Software Hardening & FIPS 140

Eugen Bacic, Gary Maxwell
{ebacic@cinnabar.ca, gmaxwell@cinnabar.ca}
Cinnabar Networks Inc.
265 Carling Avenue
Suite 200
Ottawa, Ontario, Canada
K1S 2E1

Abstract: *FIPS 140-2 does not presently have security requirements that cover software-hardening techniques. Software hardening is a method of transforming an executable and associated data into a form that does not easily permit reverse engineering or reconstruction, but that has the identical execution properties of the original software. With software hardening it is possible to embed secret data such as keys and other critical security parameters (CSPs) in a hardened executable in such a way that extraction of the data is computationally very difficult or infeasible. This suggests software hardening could potentially be used as a means to ensure the integrity of the cryptographic module and as a protection mechanism for CSPs in a cryptographic module. This paper examines the possibilities of software-hardening as it applies to the ongoing FIPS efforts.*

Keywords: security, cryptography, software hardening, hardening, FIPS.

1. Introduction

In this paper we discuss some of the benefits of a software solution and the motivation for such a solution in meeting the hardening requirements documented within FIPS 140-2 [4] and why the FIPS 140-2 standard should be expanded to accommodate software solutions.

The primary reason that software is preferable to hardware is cost. In fact, many of the positive features of software can be achieved with hardware, but the cost to do so can be prohibitive. The benefits of software are that these desirable features can be obtained cheaply [1].

1.1 Ease Of Deployment

The roll out of a new software module is usually simpler than wide deployment of comparable hardware. Software can be replicated, from a technical perspective, at essentially no cost, and can be simultaneously distributed to thousands of remote locations via electronic means.

1.2 Ease of Upgrade

The ease of software deployment translates to an ease of software upgrading. Software updates, whether due to bugs, new features, or the refreshing of key material, can usually be

performed more easily and cheaply. In the event of an attempted compromise, software can usually be restored to a functional state much more quickly than hardware. Finally, the set of people that can maintain software is much larger than the set that can maintain hardware, further lowering cost.

1.3 Diversity

Three types of diversity can be accomplished with software. First, a defence in depth strategy can be developed, where several different security measures must be defeated to compromise the module. The ease of upgrading discussed above also means that a new layer can be added at will. The second type of diversity can occur between different modules (i.e. different instances of the module on different machines), each of which can have unique, yet functionally equivalent machine code. Similarly, custom solutions are much more feasible with software. Finally, run-time diversity is possible, such that a given module executes differently every time it is run (though in a functionally equivalent fashion, as before).

1.4 Generality

Software solutions can be written for general-purpose computers allowing them to be provided within interfaces that are familiar to the end-user. This can help reduce the cost of training and increase the potential set of users. Further, the generality of the platform also allows developers to leverage expertise from many areas. For instance, a programmer familiar with a common office productivity tool can embed functionality quickly thereby enhancing the security without requiring complex user interact. Furthermore, performance improvements can be obtained without modifying the software but simply by purchasing a faster computer.

1.5 Malleability

Software, ultimately being nothing more than a sequence of bits, can be reshaped in myriad ways. It can be distributed in pieces, and different parts can even be distributed at different times. It can reside at many different locations in memory and on disk, making theft more difficult than for hardware. Software can also much more easily be made transparent to the end-user, which helps with ease of use, and also has additional security benefits one of which is that it is much less likely for someone to attack something they do not realize is there. This does not, unfortunately, preclude savvy computer users from realizing what's transpiring nor attempts to crack such systems.

2. Barriers to Software Solutions

Traditional software solutions lack many of the inherent security advantages of hardware. In this section, we discuss the limitations of software. Subsequently, we show how some of these limitations can be overcome using software hardening techniques. Some of the issues, however, will be identified as potential future work.

2.1 Environment

A software module on a general-purpose computer does not exist in isolation. Even in the situation where one can ensure that the only end-user software is the cryptographic module, the computer must still contain an operating system and hardware drivers. When evaluating software against FIPS 140-2, drawing the line between the module and the environment in which it runs is a difficult task.

Of further concern is that the other software running in conjunction with the crypto module can present a security risk. Vulnerabilities in other software may allow an attacker to gain root privileges on a machine, a useful first step in compromise of the module.

