ARTIGO

OS DESAFIOS DA COMPONENTIZAÇÃO PARA A SEGURANÇA E A FORMAÇÃO DE EQUIPES

POR
Roberto Gallo
gallo@kryptus.com

A segurança de um sistema sempre depende da segurança de seus componentes. Esta afirmação, apesar de aparentemente simples e respaldada pelo senso comum, esconde um universo de fatores que desafiam mesmo as mais competentes equipes no projeto, desenvolvimento, deployment e manutenção de sistemas computacionais compostos em os manter seguros durante todo o seu ciclo de vida.

Um exame da literatura, dos repositórios de vulnerabilidades conhecidas (vide os CVE mantidos pelo MITRE) e mesmo da mídia sobre o tópico nos permite identificar casos significativos nos quais algum aspecto da componentização de sistemas deu causa a problemas de segurança que possivelmente poderiam ser evitados com uma maior difusão dos conhecimentos em segurança para as equipes de projetos de sistemas computacionais. Alguns exemplos marcantes incluem:

1. os ataques de cadeia logística sobre o Orion da SolarWinds[1] e sobre o XZ outbreak (CVE-2024-3094) onde estes componentes utilizados por outras soluções foram subvertidos, comprometendo os sistemas que os utilizam;
2. os ataques de canais colaterais Downfall sobre CPU Intel (CVE-2023-12301), Inception sobre CPU AMD (CVE-2023-12302) e mais recentemente o ataque GoFetch sober CPU Apple Mx (2024) que permitiram o vazamento de informações (chaves criptográficas) entre processos (componentes) de um sistema;
3. os casos de arquitetura inadequada em conjunto com má configuração de buckets S3 da Amazon que deram origem a célebres vazamentos de dados como o de 1TB da Attunity em 2019 e de 100 milhões de clientes da Capital One em 2019.

Durante este artigo, exploraremos as comunalidades de diversos casos de falhas de segurança, aparentemente bastante distintas, mas que possuem como causa raiz, e, portanto, respostas, na organização e na ação coordenada dos diversos times responsáveis pelo desenvolvimento, aquisição, integração e testes de sistemas e de seus componentes associados.

1 https://www.sans.org/blog/what-you-need-to-know-about-the-solarwinds-supply-chain-attack/

Alguns problemas da componentização

A modularização de sistema na forma de componentes possui inúmeros benefícios amplamente conhecidos e difundidos tanto na academia como na indústria, em particular a facilitação da manutenção e dos testes, a reusabilidade, redução de custos e de riscos e a escalabilidade do sistema e do processo de desenvolvimento [1].

Estes benefícios, no entanto, não escapam dos fundamentos teóricos intrínsecos da computação, nem de fenômenos concretos comuns que são fontes de ameaças, e os quais, surpreendentemente, não estão incorporados no *mindset* de muitos profissionais egressos do ensino superior em Computação. A seguir, de forma sumária por conta do limite de espaço desta publicação, elenco alguns dos principais:

1. O **Teorema de Rice**, fundamental da computação, diz que qualquer propriedade não-trivial dos programas é indecidível. Isso significa que não há um algoritmo geral que possa decidir para qualquer programa e qualquer entrada possível para ele, se ele possui uma determinada propriedade comportamental que, por exemplo, não permita a execução de alguma operação insegura.

   Na prática, isso significa que testes de segurança de programas têm assertividade limitada, baseados em casos e necessariamente apoiado por métodos heurísticos.

   Além disso, apesar de clássico e

amplamente conhecido, ainda sim muitos desenvolvedores mantêm certa ilusão de que algumas tecnologias são "balas de prata", a exemplo de plataformas de virtualização.

2. **Compor políticas de segurança é muito difícil**: a composição de políticas de segurança individuais de componentes em uma política resultante para um sistema é um problema NP ou até exponencial, mesmo utilizando-se modelos formais com restrições, a exemplo das Redes de Petri estendidas [2].

   O resultado é que mesmo que exista uma descrição fiável, formal do que se esperar sobre a segurança de um componente (uma raridade), não é trivial compor tal descrição em uma política de segurança para o sistema resultante.

   Talvez pela dificuldade, talvez pelo aspecto tipicamente qualitativo dos requisitos de segurança, observa-se que na média poucos profissionais entrando no mercado possuem formação no assunto.

   Também é sintomático que a maior parte das licenças de software expressem que "este software não vem com garantia nenhuma, nem a garantia de servir para um fim específico";

3. **Modelagens otimistas ou rasas**: frequentemente, os pesquisadores e os praticantes em computação não consideram em seus modelos (teóricos e/ou mentais) que algoritmos e a suas realizações na forma de programas não são objetos concretos, mas apenas instruções para um ou mais elementos processantes (CPUs, MCUs, GPUs, NPUs, FPGAs), organizados em um ou mais equipamentos, executarem.

