Inicialização Segura de Sistemas Computacionais Dedicados

Mario T. Shimanuki e Wagner Chiepa Cunha
ITA – Instituto Tecnológico de Aeronáutica – Praça Marechal Eduardo Gomes, 50 – São José dos Campos – SP – Brasil

***Resumo* — Sistemas computacionais podem ter diferentes níveis de segurança. Muitos sistemas baseiam sua segurança e confiança única e exclusivamente nas camadas superiores (camada de aplicação), no entanto, a segurança baseada somente na aplicação não garante uma confiança nas camadas inferiores, ou seja, uma vez que a camada mais baixa esteja comprometida, todas as camadas superiores também estarão. Desta forma, em sistemas computacionais há uma real necessidade de implantar ferramentas confiáveis em todas as camadas. Este artigo descreve uma arquitetura de inicialização confiável (*Trusted Boot*) em sistemas computacionais, baseada nas especificações TCG (*Trusted Computing Group*) e na arquitetura AEGIS.**

***Palavras-chaves* — Computação segura, *boot* confiável e *Root of Trust*.**

I. INTRODUÇÃO

Sistemas computacionais dedicados são sistemas em que é executada uma tarefa (ou grupo de tarefas) em um *hardware* específico. São exemplos de sistemas dedicados: urnas de votação eletrônica, terminais ATM (*Automatic Teller Machine*), TFL (Terminais Financeiro-Lotéricos), etc.

Normalmente esses tipos de aplicações demandam um grande interesse (político, econômico, estratégico, pessoal, etc.), desta forma, é de vital importância a garantia da aplicação que está sendo executada nestes sistemas, visto que, caso esteja sendo executado um aplicativo malicioso, poderão ser coletadas informações sensíveis (dos usuários ou do sistema).

O TCG (*Trusted Computing Group*) [1] é uma organização criada para produzir, definir e promover especificações abertas para o desenvolvimento de plataformas computacionais seguras. Formado por um consórcio entre grandes empresas fabricantes de *softwares* e dispositivos, tais como a AMD, HP, IBM, Infeneon, Intel, Lenovo, Microsoft, Sun, e outros, tem como objetivo desenvolver o conceito e a especificação de uma plataforma computacional confiável.

O TCG criou a especificação para o TPM (*Trusted Module Platform*) [2, 3 e 4]. O TPM efetua funções criptográficas, de segurança, além da possibilidade de ser o *Root of Trust*, que é à base de confiança de sistemas computacionais.

Atualmente o TPM é produzido pelas seguintes companhias: Atmel; Broadcom; Infineon (Infineon TPM); Intel (Intel Manageability Engine - iTPM); Sinosun; Nuvoton (Winbond) STMicroelectronics; e ITE (ITE TPM).

Mario T. Shimanuki, explorer@ita.br, Wagner Chiepa Cunha, chiepa@ita.br, Tel +55-12-3947-5878, Fax +55-12-3947-6930.

Conforme [5], originalmente estas especificações foram elaboradas para Computadores Pessoais (PC), e posteriormente o grupo estendeu essas especificações para outros sistemas, tais como PDA (*Personal Digital Assistant*), celulares, dispositivos de impressão, sistemas dedicados, etc.

Antes da formação do TCG, [6] já havia descrito uma arquitetura para uma inicialização segura denominada AEGIS. Esta arquitetura tem por objetivo aumentar a segurança do processo de inicialização de forma a garantir que:

1. a integridade das camadas mais baixas seja verificada;
2. a transição para a camada superior ocorra somente se a camada inferior estiver integra.

Desta forma, existe a garantia da cadeia de integridade do sistema. Em [7 e 8] são apresentadas implementações da arquitetura AEGIS.

II. COMPUTAÇÃO CONFIÁVEL

Uma entidade é confiável se sempre comporta-se como esperado [9]. Uma plataforma confiável jamais irá prover uma plataforma 100% segura [10].

Embora muitos problemas de segurança possam ser resolvidos com o fortalecimento do Sistema Operacional (SO) e do aplicativo, é impossível implementar um processo de *boot* confiável puramente em *software* [11]. Para se ter uma computação confiável é necessária uma "Base de confiança computacional" [12].

São atributos da Computação Segura [13]:

- **Separação de processos**: determinadas entidades somente terão acesso a determinadas partes do sistema;
- **Proteção das informações armazenadas**: terceiras partes não têm acesso às informações armazenadas;
- **Identificação da configuração**: tem como objetivo identificar a configuração da plataforma;
- **Verificação da integridade**: detecta se o sistema encontra-se integro;

- **Confiança baseada em *hardware***: provê mecanismos baseados em *hardware* para ser o *Root of Trust* (camada mais baixa de confiança) de um sistema.