2.2 Culture

In the past, software solutions were never considered for high-security environments. This has resulted in a more immature industry in verifying software correctness, reliability and availability. Also, while people inherently understand the need to be trained on new hardware, users of software security are, on the whole, less trained.

This mind-set has resulted in a high degree of scepticism around the notion of software tamper resistance, to the extent that most people are unaware of how far the technology has progressed.

2.3 Attacker Expertise

Today many software developers are insufficiently "security aware" resulting in the ability of software "crackers" to leverage this lack of awareness to subvert application and system software. Often this lack of awareness is in reality a lack of customer focus on security. This side effect of the end-user culture has resulted in more people who are interested in attacking software as it is straight-forward and cost-effective relative to attacks on hardened and protected hardware devices. As most systems utilize commercial-off-the-shelf operating systems most crackers have at their disposal copies of the very systems they wish to attack. This simply exacerbates existing security problems, as does the Internet and the speed at which cracking information gets passed along.

There now exists an extensive suite of publicly (and often freely) available attack tools for software. Hardware cracking expertise and tools are much harder to come by.

2.4 Hardware vs. Software

In the hardware vs. software debate a number of statements are usually made to the advantages hardware has over software. A typical list includes:

1. Tampering is more difficult to make "invisible";
2. More difficult to "turn" hardware away from its original purpose & design;
3. Reliability;
4. Redundancy;
5. "Crackers" needs to be *more* sophisticated in order to plug into a hardware module and "see" what it's doing or how it's used;
6. Independent of host applications or systems; and,
7. Cracker opportunities require physical access to the hardware.

Statement number 1 is *exactly* the case that software cryptography is attempting to make. Namely, that tampering with a software crypto is *just* as difficult as tampering with a hardware crypto. Statement number 2 is similar in nature to the first statement, and can be viewed as true in the case of the physical device being outside the control and influence of any other software. But this is just another form of "appliance" argument that is currently focusing the computer industry on placing technology within an appliance to shield it from inappropriate use. However, a deft attacker can still manipulate even appliances. It provides more protection when it's within a specialized box, but software crypto could be similarly protected by placing it within an appliance – and it would still benefit from the advantages software retains over hardware in terms of flexibility of upgrade/update, etc.

Statements number 3 and 6 are inherent to the very nature of hardware. However, statement number 6 can be affected by utilizing an appliance within which the software crypto is embedded, at the expense of deployment flexibility and additional cost structures due to the included hardware appliance. Redundancy (statement number 4) and reliability can usually be improved by means of distributed

computational models. This has been utilized to great effect by software vendors such as Oracle to ensure availability of crucial computational resources.

Statement number 5 is only partially true as the very nature of software crypto requires a certain amount of sophistication on the part of any attacker. The mathematical nature of what is normally attempted is beyond the scope of most people's understanding and provides a mental barrier to attack. It does not preclude someone from poking at a device, but neither does hardware. To attack the *insides* (or contents) of a hardware cryptographic device *does* require specialized equipment and expertise, but it is one of the main benefits of hardware.

Similarly, statement number 7 is an environmental, or physical security, issue. As such it's very much an issue for any deployment, regardless of whether the device is hardware or software. A threat that recurs for hardware is physical theft, something that needs environmental protection. A stolen cryptographic device cannot offer its services resulting in a totally different set of risks, while "stolen" software implies a duplication of the software – and all the issues that raises.

3. Strength of Function vs. FIPS 140-2 Level 3

The combination of the various requirements at each security level in FIPS 140-2 results in a cryptographic module with a certain overall resistance to tampering and CSP extraction. This resistance can be measured in terms of cost to the attacker, with the goal being that as the security level increases, the cost also measurably increases.

Current requirements seem to mandate a hardware solution. If one ties all the requirements back to cost to the attacker, however, it is clear that any solution resulting in an equal or higher cost should also be granted validation.

In the absence of explicit dollar amounts, we attempt to glean the spirit of each requirement, in terms of how it would make the attacker's life more difficult. One or more software hardening techniques in a similar spirit can then be proposed as an alternative way to satisfy the requirement.

There do exist software hardening methods that address the spirit of each of the requirements in the level 2 to level 3 delta. Whether or not these methods add an equivalent cost to the attacker as the prescribed hardware requirement is a question requiring further analysis.

In the following subsections, we describe each of the new requirements of level 3 over level 2. We indicate where the language of FIPS 140-2 is proscriptive against a software solution, and detail which software hardening techniques can be applied in the same vein as the prescribed hardware technique.

