INTRODUÇÃO

Não há software seguro.

Ataques de motivação aos usuários:

• Roubar credenciais (trojan)

• Ransoware: tornar o pc inutilizável e pedir resgate

• Mineração de bitcoin: usar o pc como recurso

• Zombie: acessar a rede como usuário (span, dns, clickbot)

Ataques de motivação em servidores:

• Violações de dados: Servidor salva informações do usuário (por exemplo, cartão de crédito)

• Motivações políticas e geoestratégicas: por exemplo, eleições

• Infectar usuários de PC:

o Atacantes da cadeia de suprimentos: servidor infectado que distribuiu SW

o Ataques ao servidor Web: servidor infectado, navegador comprometido

Bug bounty: recompensa para aqueles que relatam vulnerabilidade para a empresa

Dia zero: ataque que ninguém jamais executou.

Garantia de segurança:

• Defensivo: Tudo o que não é bom não pode acontecer

• Atores: Entidades que intervêm no sistema (pessoas, organização, empregador, máquina). intenção explícita unificação

o Script-kiddies: curioso incapaz

o Atacante ocasional

o Pessoas com intenção de fazer dano

o Grupos organizados ou Concorrentes (espoionagem industrial)

o Países/Estado/Governo

Processo:

Identifique o atacante e pense como ele:

• Procure um escravo mais lento

• Identifique orçamentos confiáveis

SEGURANÇA

Modelo binário:

• Típico em croptogram

• X: Recursos do invasor Y: Objetivos de segurança

• Nenhum invasor X consegue quebrar Y

• Provas matemáticas (para sistemas muito pequenos)

• Fora de escala, os modelos podem estar errados

Gerenciamento de riscos:

• Minimize o risco com base nas ameaças mais prováveis: otimização da segurança versus custo

• Sistemas mais complexos

• A análise de risco pode estar errada, ameaça mal classificada => caos

Ciclo: Novo ataque vs nova defesa

Ativos: Recurso que tem valor para um ator que não é do sistema (por exemplo, informações, reputação, dinheiro, infraestrutura)

Perda de valor devido à falência do CID:

• Confiabilidade: Sigilo, privacidade

• Integridade: Sem alteração, autenticidade de origem

• Disponibilidade: existência, constância

Matriz de análise de risco: probabilidade X impacto

Custo de Ataque x Custo de Defesa

Triângulo: segurança, funcionalidade, usabilidade

Vulnerabilidade: falha acessível a um adversário que pode ser explorada causada por software de baixa qualidade, requisitos errados, configuração errada, uso errado, etc

Tipos de Ataque:

• Passivo: Espião

• Ativo: o invasor interage com o sistema

• Negação do serviço

Ataque bem-sucedido => sistema comprometido

Motivação/Ameaca: objetivo causar perda de valor, causar possível incidente

Vulnerabilidade: deficiência técnica

Exploit: maneira de explorar a vulnerabilidade

Ameaça: Classificada de acordo com o tipo de ataque e atacante. Varia de acordo com cada domínio

Modelo de ameaça:

• Identificar ativos, adversários, tipos de ataque para prevenir e descartar

• Perímetro de segurança: Contato com o mesmo nível de segurança

• Superfície de ataque: Pontos de contato com o exterior do perímetro de segurança.

Mecanismos de segurança: Método, ferramenta ou processo para implementar uma política de segurança. (Eu confio nesse mecanismo). EG indet/autenticação, controle de acesso mec, criptografia.

Política de Segurança: Conjunto de processos a serem seguidos para garantir a segurança em um modelo de ameaça. Pode ser prevenção/detecção/recuperação.ex: senhas, e-mails ou acessos.

Confiabilidade:

1. Defina o problema

• Analisar requisitos de segurança

• Definir modelo de ameaça: ativos, classes de ameaças e adversários

1. Definir modelo de confiança:

• Componentes/atores em que confio

• Fazer o que

1. Defina a Solução:

• Desenhar políticas de segurança

• Identificar mecanismos de segurança necessários e suficientes

1. Validar e justificar a solução:

• Validar a adequação de dois modelos para a aplicação específica da análise

• Demonstrar que: o Mecanismos de segurança são suficientes ou Idem pressupostos de confiança ou Implementar políticas de segurança

Auditoria Técnica: Validar a eficiência do nível de segurança (caixa branca/preta).