Ao inicializar um sistema computacional, as seguintes etapas devem ser percorridas até a execução de uma aplicação:

Fig. 1. Inicialização de um sistema computacional

Atualmente grande parte dos sistemas preocupa-se única e exclusivamente com a camada de aplicação, ou seja, toda a segurança/confiança está baseada nesta camada. No entanto, caso ocorra um comprometimento nas camadas mais baixas, todo o sistema estará comprometido (Fig. 2).

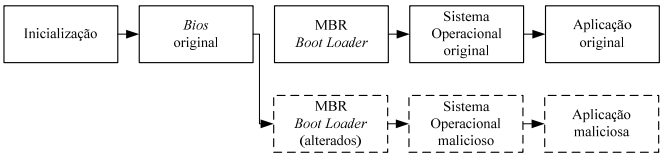

Fig. 2. Sistema redirecionado para executar uma aplicação maliciosa

A Fig. 2 ilustra um sistema em que os arquivos de inicialização (MBR e/ou *Boot Loader*) foram alterados. Uma vez que esta camada tenha sido modificada, nada garante que o SO que é carregado seja o original, consequentemente não há garantia de que o aplicativo que é executado seja o original.

Este tipo de ataque pode ser feito, por exemplo, em terminais ATM (*Automatic Teller Machine*), onde o atacante deseja obter informações do cartão bancário juntamente com a senha do usuário.

Desta forma, faz-se necessária a garantia de integridade em todas as camadas. Isto é conseguindo com base na confiança da camada superior na primeira camada logo abaixo, e assim sucessivamente. A camada (confiável) mais baixa é o *Root of Trust* do sistema.

A Fig. 3 ilustra um processo de inicialização no sistema operacional *Linux*.

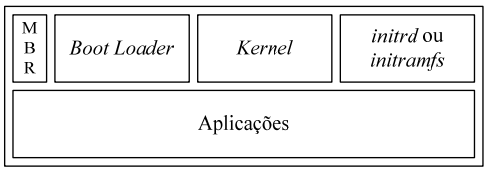

Fig. 3. Processo de inicialização do sistema operacional *Linux*

O MBR são os primeiros 512 *bytes* do setor de *boot* de uma unidade particionada de disco (o setor de *boot* de uma unidade não particionada é chamada *Volume Boot Record*). O MBR contém a estrutura organizacional do disco, esses 512 *bytes* são as primeiras informações a serem lidas no disco. O *Boot Loader* contém a listagem dos Sistemas Operacionais (SO) disponíveis no disco bem como a sua localização (partição). São exemplos de *Boot Loader* o GRUB (*GRand Unified Bootloader*) [14] e o LILO (*LInux LOader*) [15].

Uma vez selecionado o SO, o *Boot Loader* irá apontar para o setor deste SO e executar o "*kernel*" e o "*initrd* ou o *initramfs*" (dependendo a distribuição do *Linux*). O *initrd* ou o *initramfs* é um sistema de arquivos temporário usado pelo *kernel* durante o processo de *boot*. Eles são utilizados para carregar parte do *kernel* antes do sistema de arquivos ser montado. Finalizado a carga do SO, o controle pode ser passado ao aplicativo.

Para uma carga segura do sistema operacional, do ponto de vista do *software*, os arquivos que devem estar integros são o "*kernel*" e o "*initrd* ou o *initramfs*". Uma vez que estes estejam integros o aplicativo pode ser carregado.

De acordo com [16] existem as seguintes formas de prover confiança em um sistema computacional:

- um núcleo de segurança no sistema operacional;
- um núcleo de segurança em cada aplicativo;
- *BIOS (Basic Input/Output System)* seguro, evitando que programas diferentes tenham acesso ao mesmo espaço de memória;
- um dispositivo de monitoramento agregado à CPU, capaz de fornecer e armazenar funções de segurança;
- uma infra-estrutura para acesso a servidores de segurança *online* integrando *hardware* e *software* para a obtenção de atestados remotos.

O sistema deverá armazenar um *hash* [17] do estado da máquina após a inicialização. Esse *hash* é calculado usando detalhes do *hardware* e *software* da máquina. Se o *hash* calculado for autêntico (pertencerem ao *hardware* e *software* "original/confiável"), o sistema fornecerá ao sistema operacional as chaves criptográficas necessárias para acessar os dados e executar aplicativos confiáveis. Se o *hash* calculado não for autêntico, o sistema não liberará as chaves e a máquina será capaz apenas de executar aplicativos não confiáveis e de acessar material desprotegido.

