. If these access controls are circumvented, a device's firmware may be extracted or replaced.

In many versions of the MSP430, a password comparison routine suffers from unbalanced timing, such that processing an incorrect password takes two clock cycles longer than a correct byte. By observing external timing, it is possible to determine the correctness of individual bytes, drastically reducing the amount of time required to guess a password.[3]

This vulnerability had been previously demonstrated by the author in simulation, but it is in this paper that a practical implementation of the attack is first disclosed. Further, some early results in the use of voltage glitching attacks against the BSL are presented.

2. SERIAL BOOTSTRAP LOADER (BSL)

The BSL of the MSP430 resides in masked ROM. If an entry sequence is performed, as is depicted in Figure 1, the BSL–rather than the user application–is run. When two rising edges are observed on the TEST pin preceding the rising edge of the -RST pin that power on the chip, the BSL begins to execute instead of the user-defined application program. For those chips with dedicated JTAG pins, the same sequence is the same except that falling edges are sent on the TCK pin.[4]

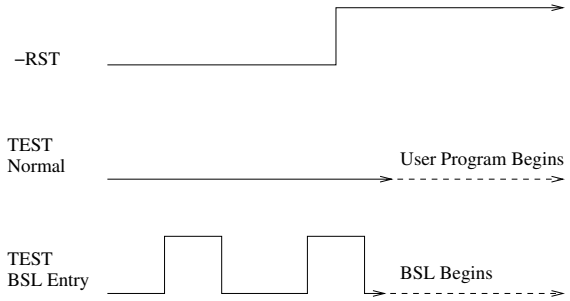

Figure 1: BSL Entry Sequence (Chips w/ Shared JTAG Pins)

As the BSL continues to function after the JTAG fuse has been blown, it is often used to allow for write-only updates without exposing internal memory to a casual attacker. For the same reason, it is a valuable attack vector. Each firmware image contains a password, and without that password little more is allowed by the BSL than erasing all of memory.

Once the BSL has loaded, commands are accepted through a bit-banged serial port. While there are many commands, the one of interest here is RX Password, which must preceed any attempt to read (TX Data) or write (RX Data) memory. Mass Erase, which bulk-erases all of memory, requires no password.

3. IVT AND PASSWORD

The BSL password is the Interrupt Vector Table (IVT) of the chip, which resides at the top of memory and is composed of sixteen 16-bit pointers to interrupt handlers. Of these 256 bits, the authors of [1] conclude that 40 are random. They then calculate that a brute force would take 128 years for a guaranteed break. This has since been reduced to 32 years in [2] by use of the Change Baud command. There might be room for further reduction, but the time required will never be so short as to be practical. Further, the method used to reduce the brute forcing time to the order of decades is only applicable to versions 1.60 and 1.61 of the BSL.

*Continuation of [3], presented by the author at Black Hat USA 2008

To appear in the Twenty-Fifth Chaos Communications Congress, December, 2008, Berlin, Germany.

```
d50:  jz 0xd56
d52:  bis #64,r11
d56:  dec r7
```

Figure 2: Byte Comparison in BSL 2.12

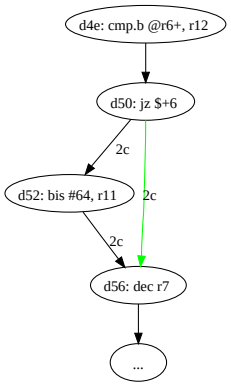

Figure 3: Control Flow of BSL 2.12 Comparison

Version 1.61 also carefully balances its timing to prevent the class of attacks presented in this paper. A cleaner solution would have been to bit-wise OR the XORing of each pair of bytes, as such a value being non-zero implies that the passwords do not match.

$$IVT = IVT' \iff \sum_{b \in IVT} b \oplus b' = 0$$

Version 2.12's comparison routine, as shown in Figure 2, suffers from unbalanced timing.[1] It is unbalanced in that one branch takes two cycles longer than the other to execute. As this code is part of a loop and the longer path is that of an incorrect byte, the timing of this program will be retarded by two cycles for every incorrect byte. The control-flow diagram in Figure 3 shows this graphically. Compare this to the invulnerable code of BSL 2.01 in Figure 4, which is has balanced timing.