3.1 Cryptographic Module Ports and Interfaces

At FIPS 140-2 Level 3 and beyond, there is a requirement that "the data ports for unprotected critical security parameters are logically or physically separated from other data ports" to prohibit leakage from side-effects, including errors or program failures. The concept of a data port is somewhat nebulous in a software context. One analogue is the routine responsible for accepting these parameters from the user or from another software process.

We note that protected CSP's are not bound by this requirement. White-box cryptography allows one to pass in cryptovariable, plaintext, ciphertext and initialization vectors in an encoded state, offering protection to these parameters.

For CSP's which must be passed in unprotected, a logical separation simply implies that the routines responsible for obtaining these values can serve no other purpose nor produce side-effects that can be used to deduce information about the unprotected values. In those instances where the CSP must be passed to a subordinate function, the CSP should be passed in only a protected state to tighten the security domain. Moreover, any data structures used to hold the unprotected CSP's or information derived from the CSP's should be local to the subroutine and de-allocated immediately after use to avoid memory leakage. Indeed, this should be the case in any properly designed modular software program.

3.2 Roles, Services and Authentication

Level 3 requires identity-based operator authentication as opposed to role-based operator authentication to clearly establish the identity rather than knowledge possessed by the user. This can be achieved using two-factor authentication in which the operator must prove something they know and something about themselves such as some form of biometrics data. There are a variety of devices available for general-purpose computers that allow for this biometric authentication component, such as fingerprint readers and tablets to allow for signature authentication, or graphical passwords.

As well as operator authentication, the module should authenticate itself to the operating system, and vice versa. In this way, only authorized modules may execute and only authorized components of the operating system are permitted to interact with the module. Fixed-key white-box cryptography, node locking and code signing can all be used as elements in such an automated authentication.

3.3 Physical Security

"Tamper detection and response for covers and doors" is another Level 3 requirement, and clearly one that is hardware specific. What is its intention? An external tamper detection mechanism represents the first line of defence (except, perhaps, for policy). If an attacker cannot even get access to the module, they clearly cannot tamper with it, or extract any information from it.

In a hardened software solution, the suite of anti-debug technologies provides this first line of defense. The employment of multiple anti-debug techniques will be sufficient to dissuade a novice class of attackers, and will slow down a more advanced class, in exactly the same fashion as a tamper detection mechanism on a cover or door.

Other techniques that can be considered analogues of physical security are code transformation and code encryption. Code transformations seek to make the software module so complicated that it becomes impenetrable. Similarly, code encryption provides a nearly unbreakable shell around a program while in storage.

Techniques and tools for reverse engineering code are more widely available than those for defeating hardware protection. This implies that software cracking has a lower "admission standard", and that the number of software crackers is likely higher. It does not necessarily imply, however, that true software experts are any better at cracking software than true hardware experts are at cracking hardware.

Software can easily be copied, which can make tamper detection difficult for two reasons:

1. The software can be removed from the original host machine, and tampered with offline. In this case, tamper detection on the host is irrelevant.

2. The software can be tampered with in place, and in the event of a successful or unsuccessful tamper, the attacker can simply overwrite the tampered module with the original module.

The first attack above can be mitigated with use of node locking. The second can be made harder via operating system protection (i.e. making programs read-only). On the whole, however, tamper detection mechanisms in software are not as strong as those in hardware.

With the caveat above, it is a straightforward matter to tie any of these techniques to a tamper response mechanism, such as the zeroization of CSP's by securely overwriting areas in memory and on disk.

Hence, it is reasonable to assume that future FIPS standards would want to consider the applicability of "hardware" specifications to "software" in terms of how software obfuscation, for example, can meet the requirements as laid out under "Physical Security". Work by Chow et al [3] discuss approaches to obfuscating the control flow in computer software.

3.4 Operational Environment

FIPS 140-2 Level 3 requires "referenced Protection Profiles plus a trusted path evaluated at EAL3 plus security policy modeling." This is not a hardware or software specific requirement; we merely observe that there are configurations of commercial operating systems which have been evaluated to EAL3.

3.5 Cryptographic Key Management

"Secret and private keys established using manual methods shall be entered or output encrypted or with split knowledge procedures." As discussed above, the key enabler for this requirement is white-box cryptography.