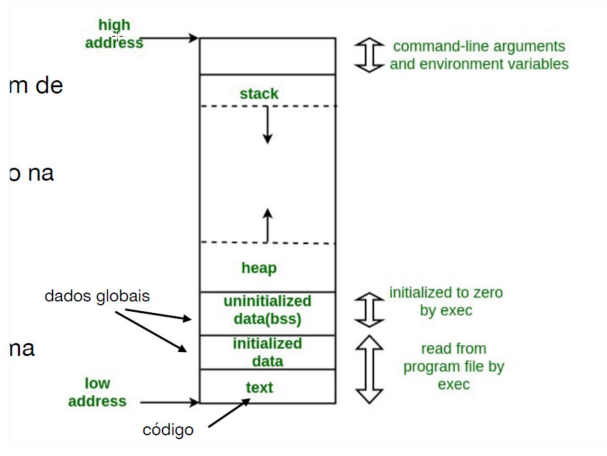
CONTROLE 1

Alvo do ataque: assuma o controle

usurpação do controle:

• Input chama o programa a ser executado

• Instruções do atacante alterando a sequência de instruções



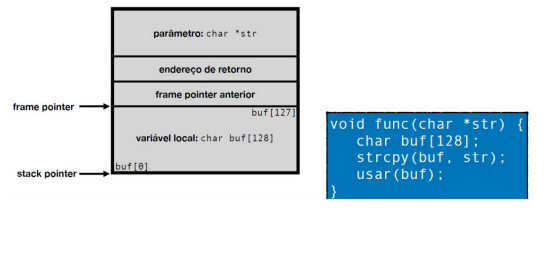
Quebrando a pilha:

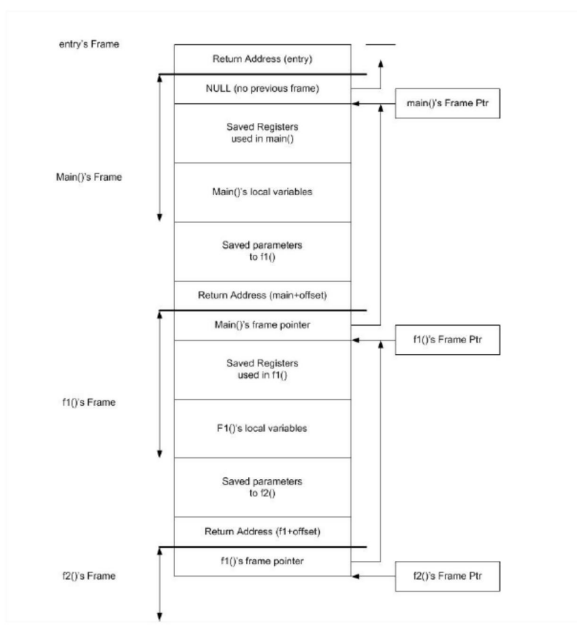
Quadro com:

• FP(RBP) base (var locales)

• SP(RSP) top (end mais baixo)

• Contem: parâmetros, dados salvos para retorno, var locais, outros var guar tem





Código malicioso. Sheel code (binário), comando seq que executa comandos no shell

Precedido por NOP e inserido na memória. Sem \0 e outros truques

Causa da vulnerabilidade: Uso indevido de libs, strncpy ou root (mv já tem vulnerabilidades).

EG: Aviso ignorar verificação nao crítica, estouro de 1 byte.

CONTROLE 2

Estouro de buffer na pilha alterando-os:

• Apontadores para funções: 1 função com várias implementações, vinculação dinâmica

• Manipuladores de exceção: lista vinculada de ponteiros para funções

• Buffers LongJmp que lidam com exceções em c

Ideia: Escreva acima 1 dessas listas de funções para que quando essa função for executada, o código do adversário seja realmente executado.

EG: blob que bem de fora, desencontro entre a ideia maluca e o tamanho do blob.

Robustez: antecipar flapping, localização de shellcode e evitar crash => heap spraying (preencha a memória com uma cópia do shellcode)

Use After Free: Código usa/lê uma instância já destruída => desconheça se comporte mais sim ou adversário o código pus seu teve ataque.

Outros tipos de transbordamento:

• Inteiros: truncar inteiro pode dar valores negativos. Se este inteiro for usado para negar entradas com tamanho maior que o buffer pode levar a um ataque. O mesmo acontece com contagens já que axb pode exceder. EX: malloc (aXb) pode dar malloc(-1), então ler e copiar o buffer para que certesa var substituirá.