[1]BSL 2.12 from the MSP430FG4618/G is used as an example throughout this paper. Others, such as 1.30 from the MSP430F1101A, are also vulnerable.

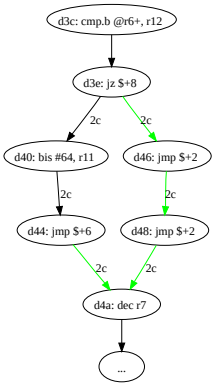

Figure 4: Control Flow of BSL 2.01 Comparison

4. SIMULATION

To demonstrate this in simulation, the author has written a C program for the MSP430 that wraps the BSL within an MSP430 simulator. The image was run 256 times, guessing passwords of every possible byte repeated. Timing was observed and recorded.

Shortened runtimes were found for repetitions of 0x00, 0x11, and 0x3A. Compared to an average (mode) runtime of 6543 cycles, a password of 0x00 repeated took only 6541 cycles to complete, a difference of 2 cycles. A password of 0x11 repeated took 6511 cycles, while 0x3A repeated took 6513 cycles. Thus the offsets were as shown in Table 1.

Guess	Cycles	Δ	$\Delta/2$
00*	6541	2	1
11*	6511	32	16
3A*	6513	30	15
all others	6543	0	0

Table 1: Runtimes of MSP430 BSL Wrapper

The rightmost column of the table gives the frequencies of each byte within the BSL. There must be a single 0x00, sixteen 0x11, and fifteen 0x3A bytes. As the less significant byte, being of an aligned instruction address, must be even, 0x11 is likely the more significant byte of each of 16 fields. Thus we have fifteen vectors of 0x113A and one of 0x1100. As the reset vector always points to the bottom of flash, it is 0x1100 and the rest are 0x113A. The BSL password for the image is shown in Table 2.

While this has demonstrated that the comparison routine itself has non-standard timing, more is required to break the BSL in practice. In particular, as the next section explains, any shift in timing will be hidden from measurement if the victim chip should wait for a start bit of a serial frame.

5. EXPLOITATION

The BSL runs at 1MHz until clocked higher[2], and a modern MSP430 can be clocked as high at 25MHz. Therefore, a 16MHz MSP430F2274 is quite capable of the timing necessary to break the password of a vulnerable chip. To that end, the author has designed a number of 'BSLCracker' boards around this chip which attack the BSL.

There are some complications, however. First, the BSL's timing is not hard-coded, but rather comes from a tare rou-

[2]By the Change Baud command, password protected after 1.61.

0x1100	0x113A
0x113A	0x113A
0x113A	0x113A
0x113A	0x113A
0x113A	0x113A
0x113A	0x113A
0x113A	0x113A
0x113A	0x113A

Table 2: Password of MSP430 BSL Wrapper

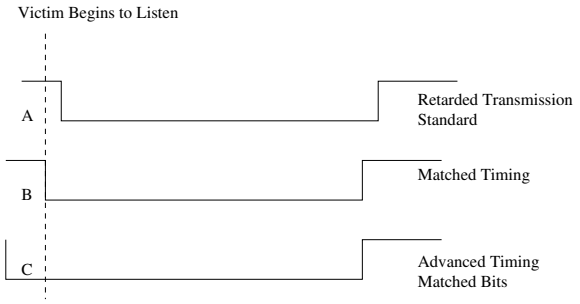

Figure 5: Three ways to say '0x80'.

tine.[3] This routine calibrates the bit-banging serial port handler by observing the timing of a header byte received by the MSP430. This header byte, 0x80, may be sent at an odd baud rate, something other than the standard 9600 baud that the BSL expects.

As Figure 5 demonstrates, there are three distinct ways in which a byte may be transmitted in a half-duplex serial port. (a) If the victim begins to listen before the start bit begins, as intended for reliable communication, all timing information is lost. There will then be no timing variance to observe. (b) If the victim begins to listen immediately after the beginning of the start bit, communication remains reliable and timing information is preserved. The attacker, however, would need to minimize the delay between the falling edge of the start bit and the victim's beginning to listen. (c) For this reason, it is practical to drop the start bit early, then gamble on the moment at which the victim begins to observe the byte. In this way, timing information is preserved whether the victim's timing is advanced or retarded from the estimate.