Nesta linha, para suprir um sistema computacional confiável algumas alternativas são:

1. ***BIOS* seguro**: desenvolvimento de um *BIOS* que somente seja carregado em determinadas condições (presença de determinados *hardwares* ou trechos de *software*);
2. ***Token* com ID único**: após o carregamento do SO verificar por algum *token*, que deve ter um ID único, e somente carregar o aplicativo após esta confirmação;
3. **Sistema desafio-resposta**: após o carregamento do SO, sob o controle de um sistema baseado em *hardware*, a inicialização deve prosseguir somente em determinadas condições (presença de determinados *hardwares* ou trechos de *software*).

A primeira alternativa *BIOS* Seguro, altera o *BIOS* para que este seja o *Root of Trust* do sistema, é indicada para sistemas novos que estão sendo desenvolvidos e tenham uma demanda muito grande, visto que o custo de se criar um *boot* seguro é bem alto.

Por outro lado "*Tokens* com ID único" tem um custo bem baixo e é indicado para prover um *Root of Trust* que pode ser movido de um sistema para outro. Existe a resalva de que temos que garantir que o SO carregado seja integro, visto que a verificação do *token* se dará apenas após a carga do SO.

A última alternativa (Sistema desafio-respota) oferece uma flexibilidade bem grande quanto a sua implementação. Não é intrusivo, pois não estamos alterando o *BIOS* do sistema e tão seguro quanto a implementação de um *BIOS* seguro.

Os itens acima mencionados têm como objetivo incorporar o *Root of Trust* de um sistema (base de confiança). Um outro ponto que deve ser observado no projeto de um sistema confiável é a segurança do sistema.

Dentre as características desejáveis quanto a segurança/ confiança de um sistema de inicialização confiável, podem ser citados:

Confidencialidade e integridade do *software*: utilizar mecanismos de segurança (criptografia e funções *hash*) em setores ou na totalidade da unidade de armazenamento;
Integridade do *hardware*: armazenar em um registrador (externo), a situação atual do *hardware*. Caso haja algum dispositivo "não integro" o SO e/ou a aplicação não será executado;
Disponibilidade do sistema em locais pré-definidos: parte das informações armazenadas na unidade de armazenamento somente será acessada se e somente se estiverem em locais pré-definidos.
Root of Trust: suportar uma base de confiança para o sistema que deve ser baseado em *hardware*.

A "Disponibilidade do sistema em locais pré-definidos", tem como objetivo criar um sistema em que este é auto-suficiente para que consiga obter as coordenadas, via GPS (*Global Positioning System*), em que está situado, com uma atualização constante do último sinal de coordenadas. O sistema somente irá executar parte das funcionalidades se estiver em uma região pré-definida.

O *Root of Trust* jamais pode ser baseado em *software*, visto que os *softwares* podem ser duplicados.

III. Arquitetura para Inicialização Segura

Diante do exposto, é incontestável a necessidade de prover uma base de confiança em um sistema, com o intuito de manter a confiança/segurança do sistema.

Para prover uma inicialização segura, uma das alternativas é utilizar o TPM como *Root of Trust*, no entanto, atualmente não existem TPM desenvolvidos no Brasil. Tendo-se em vista que este é um "sistema fechado", este não pode ser auditado. Desta forma é apresentada uma proposta de arquitetura para inicialização segura baseado nas especificações TCG e na arquitetura AEGIS.

A Fig. 4 ilustra o diagrama de blocos do RoT (*Root of Trust*) do sistema computacional.

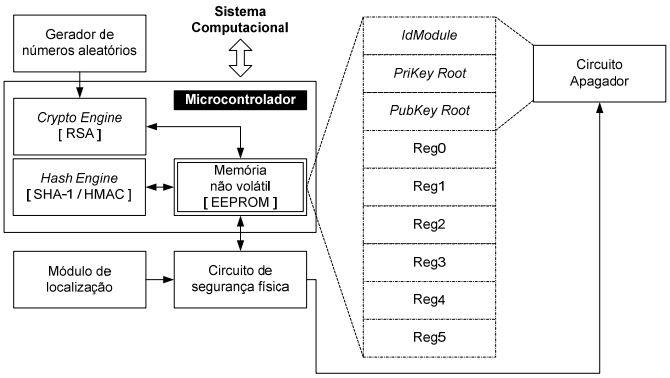

Fig. 4. *Root of Trust* (RoT) do sistema computacional

O RoT é um módulo independente baseado em um microcontrolador de baixo consumo, por exemplo o MSP430F123 (microcontrolador de 16 *bits*, *ultra-low-power* da empresa *Texas Instruments*) [18] .