   Essa redução de modelagem rotineiramente implica em assunções otimistas e irreais sobre a isolação entre os componentes de software e sobre dados sensíveis. Por exemplo, a arguição de que um sistema implementa "proteções em camada" quando os seus componentes executam todos em uma mesma máquina, sob um mesmo usuário, é geralmente falsa pois possui diversos modos comuns de falha (i.e., item que, se atacado, viola a segurança de mais de um componente, como o processador, o disco, e o kernel).

   Outro erro que frequentemente se mostra fatal é considerar máquinas virtuais (ou *containers*) como realmente isoladas, mesmo com inúmeros casos de "escape" das principais tecnologias nos últimos anos.

4. **Excessos nas abstrações e dependências de software**: a excessiva abstração dos recursos computacionais em APIs e frameworks e o vertiginoso aumento do número de dependências de software simultaneamente facilitam a inserção proposital de vulnerabilidades e também dificultam, ou mesmo impedem, que as equipes res-

ponsáveis pela implementação, manutenção e segurança de corretamente mantenham uma modelagem atualizada de ameaças e de arquitetura de sistema.

Pouco se tem noção do problema, mas dados do Apache Maven [3] indicam que a aplicação média em Java em 2022 dependia de aproximadamente 40 (!) bibliotecas de terceiros. Neste contexto, uma chance de apenas 1,7% de contaminação por ataque *supply chain* individual de componente leva a uma chance composta de comprometimento da aplicação de 50%.

Por outro lado, sabe-se bem da disciplina de Engenharia de Software que a corrosão de arquitetura [4] é um elemento que dificulta a identificação e correção de problemas de segurança, tornando o ciclo de mitigações lento.

5. **Muitos desenvolvedores sobrestimam a dificuldade de um ataque**: nestes últimos 25 anos pude observar um padrão – a maioria dos desenvolvedores, mesmo aqueles formados em nossas melhores escolas, nunca viu um ataque real, prático, sendo executado sobre um sistema conhecido ou desenvolvido por eles. Como resultado, muitos deles subestimam, quando não simplesmente ignoram, ameaças típicas.

   Por outro lado, ao trabalhar em diversos casos de clientes em minha trajetória profissional junto de uma equipe competente de pesquisadores em segurança, pude observar que em torno de 9 de cada 10 casos pudemos "vencer" e tomar o controle de sistemas, às vezes atacando um único módulo, mas muitas vezes abusando da arquitetura – este tipo de experiência muitas vezes não é oferecida nos currículos básicos de formação de profissionais de Computação.

   Esta dicotomia, da falta de repertório, tem o frequente efeito que pode ser descrito como "gente nova comete erros clássicos".

Melhores práticas para a segurança de sistemas compostos

As melhores práticas de segurança para sistemas compostos e seus componentes varia a depender de seus objetivos de segurança e de asseguramento, já que diversas metodologias e escolhas de engenharia possuem impacto em custos, trabalho adicional, escolhas tecnológicas e prazos de execução. Ainda não se pretende ser exaustivo nas práticas, mas apenas listar algumas que aparentemente recebem muito pouca atenção nos currículos de graduação.

Antes de avançar, faz-se necessário o estabelecimento de alguns termos usados nesta seção. Um **objetivo de segurança**, também chamado de **reivindicação** de segurança, do inglês "claim" é uma descrição daquilo que uma peça de software, subsistema ou o próprio sistema diz que entrega. Por exemplo: "Claim 1: a mensagem tem confidencialidade garantida

com nível de segurança de 2256 contra adversários não-quânticos". Já o **asseguramento** ("assurance") se refere ao nível de certeza que um dado "claim" é verdadeiro. Por exemplo, o Claim 1 é verdadeiro com "alto nível de probabilidade" ou "com prova formal".

A experiência prática mostra que conceber, implementar e manter componentes e sistemas com nível alto de segurança tende a ser relativamente menos trabalhoso do que obter alto nível de asseguramento, principalmente pelos efeitos do Teorema de Rice e da Composição de Políticas de Segurança. Em geral, o alto nível de esforço necessário para se obter alto nível de asseguramento é incompatível com muitos cenários de uso, chegando a ser 3.83 maior [5] do que para asseguramentos mais relaxados, a exemplo dos níveis EAL 1 versus EAL 7 no padrão ISO/IEC 15408 – "Common Criteria".

