A9

Usando compotentes com vulnerabilidades conhecidas,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ameaça**  **Agentes** | **Vetores de Ataque** | **Segurança**  **Fraquezas** | **Impactos Tecnicos** | **Impactos no negócio** |
| **Especificação da aplicação** | **Exploração**  **MÉDIA** | **Prevalência**  **COMUM // Média de Detecção** | **Impactos Severos** | **Aplicação /**  **Específico de Negócios** |
| **Alguns componentes vulneráveis (por exemplo, bibliotecas de estrutura)**  **Podem ser identificados e explorados com ferramentas automatizadas,**  **Ameaça agente pool além atacadores alvo para incluir atores caóticos.** | Os atacantes identificam um componente fraco através de digitalização ou análise manual.  Eles personalizam o exploit conforme necessário e executam o ataque. Fica mais  Difícil se o componente utilizado for profundo na aplicação. | Muitas aplicações e APIs têm esses problemas porque suas equipes de desenvolvimento  Não se concentram em garantir que seus componentes e bibliotecas estejam acima  Até a presente data.    Em alguns  Casos, os desenvolvedores nem sequer sabem todos os componentes que estão usando, nunca  Mente suas versões.    Dependências de componentes tornam as coisas ainda piores. As ferramentas estão se tornando comumente disponíveis para ajudar a detectar componentes com vulnerabilidades conhecidas. | Ele gama completa de fraquezas é possível, incluindo injeção, controle de acesso quebrado, XSS, etc.    O impacto pode variar de um mínimo a uma aquisição completa do  compromisso. | Considere o que cada vulnerabilidade pode significar para o  Negócio controlado pelo aplicativo afetado.    Poderia ser trivial ou poderia significar um compromisso total.. |

**Sou Vulnerável ao vulnerabilidades conhecidas?**

O desafio é monitorar continuamente os componentes (Tanto do lado do cliente quanto do lado do servidor) que você está usando para novos relatórios de vulnerabilidade. Este monitoramento pode ser muito difícil porque os relatórios de vulnerabilidade não são padronizados, e é difícil encontrar e procurar os detalhes que você precisa ( por exemplo, o componente exato em uma família de produtos que tem o vulnerabilidade). Pior de tudo, muitas vulnerabilidades nunca são Relatados a “clearinghouses” centrais como [CVE](http://cve.mitre.org/) e [NVD](http://nvd.nist.gov/home.cfm).

Determinar se você está vulnerável, é necessário pesquisar Bancos de dados, bem como manter-se a par das listas de discussão do projetos e anúncios para qualquer coisa que possa ser uma vulnerabilidade. Este processo pode ser feito manualmente ou com

ferramentas automatizadas.

Se uma vulnerabilidade em um componente é descoberto, avalie cuidadosamente se você está realmente vulnerável. Verifique se o código usa a parte vulnerável deste componente e se a falha pode resultar em uma Impacto que você se preocupa. Ambos os cheques podem ser que os relatórios de vulnerabilidade podem ser deliberadamente vagos.

**Como posso evitar isso?**

A maioria dos componentes de projetos não criam patches de vulnerabilidade para versões antigas. Portanto, a única maneira de corrigir o problema é atualizar para a próxima versão, que pode exigir outro código modificado. Os projetos de software devem ter um processo para:

1. Inventário contínuo das versões do lado do cliente e componentes do servidor e suas dependências usando ferramentas como versões, DependencyCheck, retire.js, etc.

2. Continuar monitorando fontes como NVD para vulnerabilidades em seus componentes. Usar software de ferramenta de composição para automatizar o processo.

3. Analisar as bibliotecas para ter certeza de quem elas são realmente antes de fazer alterações, já que a maioria das componentes nunca são carregados ou invocados.

4. Decida se deseja atualizar o componente (e reescrever o aplicativo para corresponder se necessário) ou deploy virtual que analisa o tráfego HTTP, fluxo de dados ou execução de código isso evita que vulnerabilidades sejam exploradas.

**Exemplo de Cenários de Ataque**

Os componentes quase sempre são executados com o privilégio

Aplicação, de modo que falhas em qualquer componente podem resultar em

impacto. Tais falhas podem ser acidentais (por exemplo, erro de codificação) ou

Intencional (por exemplo, backdoor no componente). Algum exemplo

Exploráveis descobertas são:

Apache CXF Authentication Bypass - Ao não fornecer

Um identificador de identidade, os invasores poderiam invocar qualquer serviço da Web

Com permissão total. (Apache CXF é uma estrutura de serviços,

Não deve ser confundido com o Servidor de Aplicativos Apache.)

\* Struts 2 Execução Remota de Código - Envio de um ataque no

O cabeçalho Content-Type faz com que o conteúdo desse cabeçalho

Ser avaliada como uma expressão OGNL, que execução de código arbitrário no servidor.

Aplicativos que usam uma versão vulnerável de qualquer componente

São suscetíveis a ataque, já que ambos os componentes são diretamente

Acessível aos utilizadores da aplicação. Outras bibliotecas vulneráveis,

Mais profundo em uma aplicação, pode ser mais difícil de explorar.

Referências da OWASP

[OWASP Dependency Check (for Java and .NET libraries)](https://www.owasp.org/index.php/OWASP_Dependency_Check)

[OWASP Virtual Patching Best Practices](https://www.owasp.org/index.php/Virtual_Patching_Best_Practices)

External

[The Unfortunate Reality of Insecure Libraries](http://www.aspectsecurity.com/research-presentations/the-unfortunate-reality-of-insecure-libraries)

[MITRE Common Vulnerabilities and Exposures (CVE) search](https://www.cvedetails.com/version-search.php)

[National Vulnerability Database (NVD)](https://nvd.nist.gov/)

[Retire.js for detecting known vulnerable JavaScript libraries](https://github.com/retirejs/retire.js/)

[Node Libraries Security Advisories](https://nodesecurity.io/advisories)

[Ruby Libraries Security Advisory Database and Tools](https://rubysec.com/)