3.6 EMI/EMC

While EMI and EMC issues are ultimately hardware related, it is possible to modulate certain types of observable emissions such as power, processor, device or video activity by software means. When a module must communicate with a hardware device, this communication should happen via the trusted operating system, and the OS should modulate the emissions to prohibit high-bandwidth channels to leak information. It may still be possible to facilitate a low-bandwidth communication channel below the modulation dampening of the operating system though it should be low enough to be impractical for the minimum length of facilitating information available to the attacker.

Side channel attacks that are feasible for software include timing attacks and fault injection. White-box cryptography, which makes use of a consistent set of lookup tables regardless of input, is resistant to timing attacks. There are no known fault injection attacks on the recommended variants of white-box DES or AES.

3.7 Design Assurance

The final new requirement at Level 3 is "implementation in a high-level language". Clearly, this is an easily satisfied requirement for a software solution.

4. Beyond FIPS 140-2

Technology progresses at an amazing rate. Documents, such as FIPS 140-2, attempt to capture "best-of-breed" technologies and techniques and ensure what is learned is re-applied correctly as time goes on. However, as software becomes more and more prevalent and "hardware" devices become no more than generic boxes running powerful operating systems performing – hopefully – a single task, documents such as FIPS 140 must evolve.

Today routers, VPNs, Firewalls, and Web servers appear on the surface to be hardware devices – often called "appliances". However, under the cover they are no more than general-purpose operating systems purposed to a specific task. Even security products like Chrysalis's LUNA SA is no more than a Linux-based system that includes special purpose hardware to perform various cryptographic machinations. And this trend will only continue as computer scientists continue to leverage *working* technology to make more and more complex solutions viable.

Locking down "software" within such appliances is where many vendors will desire to take their cryptographic devices. This will require interesting and new approaches to what has been a strictly hardware problem. Obfuscation is one possible alternative to examine and how it may meet a good many of the problems that must be met in order to *fully* meet FIPS 140-2 for the higher levels governments, in particular, require.

5. Future Work

As indicated above, we feel there is sufficient evidence that a software cryptographic module can achieve a Level 3 designation, if the terminology of FIPS 140-2 is expanded to accommodate such a solution. This work could be incorporated into efforts on FIPS 140-3.

More effort is required to determine an ideal environment for a software solution. The biggest concern is that of the lack of physical security on a general-purpose computer. What are the restrictions that must be placed on the module, the operating system, and other programs running on the computer, to ensure that the barrier to the attacker is equivalent to one where physical security is in place? A related issue is how to define the cryptographic boundary, in terms of exactly what portions of a software solution and its surrounding environment are parts of the module.

If software solutions are to be deployed in high-security environments, the users of that software must be educated, in the same fashion as they would be for secure hardware. It is vitally important to ensure that users realize that operating procedures and the security policy are just as important for software, contrary to the current mind-set that software is inherently insecure, and so no special measures are needed. Future work would involve the construction of training sessions to convey the appropriate message.

A logical and practical set of software security metrics would go a long way towards giving potential users confidence in software cryptographic modules. Metrics will likely be developed using a combination of mathematical analysis and experimentation in the form of penetration testing.

Furthermore, it is probably prudent to pursue the identification of recommended changes to FIPS to incorporate software hardening techniques. This would obviously have to discuss each of the hardening techniques in this initial study in more detail. This would also have to be coupled to the development of methodologies for reviewing and testing software hardening.

6. Circumvention Research

In 2001 both Chang & Atallah [2] and Horne et al [5] proposed advanced self-checking software-based tamper resistant integrity verification techniques. One of the most appealing aspects of their work was being able to verify the integrity of software independently of the environment. They also proposed that automatic checksumming of code during program compilation or linking would be feasible, thereby increasing the security and tamper-resistance of software.

Additional work [1, 2, 3, 5] has been performed by others that include obfuscation techniques, as mentioned earlier in this paper. These obfuscation techniques are designed to advance the strength of the tamper resistance.

However, during 2005 at least two [8, 11] papers have come to light that question the resistance to attack of the proposed systems. Although these attacks are particular to specific hardware platforms efforts must be expended to determine which platforms are susceptible to such attacks and which ones aren't. Furthermore, it is necessary to determine whether or not the vulnerable platforms can be hardened to alleviate the problems defined.