• String de formatação: String controlada pelo adversário. Printf roda de baixo para acima. Se eu enviar mais/menos parâmetros, dou origem ao ataque:

o Printf(“%d\n”): imprime algo diretamente da pilha

o Printf(“%s\n”): imprime algo apontado pelo endereço na pilha

o Printf(“%d%d%D…%s”) permite ler qualquer endereço na memória => a própria string de formato está na pilha

• Spstringf: escreve

%n: escreve na memória usando printf. Parâmetro é um endereço na memória, programa vai para esse endereço e coloca o número de caracteres que são impressos para não serem ecrados ou momentâneos. Eu escrevo endereços byte por byte.

CONTROLE 3

Sistemas modernos:

• Prevenção de execução de dados (DPP): uma página escrita não pode ser executada e vice-versa W(V)X

• Randomização de layout de espaços de endereço (ASLR): a localização da memória varia em cada execução ou como é difícil prever direções

Portanto, não injete código, mas use o que está presente no sistema: Bibliotecas

libc (direito conhecido): sistema para executar shell e mprotect para alterar as permissões de memória

Ideia de ataque: Altere o endereço de retorno da biblioteca configurando a pilha de acordo. Observe que quem chama uma função é quem limpa os parâmetros que eu coloco e observe que preciso de um endereço de retorno para quando essa função terminar (sair)

Programação orientada de retorno: Chame N funções de biblioteca em uma linha para obter o comportamento desejado.

Sem parâmetros ou com parâmetros apenas na função N => funciona

Com parâmetros em cada função: digite f1 e f2 chame um f1prima que limpa a pilha (não recebe parâmetros) sendo ele um pedaço arbitrário de código de outra função.

Hoje em dia existe uma posição independente de execução (PIO) que torna o ROP mais difícil

Ler fora da zona correta também é um problema: HeartBeat(timeaout send int, server lê esse número de bytes e envia para o cliente)

CONTROLE DE DEFESA

Ataques: Stach smashing, heap spraying, use after free, format string, integer overflows, rop

Defesa em profundidade:

• A) Minimizar a probabilidade de erro na memória: linguagens seguras de memória e verificação de código

• B) Medidas de controle da plataforma: Evite executar código malicioso

• C) Medidas de proteção não executáveis: Detectar tentativa de sequestro

b)

• Proteção de Espaço Executável: W(V)R em HW ou SW. Não previne ROP, sim injeção de código, problemas em sistemas just in time compilação.

• Randomização do Layout do Espaço de Endereço: varia o endereço em cada execução. cara difícil antecipar direitos, mas pode ser evitado com:

o Leia endereços com erros de string de formato, tente vários endereços, use código do kernel (o logotipo surgiu KASLR e KARL)

c)

• Segundo nível de proteção: na compilação (transforma ataques em DoS)

• Monitoramento de tempo de execução: Stack canaries para evitar injeção de código usando stack guard gcc.Eu coloco informações na pilha entre FR e local vars, então quando eu saio da função eu verifico se o canary ainda está lá. Pode ser aleatório ou final (com caracteres como \0). Pode ser derrotado com:

o Código com variável e ponteiro ou função de ordem superior (recebo ponteiro de função, modifico e vou para onde quero para que

o Outra vulnerabilidade para aprender canário

o Força brutal (processo que lança fios)

D) Medidas adicionais

• Tampões próximos aos canários

• Copiar argumentos de função para dois buffers locais

• Pilha de sombra (pilha de controle redundante)

• Para dados perigosos em pilha separada

Marcação de memória:Tanto na compilação quanto na execução, os quadros e marcadores possuem tags diferentes e devem ser iguais ou ter erro. Regiões de memória com finalidades diferentes possuem tags diferentes

Integridade do fluxo de controle: protege a modificação de ponteiros para funções geradas dinamicamente (controle seq e chamadas de função). Na compilação determina um conjunto válido de pontos de origem para cada destino e na execução é verificado.

Ajuda a evitar a entrada de uma função como resultado de um retorno. As mais avançadas usam gráficos e criptografia. Ele é implementado com um sinalizador ativo ao chamar uma função ou validar o direito de chamada.