Em suma, é fundamental harmonizar a criticidade do caso de uso com os objetivos de segurança e os níveis de asseguramento sob a perspectiva das capacitações das equipes envolvidas, bem como dos recursos disponíveis. Em termos de práticas, observamos como mais efetivas:

1. **Baseline de vocabulário e metodologias:** todos os integrantes das equipes devem ser treinados para utilizar uma nomenclatura comum para se expressar em termos de objetivos de segurança, ameaças, vulnerabilidades, verificações de segurança, resposta a incidentes, etc., utilizando para isso algum framework documentado de gestão de ciclo de vida seguro de sistemas, a exemplo do Microsoft SDL ou do modelo SAMM;
2. **O arquiteto de solução deve ser o dono da segurança em projetos pequenos e médios:** em projetos de pequena até média escala, é importante que o arquiteto da solução, ou papel similar, entenda *todos os módulos e os componentes* de sistema, tanto em termos de código fonte, como em termos de arquitetura e também seja versado nos principais tipos de ataques. Esta é uma posição bastante exigente, mas que reduz em muito a necessidade de formalização do processo de desenvolvimento e manutenção de software. Em geral, a formação deste talento envolve trilhas de formação em ciclo de vida seguro (p.e. MS-SDL, SAMM), tecnologias específicas usadas na aplicação, e segurança ofensiva (p.e. CEH, CompTIA PenTest+, ECSA/LPT);
3. **Projetos críticos de qualquer tamanho requerem formalismo**: é fundamental o emprego de uma metodologia de asseguramento, a exemplo da NATO AEP-67 ENGINEERING FOR SYSTEM ASSURANCE NATO IN PROGRAMMES, que apresenta uma forma de documentar e demonstrar as reivindicações de segurança e os respectivos níveis de asseguramento durante o ciclo de vida da solução, levando-se em conta todos os seus componentes. Pelo seu poder de coordenação e nível de esforço ajustável, a NATO

AEP-67 tem sido empregada com sucesso em diversos projetos com sucesso, bem como serviu de base para treinamento de equipes [6], assunto que é tratado mais adiante;

4. **Separar na origem qual o software descartável daquele de produção**: é preciso evitar utilizar na produção a mentalidade de software descartável, típico das etapas de prototipação, onde o uso de versões "*nightly build*" de componentes é comumente feito pelas equipes de desenvolvedores entusiastas e que buscam sempre as "últimas features". O congelamento dos componentes em versões "Long Term Support - LTS" tem papel fundamental no provimento de sistemas seguros não só porque garantem a manutenibilidade de longo prazo do sistema composto, mas fundamentalmente porque "congelam" a superfície de ataques e reduzem a inserção de defeitos de segurança ao longo do tempo, permitindo que a equipe possua um melhor modelo formal e/ou mental daquilo que quer proteger.

Casos de asseguramento como ferramenta de formação de equipes

Casos de asseguramento na forma da NATO AEP-67 organizam reivindicações de segurança de forma hierárquica, conforme exemplo da figura 1. Cada "claim" pode ser suportado por uma ou mais reivindicações intermediárias ("sub-claims"), recursivamente.

No exemplo oferecido, o "claim A" requer simultaneamente que os sub-claim 1 e 2 (e possivelmente outros) sejam verdadeiros para que ele seja verdadeiro. Em um certo momento um sub-claim deve ser evidenciado ou assumido como verdadeiro, com um certo nível de certeza. No caso do sub-claim 1, ele é mostrado verdadeiro através de argumentação e de critério de avaliação pré-definido.

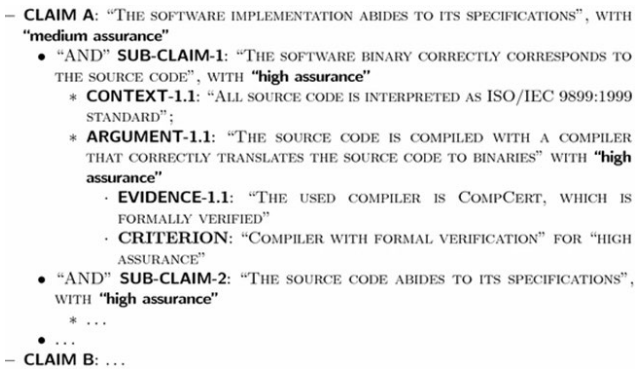

FIG. 01 | **EXCERTO DE UM CASO DE ASSEGURAMENTO, FONTE [6].**

Como o leitor atento pode imaginar, os casos de asseguramento podem se tornar bastante detalhados já que garantir que determinada asserção é verdadeira pode requer diversas condições intermediárias, afetando muitos componentes de sistemas. E mais, os casos de asseguramento são flexíveis o suficiente para incorporar as diferentes fases do ciclo de vida de um componente ou sistema.