The technique of the preceding paragraph must be employed from the first byte of the password guess until the conclusion of the command sequence. Thus, as a checksum proceeds the password, checksum bytes must employ the same technique to avoid destroying timing information. The same is not true of header bytes, which precede the password and are thus irrelevant to the timing being measured.

6. LOCKOUTS, SELF DESTRUCTION

There are two self-protection features which have been added to the BSL in recent versions. The first is a lock-out, whereby the BSL can be disabled by placing 0xAA55 into the location named BSLKEY. The second is a self-destruct feature, where recent MSP430 chips will erase all of memory in the case of a first incorrect password attempt. The author is presently experimenting with a few methods of bypassing these restrictions.

A disabled BSL may be bypassed by voltage glitching, a technique borrowed from Smart Card 'Unlooper' technology. An R/C circuit is charged to a voltage which is significantly less than the minimum required by the victim chip. If this is timed properly, faults may be introduced into the behavior of that chip, such as the skipping of a register write-back. A scope recording of such a glitch is presented in Figure 6,

[3]See 0xE86 of BSL 2.12 from the MSP430FG4618/G.

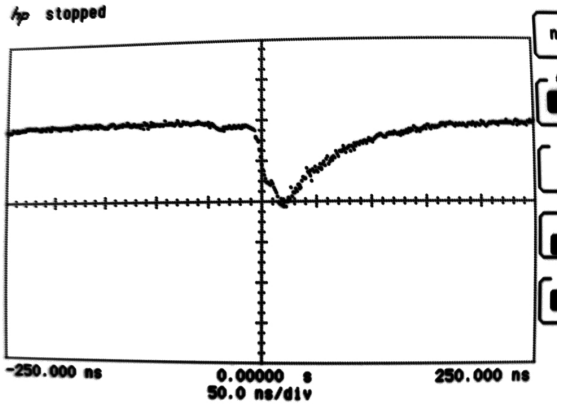

Figure 6: 45ns Voltage Glitch

```
c0c:  cmp #0aa55, &0xffbe
c12:  jz 0xc12
```

Figure 7: Disabled BSL Check, from BSL2.12

taken from the glitching of an MSP430 on the author's business card.

Reliable glitching is quite unnecessary to enter a disabled BSL, as can be seen by the code in Figure 7. When the BSL has been disabled by placing 0xAA55 into 0xFFBE, the CPU will repeatedly execute the jump instruction at 0xC12 until the watchdog timer resets the chip to the user application. During each execution of this instruction, a successful glitch will continue execution at 0xC14, rather than returning to re-execute the same instruction. An unsuccessful glitch might reset the chip, or it might have no effect. In the latter case, the attacker is free to try again on the next instruction. Should glitching become more reliable, it might be used to skip less often executed instructions within the BSL. For example, the password comparison routine might be exited after the first comparison, such that only the first byte need be correct.

While glitching register write-back is certainly the most impressive result of a changing supply voltage, there's a much less impressive effect that can be gained by setting the voltage above the minimum required for CPU operation yet beneath the minimum required for erasing flash memory. Attacks of this sort against flash memory have yet to be thoroughly investigated by the author.

7. HARDWARE AND SOFTWARE

A schematic diagram of BSLC30–the first attempt at the third major revision–is presented in Figure 9 and a photograph in Figure 8. Resistors and capacitors are added to test points for voltage glitching using the 74HC4053 MUX gates. These analog, bi-directional MUXes cut off all I/O traffic to the victim and drop voltage to a fraction of the minimum required for operation. Later revisions of the BSLC will replace the MSP430F2274 with another MSP430, one that contains a Digital to Analog converter more accurately selecting a target voltage. Further, some sort of level converter will be added to facilitate running the victim at an