Segurança do programa:

• Programa permanece dentro da semântica da linguagem (faz so ou que mus): Linguagens fortemente tipadas, soma de verificação de tempo de execução, ferrugem usa verificação estática e dinâmica

• Ferramentas para verificar programas

SEGURANÇA DO SISTEMA 1

Princípios fundamentais:

• Economia de mecanismos: Sistema suave com as funcionalidades necessárias e se mecanismos de segurança equivalentes escolhem o mais simples

• Proteção padrão: A configuração padrão é conservadora. Por exemplo, não deixar senha padrão

• Design aberto: o sistema é público, os segredos são os parâmetros

• Defesa em profundidade: não confie em nosso mecanismo de segurança, existem vários

• Privilégio mínimo: Cada ator deve ter apenas as permissões/privilégios necessários para suas tarefas

• Separação de privilégios: todas as funcionalidades devem ser compartimentadas (isolado) com domínio de confiança diferente.

• Mediação completa: Cada recurso deve definir suas possíveis operações, permissões de usuários a ela e isso deve ser verificado em cada acesso a ela.

Mediação Completa: Controle de Acesso

• Ator: Entidade que realiza a ação

• Recurso: atividade na qual a operação é realizada

• Operação: ação a realiza

• Recurso + Operação: permissão

Tipos:

• Matriz de acesso: Ator x recurso (operação). Claro e eficiente, mais verdadeiro

• Lista de Controle de Acesso (ACL): Linha ou coluna da matriz

• Controle de acesso baseado em função (RBAC): defino uma função com suas permissões e vinculo atores a funções

• Controle de acesso baseado em atribuição: armazene recursos e atributos. Matriz dez atributos preciso para cada recurso

No SO: Os atores e aplicativos são ameaças aos recursos de HW, portanto, devem ser isolados virtualmente por meio de mecanismos de acesso mediado (gerenciar SO é configurar esses mecanismos)

SEGURANÇA DO SISTEMA 2

núcleo:

Modo kernel (acesso a tudo) vs modo usuário (sem acesso direto aos recursos HW)

Para acessar o HW ou o modo usuário use como chamadas de sistema (código executado em modo kernel), que são pontos críticos e, portanto, requerem monitor de referência

Para que o modo usuário funcione com mais privilégios é preciso preparar os argumentos, identificar o ponto de acesso permitido ao kernel, executar essa instrução para passar ao modo kernel (esses pontos são poucos).

Processos:

A fronteira entre os processos é uma fronteira de confiança (eles são isolados uns dos outros)

O sistema operacional usa uma hierarquia de processos e geralmente herda privilégios de dois criadores. Cada usuário tem UID (id do usuário) e GID (id do usuário do grupo), processos idem (0 é root)

Comunicação entre processos IPC: chamadas de sistema (arquivos, mem compartilhados, sockets, sinal)

Modelo de confiança:

• Bios e kernel, processo de inicialização, processo de hibernação

• Processos criados pelo kernel e administrador não podem alterar o status de confiança

• Sistema confiável é mesmo se faz o que faz ou o que foi criado

Modelo de ameaça:

Bios corrompido, arquivos de hibernação corrompidos, bootloader corrompido, ataque de inicialização a frio, vulnerabilidades não-kernel

Medidas de mitigação:

• Monitorização: logs, aplicativos de monitoramento

• Instalação/execução de código baseado em assuntos digitais

Memória:

• Um processo não pode acessar o espaço de memória de outro

• Confidencialidade, integridade e controle de fluxo do kernel devem ser protegidos no modo de usuário: acesso à memória mediado pelo kernel, acesso à página não seguro do disco desde a criptografia do disco (isolamento versus eficiência)

• TLB: cache de tradução de direitos de leitura/gravação/execução

• Como a chamada do sistema é executada metade no modo de usuário (UR/UW/UX) e metade no kernel mode(PR/PW/PX), para ser mais eficiente o mapeamento do kernel é usado onde parte da memória virtual do kernel é mapeada para a memória virtual do processo com diferentes permissões

Defesa em profundidade:

Nem mesmo o kernel deve ter todas as permissões de escrita em outros processos (até respeita o W(V)X) pois pode ser corrompido

SEGURANÇA DO SISTEMA 3

Sistema de arquivo

• Colar R/W/X: listar, criar conteúdo, inserir um colar. • -RWXRWXRWX: proprietário-grupoproprietário-todos os usuários • Superusuário é uma função irrestrita que pode alterar permissões (proprietário tmb) • O proprietário só pode ser alterado por SU • O grupo pode ser alterado por SU e proprietário