Pois bem, justamente esta capacidade (ou necessidade) de expressão e detalhamento na composição dos casos de asseguramento é que tem se mostrado instrumental no treinamento e no aperfeiçoamento de equipes que lidam com o projeto, desenvolvimento e manuten-

ção de sistemas, conforme reportamos no em [6]. Naquele projeto, estudantes de pós-graduação e graduação da Unicamp foram organizados em times que tinham por objetivo implementar um serviço de mensageria seguro e assegurado por grupo, documentar as suas garantias de segurança e, posteriormente, atacar o sistema da outra equipe. Os times foram assim conduzidos por todo o ciclo de vida de suas soluções, provendo insights poderosos sob a visão holística necessária na área de segurança da informação.

No experimento educacional, os casos de asseguramento se mostram fundamentais pelo menos em três aspectos educacionais: (i) obrigaram cada um dos estudantes a desafiar as suas assunções sobre a segurança dos componentes do sistema, (ii) demonstraram para a equipe a necessidade de se definir e documentar políticas/reivindicações de seguranças simples e precisas para os componentes dos sistemas, e (iii) serviram como eixo de comunicação objetivo entre os membros das equipes, otimizando esforços e limitando lacunas.

Longe de ser apenas um ganho teórico e um exercício acadêmico, depois do experimento reportado em [6], pude ver em primeira mão como os casos de asseguramento foram fundamentais para aperfeiçoamento contínuo, "*on-the-job training*" de profissionais. Mais do que isso, tais casos têm servido aos *stakeholders* dos sistemas desenvolvidos já que passam a ter uma descrição precisa daquilo que podem esperar sobre os seus sistemas.

Conclusão

A concepção, implementação, obtenção e manutenção de sistemas compostos seguros é um desafio feroz e requer acima de tudo consciência situacional das equipes envolvidas. Isto é, em que pese os benefícios de negócio da modularização de software, esta mesma organização em componentização abstrai diversos dos caminhos práticos que os adversários utilizam para realizar ataques de sucesso.

Para minimizar o número de vulnerabilidades é necessário que as equipes de desenvolvimento, operação e segurança possuam a mesma modelagem de ameaças, vocabulário e práticas de desenvolvimento, bem como visão holística sobre os sistemas e seus componentes. Para este fim, metodologias e frameworks como o Microsoft SDL, SAMM e NATO AEP-67 têm se demonstrado efetivas.

Além disso, as equipes dedicadas à concepção e desenvolvimento devem ter contato reiterado com as equipes de segurança de forma a se manterem educadas nas técnicas de ataques e, sobretudo, no frequente baixo esforço necessário para um ataque de sucesso.

Na visão deste autor, nada impede que todos estes conhecimentos e práticas sejam incorporados nos currículos de graduação e pós-graduação em Computação.

Referências

1. Len Bass, Paul Clements e Rick Kazman, "Software Architecture in Practice", 4ª Edição, agosto de 2021.
2. Yen, Hsu-Chun. "On the Regularity of Petri Net Languages." Proceeding of 13th IEEE Annual International Phoenix Conference on Computers and Communications (1994): 329.
3. A. M. Mir, M. Keshani and S. Proksch, "On the Effect of Transitivity and Granularity on Vulnerability Propagation in the Maven Ecosystem," 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Taipa, Macao, 2023, pp. 201-211, doi: 10.1109/SANER56733.2023.00028.
4. M. Ullah Khan, M. Munib, U. Manzoor and S. Nefti, "Analyzing risks at architectural level," International Conference on Information Society (i-Society 2011), London, UK, 2011, pp. 231-236, doi: 10.1109/i-Society18435.2011.5978442.
5. Kou, K., Jeong, J., & Lee, G. (2008). Definition of Evaluation Assurance Levels and Estimation of Evaluation Efforts for Operational System Based ISO/IEC 19791. 2008 International Conference on Security Technology, 176-183. https://doi.org/10.1109/SECTECH.2008.41
6. Gallo, R., Dahab, R. (2015). Assurance Cases as a Didactic Tool for Information Security. In: Bishop, M., Miloslavskaya, N., Theocharidou, M. (eds) Information Security Education Across the Curriculum. WISE 2015. IFIP Advances in Information and Communication Technology, vol 453. Springer, Cham. https://doi.org/10.1007/978-3-319-18500-2_2

ROBERTO GALLO é veterano em defesa e segurança cibernética para aplicações de Estado, telecomunicações, militares, (contra-)inteligência e corporativas. Atuação nos âmbitos acadêmico, profissional e institucional. Possui Doutorado e Mestrado em Ciência da Computação com foco em segurança cibernética pela UNICAMP. Possui graduação em engenharia de computação pela mesma universidade.