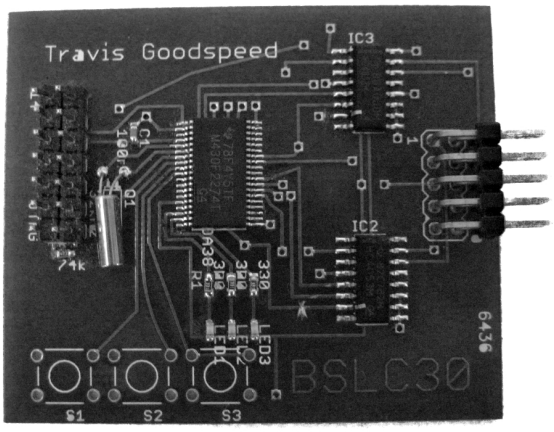

Figure 8: BSLCracker 3.0

arbitrary voltage during continued communication.

Neither the hardware nor the software of the author's implementation is complete, and further work is necessary to reliably attack the BSL. It is expected that as the software matures, the timing attack will give way to voltage glitching which–while more difficult to calibrate–is potentially effective at bypassing the access restrictions of every BSL version, even those with balanced timing.

8. RESULTS

By use of the BSLC and similar apparatus, the author has experimentally confirmed variable timing of the password-comparison routine in two of the vulnerable BSL versions, breaking a simple password by manual interpretation of timing results, similar to the method used in simulation. Further experiments have confirmed the vulnerability of the MSP430 to voltage glitching attacks by skipping over a "jz $+0" loop, both in isolation and to gain unauthorized entry to a disabled BSL.

Automated cracking of a BSL password has not yet been performed, but is expected within the next few months as the final firmware image is constructed. Glitching of MSP430 instructions other than tight loops will be the topic of forthcoming paper.

9. CONCLUSION

A side-channel timing attack against the MSP430 has been presented in sufficient detail for a practical implementation of attack. A method for using voltage glitching to enter a disabled BSL has also been presented, with citations of the code being glitched. Further, a brief outline of the author's implementation of an attack tool has been presented.

Versions of the BSL prior to 1.61 are forfeit to the timing attack, while versions after 2.01 are only vulnerable when the self-destruction feature has been disabled. Disabling the 2.x BSL by the BSLKEY flag is ineffective, and an MSP430 with a disabled BSL ought to still have a randomized password, such as by Alexander Becher's IVT randomization script.[2] As a disabled BSL erases flash memory on the first failed attempt at authentication, the timing vulnerability of recent MSP430 revisions is less serious than that of prior versions.

So far as the results of this research affect security, it should be emphasized that few if any general purpose microcontrollers are designed to defend themselves against a motivated attacker. The MSP430's insecurity, to both timing and voltage glitching attacks, does not in itself imply that similar chips from competing manufacturers are less vulnerable.

10. REFERENCES

[1] A. Becher, Z. Benenson, and M. Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *SPC 2006*, pages 104–118.

[2] T. Goodspeed. MSP430 BSL passwords: Brute force estimates and defenses, June 2008.

[3] T. Goodspeed. A side-channel timing attack of the MSP430 BSL. Black Hat USA, August 2008.

[4] S. Schauer. Features of the MSP430 bootstrap loader. TI Application Report SLAA089D, August 2006.

All code citations are from BSL 2.12 of the MSP430FG4618, Revision G, a product of Texas Instruments.

APPENDIX

A. BSL VERSIONS

Table 3 lists the BSL versions which have–and have not–been sampled by the author for vulnerability to the timing attack. All are presumed to be vulnerable to voltage glitching.

The BSL version is contained in memory at 0xFFA as a pair of bytes to be read visually. '0x02 0x12' therefore indicated version 2.12. The BSL itself spans from 0xC00 to 0x1000.

Version	MSP430	Timing
1.10		?
1.30	F1101A	Vulnerable
1.40		?
1.60		?
1.61	F1612	Invulnerable
2.01	F2274	Invulnerable
2.12	FG4618	Vulnerable
>2.12	?	?

Table 3: BSL Vulnerability by Version

B. BSL SYMBOL TABLE

It is expected that Table 4 will be valuable to those attempting to reverse engineer the MSP430 BSL from machine code. The values refer to the masked ROM of the MSP430FG4618, Revision G, from 0xC00 to 0x1000.