Processo:

• Ter um ID de usuário efetivo que é aquele que iniciou o processo e geralmente é determinado as permissões

• Alguns casos como login fazem drop de permissões

• Bit setuid faz que o ficheiro seja executado com o EUID do owner

• Users: o EUID(Effective user id): determina permissões o RUID(Real user id): quem lanca o processo o SUID(saved user id): útil para transições

• Setuid(x): root troca tudo ax, outros so mudam EUID(para RUID ou SUID)

• Seteuid(x): altera so EUID(não baixa definitivamente privilegios, nao usar para isso)

Confinamento:

Preciso executar código de fontes externas em uma plataforma confiável (código legado, java script no navegador)

Tipos:

• Airbag: confinamento em HW, PC completamente isolado. Difícil gerenciamento

• Máquina virtual: hw que separa o SO, hypervisor garante apenas entre eles

• Dentro do SO:

o Software Fault Isolation (SFI): Isolamento de memória o System Call Interposition (SCI): Acesso com chamadas sys monitoradas. kernel vs modo usuário

• Sandboxing: confinamento dentro de uma aplicação (javascript não broser)

• Todos: monitor de referência onipresente e sempre invocado para do mediation

•Chroot(change root paste)=>jailkit=>fopen(“..\..”) quebra =>freebsd jail (quase virtualization): sempre evitar que o usuário se torne root é a coisa mais importante

SEGURANÇA DO SISTEMA 4

Confinamento:

Namespaces e cgroups

Interposição de cal do sistema (SCI):

Monitores chamadas para chamadas sys e bloqueia as inapropriadas no kernel com seccomp, no modo de usuário com program sephere

• Ptrace: ao iniciar um processo, ele escuta e monitora quando o filhp enfrenta chamadas para syscals. É ineficiente porque intercepta todas as chamadas, aborta syscall e depois aborta o processo, Toctou por condições de corrida

• Seccomp (modo de computação de segurança): Chamo prctls e entro no modo de segurança (assim posso encerrar/retornar ou usar arquivos abertos). A violação do kernel termina ou processa

• Seccomp + bpf: linguagem bpf que permite ser mais fino com as syscalls para controle e outros. O resultado é :kill/rejeitar e retornar erro/permitir. O Dockerl usa isso porque os contêineres do sistema operacional são encadeados

Isolamento de falhas de software (SFI): Limite a área de memória acessível a um aplicativo com um mínimo de sobrecarga e controle de execução de código (perigoso ou carregamento de verificação).

Código perigoso:

• Carregar/armazenar memória: verifique o endereço dentro do segmento

• Saldos: validar endereço de destino válido

Máquinas virtuais:

O SO ou o Hypervisor podem estar corrompidos, mas ambos são mais improváveis.

• Nsa nettop: Tão classificado e tão não classificado + hypervisor + SO + HW

• Nuvem do provedor: : Tão classificado e tão não classificado + hypervisor + HW

• Quber So: Tão orientado à virtualização com máquinas virtuais

Quebra de isolamento: canais subliminares (observe os efeitos colaterais SW/HW)

Os programas podem saber se estão no SO virtual: malware evita análise, direitos autorais. Como? análise de instruções disponíveis, acesso à memória latente, etc.

Malware: Nosso pc enfrenta coisas que o atacante quer

Tipos:

• Vírus: código que se espalha quando é executado

• Worm: código que se espalha automaticamente

• Rootkit: código que oculta e permite acesso com privilégios elevados

• Trojan: código aparentemente legítimo mais com ou objetivo de transmitir informações ao atacante

Malware é executado com privilégios de usuário ativo em nenhum momento, dá controle e tenta escalar privilégios graças a outras vulnerabilidades com objetivos de brinardeiras/dano ou vigilância/espionagem

Botnets: DDoS, spam, fraude de cliques, malware em grande escala, extorsão (rasonware, credenciais)

MALWARE

Casos:

• Ataque um serviço vulnerável exposto na rede: sempre atualize caminhos críticos • Servidor web malicioso injeta malware não cliente: malvertising • Software enganoso que convence o usuário a instalar run-o: truque antivírus, usb autorun

• Desfigurar equipamento não fabricado ou comprar desfigurado, ou desfigurar não transitou • Atacar fornecedor de software, injetar atualizações com backdoors • O invasor tem acesso direto à máquina comprometida: controle dos pais

Botnets: Rede de computadores com sistema de comando (c2) onde cada computador envia comandos para a rede. Arquitetura centralizada ou ponto a ponto, fluxo push ou pull

Defesa: detecta maq comprometida, tráfego de rede suspeito, honeypots.