O módulo é composto por um "gerador de números aleatórios", baseado em um evento físico com alta entropia (ruído térmico, desintegração radioativa, etc.). O gerador de números aleatórios é utilizado para a geração de chaves assimétricas pelo *Crypto Engine*. O *Hash Engine* irá calcular funções *hash* do *hardware* e *software* e armazena-los em uma memória não volátil (EEPROM), em determinadas posições (Reg0 – Reg5).

Quanto o RoT for inicializado, o *Crypto Engine* irá criar um identificador único (*idModule*), além do par de chaves (*PriKey Root* e *PubKey Root*). Sendo que o *PriKey* jamais irá sair do RoT.

O "módulo de localização" irá fornecer ao "circuito de segurança física" as coordenadas do sistema (via GPS), e verificar se este encontra-se em uma localidade na qual foi programado para operar, além de prover meios para verificar a detecção de violações físicas (tais como, monitoração de chaves físicas, análise de temperatura, falta/alteração do nível de energia, movimentação, etc.).

O circuito apagador irá apagar parte da memória não volátil (*IdModule*, *PriKey Root* e *PubKey Root*), caso ocorra alguma violação (por exemplo, tentativa de ataque).

A Fig. 5 apresenta um diagrama de blocos da arquitetura para a inicialização segura utilizando o RoT.

Na arquitetura proposta, é utilizado um cartão *Secure Digital* (*SD Card*), protegido somente para leitura. Neste são armazenados o MBR, *Kernel* e o *initrd* ou *initranfs*. A inicialização do sistema ocorrerá a partir desta unidade. Todos os demais *softwares* estarão armazenados, de forma cifrada, na unidade de disco principal (*Hard Disk*).

Caso o sistema não esteja em uma localização permitida (indicada pelo "módulo de localização", esta nova localização será armazenada no registro de *logs* da "Mídia de Armazenamento Portátil (MAP)" e o sistema será desligado).

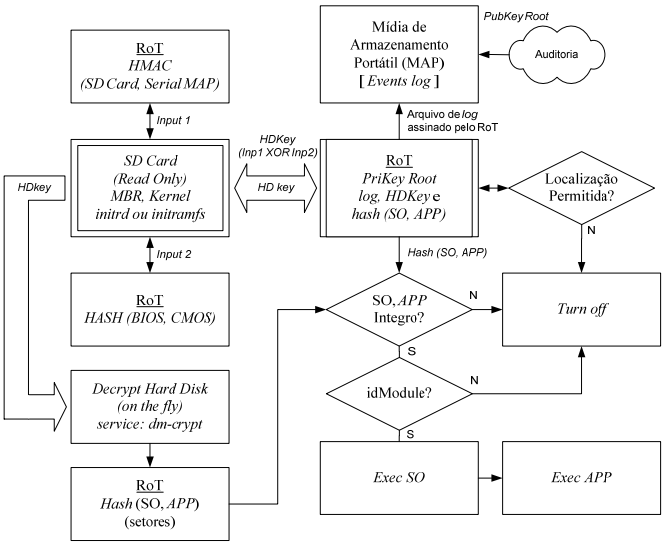

Fig. 5. Arquitetura para inicialização segura

O RoT irá registrar todos os eventos do sistema e assiná-lo com a chave privada (*PriKey*) para que o sistema possa ser auditado.

O RoT utilizará para decifrar a unidade de armazenamento principal os seguintes parâmetros:

$$input1 = HMAC(SDCard, SerialMAP) \quad (1)$$

$$input2 = Hash(Bios, CMOS) \quad (2)$$

Onde HMAC é uma função *hash* com chave secreta [19].

A partir de (1) e (2), a chave para decodificar o sistema de disco, será dado por:

$$HDKey = input1 \oplus input2 \quad (3)$$

Desta forma, estamos garantindo que o SO e o APP somente serão decifrados se satisfeitas a condição de presença/ integridade do *SD Card*, MAP, *Bios* e *CMOS*. Caso a unidade de disco seja instalada em outra máquina, este simplesmente continuará cifrado, e não será carregado.

Com a chave *HDKey*, a unidade de disco poderá ser decifrada (através o serviço *dm-crypt* [20]). O RoT irá verificar a integridade do SO e do APP através do cálculo da função *hash* em determinados setores, e confrontando com os dados armazenados no registrador da memória não volátil (registradores Reg0-Reg5). Se estes não estiverem integros, o sistema é desligado. Caso contrário é verificado se o *idModule* está presente e verificado se o sistema corresponde ao *hash* armazenado nos registradores. Se tudo estiver a contento, é carregado o SO e posteriormente o aplicativo.