6.1 Wurster's Generic Attack

Glenn Wurster presented within his Master's Thesis at Carleton University [10] a generic attack on checksumming-based software tamper resistance. He showed, with a few lines of code, how to defeat checksumming on many modern day platforms. Although Wurster's initial work shows a simple attack capable of defeating software tamper-resistance techniques on the UltraSparc, x86, and Alpha processors more recent work by Wurster at Carleton University has shown that the ARM, AMD64 and PowerPC are also susceptible.

The worst aspects of Wurster's discovery are that:

- It is difficult to detect the attack code.
- It is feasible even where emulator-based attacks would fail.
- The attack code is generic and not program dependent.

The implications of Wurster's discovery are quite devastating. Within his thesis he describes possible solutions, although all fall short of the desired goals of tamper resistance. The only solution discovered to date is one the use of intermediate integrity checks that examine intermediate computation results. Unfortunately such a solution is prohibitively expensive computationally and difficult to add to existing solutions.

Obviously advances in static and run-time analysis of executing binaries would offer a solution. It is unknown at what cost, however.

6.2 Hyper-Threading Vulnerability

Colin Percival [6,7,8] has discovered a similar threat with the hyper-threading architecture of the Pentium 4. His attack allows an unprivileged user, on any operating system, to steal a private key that is being used on a given machine. Percival indicates that the only solution he is currently aware of, as of May 2005, is to disable hyper-threading on the Pentium 4.

7. Conclusion

The extension of FIPS into providing for pure software-based solutions for cryptography at Level 3 is something that should be properly considered and addressed. Notwithstanding the recent research that shows vulnerabilities with various obfuscation and software-based tamper resistance techniques, efforts should continue to include software-based cryptography as a viable form of cryptography up to and including Level 3. Research must be pursued that accurately

defines *exactly* what is necessary for a software cryptographic device to attain Level 3 status. The tests and other aspects of certification must be codified and solidified in such a way as to ensure that a software-based cryptographic device meets the requirements of FIPS no less stringently than does a hardware device.

The research outlined in Section 6 must be examined in detail to understand its impact on software-based cryptography in general and for Level 3 cryptography in particular.

References

[1] Eugen Bacic, Phil Eisen, Gary Maxwell, Mike Sues. *Feasibility Study: Software Hardening Techniques for Achieving FIPS 140-2 Level 3 for Cryptographic Module Validations.* Unclassified research report presented to the Communications Security Establishemnt . April 2004.

[2] H. Chang & M. Atallah. Protecting Software Code by Guards. In *Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001),* volume 2320 of *Lecture Notes in Computer Science,* pg. 160-175. Springer-Verlag, 2002.

[3] Stanley Chow, Yuan Gu, Harold Johnson, Vladimir Zakharov. An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs. In *Proceedings of the 4th International Conference on Information Security,* volume 2200 of *Lecture Notes in Computer Science,* pg. 144 – 155. Springer-Verlag, 2001.

[4] FIPS 140-2: Federal Information Processing Standard Publication 140-2: Security Requirements for Cryptographic Modules. http://csrc.nist.gov/cryptval/140-2.htm.

[5] B. Horne, L. Matheson, C. Sheehan, & R. Tarjan. Dynamic Self-Checking Techniques for Improved Tamper Resistance. In *Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001)*, volume 2320 of *Lecture Notes in Computer Science,* pg. 141-159. Springer-Verlay, 2002.

[6] Colin Percival. FreeBsd: Hyper-Threading Vulnerability. http://kerneltrap.org/node/5103.

[7] Colin Percival. Hyper-Threading Considered Harmful. http://www.daemonology.net/hyperthreading-considered-harmful/.

[8] Colin Percival. Cache Missing For Fun and Profit. http://www.daemonology.net/papers/htt.pdf

[9] P. van Oorschot, A. Somayaji, & G. Wurster. Hardware-Assisted Circumvention of Selt-Hashing Software Tamper Resistance. In *IEEE Transactions on Dependable and Security Computing,* April-June 2005.

[10] Glenn Wurster. *A Generic Attack on Hashing-Based Software Tamper Resistance.* Master of Computer Science Thesis, Carleton Univeristy. April 2005. http://www.scs.carleton.ca/~gwurster/publications/Thesis-2005.pdf

[11] Glenn Wurster, P. van Oorschot, & A. Somayaji. A Generic Attack on Checksumming-based Software Tamper Resistance. In *IEEE Symposium on Security and Privacy,* May 2005. Pg. 127-138.