Address	Value/Type	Name
c00	0c06	Hard BSL Entry Point
c02	0c1e	Soft BSL Entry Point
c0c	code	Loop if Disabled
c1e	code	Watchdog Timer Disabled
c54	sub	Self-Erasure
ce0	sub	Set Main Offset
ce6	sub	Change Baud
cfe	sub	Load PC
d06	sub	Erase Segment
d0a	sub	Mass Erase
d2c	sub	Erase Check
d40	sub	RX Password
d50	sub	Password Byte Comparison
d80	sub	RX Data Block
dfa	sub	TX Data Block
e86	fn	Tare Bit Width
eec	fn	Write Byte
f54	fn	Read Byte
ff0	f46f	Chip ID
ffa	0212	Version in BCD

Table 4: BSL 2.12, FG4618/G

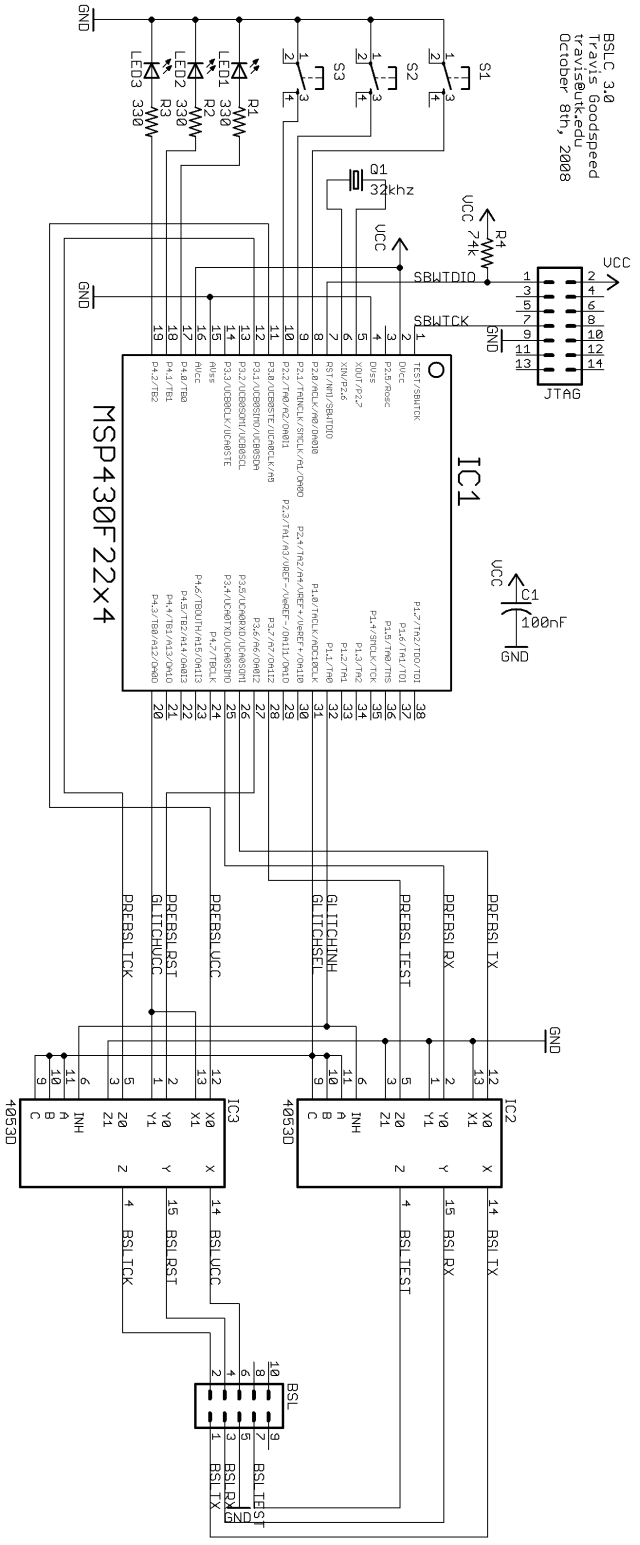

Figure 9: BSLCracker 3.0 Schematic