Garantimos desta forma, que ocorra a carga do aplicativo baseado no RoT (*Root of Trust* do sistema). Em caso de qualquer alteração em qualquer uma das camadas, o aplicativo não será carregado.

IV. Conclusões

Com base nas especificações TCG e na arquitetura AEGIS, foram apresentados mecanismos para aumentar a segurança/confiança do mesmo. Apesar do TCG ser uma especificação aberta, algumas informações são divulgadas somente para os membros do TCG.

A arquitetura de inicialização confiável em sistemas dedicados procura eliminar a lacuna que existe na confiança das camadas inferiores a camada de aplicação, impedindo a inicialização caso estes não estejam íntegros.

Referências

[1] Trusted Computing Group. **TCG Web-Page**. Disponível em: <http:// www. trustedcomputinggroup.org>. Acesso em: 01/07/09.

[2] Trusted Computing Group. **TPM main: part 1 design principles (specification version 1.2 level 2 revision 103)**. Disponível em: < http://www.trustedcomputinggroup.org/files/resource_files/ACD19914-1D09-3519-ADA64741A1A15795/mainP1DPrev103.zip>. Acesso em: 01/07/09.

[3] Trusted Computing Group. **TPM main: part 2 tpm structure (specification version 1.2 level 2 revision 103)**. Disponível em: < http://www.trustedcomputinggroup.org/files/resource_files/8D3D6571-1D09-3519-AD22EA2911D4E9D0/mainP2Structrev103.pdf>. Acesso em: 01/07/09.

[4] Trusted Computing Group. **TPM main: part 3 commands (specification version 1.2 level 2 revision 103**. Disponível em: < http://www.trustedcomputinggroup.org/files/static_page_files/ACD28F6C-1D09-3519-AD210DC2597F1E4C/mainP3Commandsrev103.pdf>. Acesso em: 01/07/09.

[5] Trusted Computing Group. **TCG architecture overview**, version 1.4. Disponível em: <http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf>. Acesso em: 01/07/09.

[6] Arbaugh, W. A.; Farber, D. J.; Smith, J. M. **A secure and reliable bootstrap architecture**. IEEE Symposium on Security and Privacy. P. 65-71. 1997.

[7] Suh, G. E; O'Donnell, C. W.; Devadas, S. **AEGIS: a single-chip secure processor**. Information Security Technical Report. p. 63-73. 2005.

[8] Suh, G. E, Clarke, D. Gassend B., Dijk, M. van; Devadas, S. AEGIS: architecture for tamper-evident and tamper-resistant processing. ACM Press. p. 160-171. 2003.

[9] Kinney, S. L. **Trusted platform module basics**. Burlington: Newnes, 2006.

[10] Balacheff, B., Chen, L., Pearson, S., Plaquin, D., Proudler, G. **Trusted computing platform**. New Jersey: Prentice Hall PTR, 2003.

[11] Kursawe, K; Schellekens, D; Prennel, B. **Analyzing trusted platform communication**. Disponível em: <https://www.cosic.esat.kuleuven.be/publications/article-591.pdf>

[12] Hendricks, James; Doorn, L. van. **Secure bootstrap is not enough: shoring up the trusted computing base**. Proceedings of the Eleventh SIGOPS European Workshop. p. 1-5. 2004.

[13] Challener, D., Yoder, K., Cathermn, R., Sfford, D., and Doorn, L. V. **A practical guide to trusted computing**. New York: IBM Press, 2008.

[14] Free Software Foundation. **GNU GRUB - GNU Project**. Disponível em: <http://www.gnu.org/software/grub/>. Acesso em: 01/07/09.

[15] Freshmeat. **LILO Boot loader Project**. Disponível em: <http:// freshmeat.net/projects/lilo/>. Acesso em: 01/07/09.

[16] Anderson, R. J. **Security engineering: a guide to building dependable distributed systems**. New York: John Wiley & Sons, 2008.

[17] National Institute of Standards and Technology. **Fips pub 180-3: Secure hash signature standard**. Disponível em: < http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>. Acesso em: 01/07/09.

[18] Texas Instruments. **MSP430x12x mixed signal microcontroller**. Disponível em: <http://www.datasheetcatalog.org/datasheet/texasinstruments/msp430f123.pdf>. Acesso em: 01/07/09.

[19] National Institute of Standards and Technology. **Fips pub 198: the keyed-hash message authentication code**. Disponível em: < http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>. Acesso em: 01/07/09.

[20] dm-crypt. **dm-crypt: a device-mapper crypto target**. Disponível em: <http://www.saout.de/misc/dm-crypt/>. Acesso em: 01/07/09.