Detecção de malware

Sistema de detecção de intrusão e sistema de prevenção de intrusão na rede ou host: NIDS/HIDS

I: evento de intruso, D: evento de alerta/detecção Falso+:P(D/não I) Falso-:P(Não D/i)

Aquisição, por exemplo: 0,1% FP + 2%Fn

Reconhecer ataques:

• Baseado em assinatura: lista negra

• Monitorar anomalias no uso de recursos

• Integração/especificação: registro de checksum de arquivos críticos, processo de whitelist • Tudo em nível de rede, urls, processos

Vírus: Assunto conhecido => planilha em dia criptografado "motor de mutação" => xecucao controlado

Antivírus: bloquear url/web + varredura de rede + varredura de carga útil + verificar atualizações online

Hids: execute o programa em sandbox + varredura de disco/memória + monitoramento de métrica

Nids: monitora toda a rede + analisa protocolos + analisa assuntos e comportamentos conhecidos + executa shadow + log + u+daes

SEGURANÇA NA WEB: MODELO

HTTP: esquema + domínio + caminho + informações extras (opcional)

Ops: obter (sem efeito colateral) para obter, postar (substituir todos os outros) para criar, colocar para substituir, corrigir para atualizar e excluir para excluir.

Estado mantido através de cookies

• Servidor: Db, gerenciamento de sessão, etc.

• Cliente: Navegador html, css, js, eventos.

Browser como um pequeno sistema de virtualização (já q executa código que bem de servidores).

Frame: Isolamento entre eles (funciona mesmo se falhar).

js dom pode mudar para page.

Modelos de ataque:

• Browser-Adv-Server: Segurança de rede

• Servidor de controles Adv: Ataque no PC do cliente

• Adv controla o cliente: Ataque ao servidor

• Invasor interno/web:

o 2 abas, 1 com adv, outra boa, servidor bom. O navegador garante nenhuma interferência entre páginas (malvertising)

o Adv controla uma página principal e tem um frame interno bom, servidor bom, navegador idem (solicite recursos do google para ver se você está logado)

Modelo de seguro: Processo como Página; arquivo como recursos/cookies; socket/tcp como fetch/http; subprocessos como frame

Isolamento: Política de Mesma Origem (SOP).

Confidencialidade: Nao conseguiu acessar por código dados de uma origem diferente

Integridade: O mesmo que confidencialidade mais para alterar

• A) DOM: c/ frame tem 1 origem (esquema + domínio + portal) e dados de acesso com = origem

• B) Mensagens: Os quadros podem se comunicar entre si (o programador deve filtrar)

• C) Comunicar-se com o servidor

• D) Cookies: envia cookies para servidor com = origem

C)Frame face requer http para != origins, ele também expõe seus recursos para != origins ou o que está lá, mas aqueles dados na resposta não podem ser analisados/modificados programaticamente, eles são processados pelo navegador, embora a incorporação exiba algumas informações ( por exemplo, login de imagem)

• HTML: Crie um quadro, mas não inspecione/modifique o conteúdo do quadro

• JS: Obtenha e execute JS terceirizado sem meu contexto, mas não inspecione/ manipule JS terceirizado.

Compartilhamento de recursos de origem cruzada (cors): permite que os serviços executem políticas SOP

• Solicitação Simples: Ao solicitar um recurso, o navegador solicita diretamente

• Solicitação pré-voada: navegador enfrentando solicitação fictícia, servidor autoriza ou não, é autorizado ou pde ou recurso do navegador

D) Cookies: Origem dada por domínio + caminho + esquema (opcional)

As páginas podem definir cookies para seu domínio ou hierarquia superior (exceto suf público)

Hoje o cabeçalho “SameSite” é usado para que NONE/STRICT envie cookies então se solicitado tem = origin que top level, LAX +é mais relaxado.

Importante usar sinalizar cookie seguro para que seja enviado apenas por http. Sinalizar apenas HTTP para que o cookie não seja acessível por javascript

