INTRODUÇÃO AO DESENVOLVIMENTO DE EXPLOITS

PROF. JOAS ANTONIO

SOBRE O PROFESSOR

- Pesquisador de segurança da informação pela Experience Security
- Assistente de Professor pela Cybrary
- Owner e Founder da Cyber Security UP
- Bounty Hunter pela HackerOne
- Palestrante
- Professor de Redes, Informática e Segurança da Informação pela Udemy
- Desenvolvedor Web
- Acumulei mais de 180 cursos e 25 certificações

SOBRE O LIVRO

- Esse ebook é destinado para
  - Aqueles que estão iniciando na área de desenvolvimento de exploits
  - Para profissionais de segurança obter um conhecimento prévio sobre exploits e seus fundamentos
  - Para aqueles que desejam saber como funciona um exploit e como é o seu desenvolvimento
  - Além de dar uma base para sobre o assunto para que você aprofunde suas pesquisas.

OBS: Alguns conteúdos foram difíceis de traduzir, então se alguma tradução estiver errado eu já adianto minhas humildes desculpas

CONCEITO DE EXPLOIT

- Existem diversos conceitos em relação a palavra exploit, mas todos chegam a seguinte conclusão
- Exploits é um código que explora uma brecha de segurança
- No caso os exploits eles tem como objetivo, efetuar explorações de vulnerabilidades com base em conjunto de instruções passadas
- Resumidamente, para que um exploit consiga ter êxito na sua exploração é necessário muita das vezes, a exploração manual da falha, pois a partir dela, surge o exploit
- Obviamente que a utilização de exploits é essencial para efetuar uma exploração em massa ou até mesmo ir mais além, diferente de uma exploração manual que pode se tornar limitada na maioria das vezes.

CONCEITO DE 0DAY

- Uma vulnerabilidade de dia zero é uma vulnerabilidade de software de computador que é desconhecida ou não tratada por aqueles que devem estar interessados em atenuar a vulnerabilidade (Wikipédia)
- Um 0day não precisa ser uma vulnerabilidade nova, que surgiu de hoje, mas sim uma vulnerabilidade que a empresa alvo não tenha conhecimento dela
- As vezes um 0day leva mais de 20 anos para ser descoberto, pois ele sempre esteve lá, mas ninguém da empresa quis tentar uma exploração, assim alguém externo que consegue efetuar tal exploração, ou divulga a vulnerabilidade para a empresa, ou se aproveita dessa vulnerabilidade
- Geralmente alguns vendem no mercado negro em troca de bitcoin.

CONCEITO DE PoC

- Uma prova de conceito, ou PoC é um termo utilizado para denominar um modelo prático que possa provar o conceito estabelecido por uma pesquisa ou artigo técnico
- Aplicando na área de segurança, um PoC é a prova de conceito de uma exploração sucedida em algum serviço ou aplicação, pode ser tanto um exploit que o pesquisador codou, tanto um vídeo ou um relatório
- Geralmente as PoC são mais vistas em Bug Bounty (Programa de recompensa para Pesquisadores/Hackers), para provar o conceito de uma vulnerabilidade encontrada.

CONCEITO DE SHELLCODE

- O shellcode é um código executado na exploração de vulnerabilidades, como por exemplo buffer overflow. Tradicionalmente, o shellcode dá ao atacante acesso à uma shell no computador explorado por meio de conexão reversa. Porém, ultimamente existem shellcodes que fazem ações mais complexas e variadas (WIKIPÉDIA)

BUFFER OVERFLOW

O que é Buffer?

- Em computação, um buffer é uma região temporária da memória onde são guardados dados para serem transportados de um lugar para o outro. Uma aplicação frequente de um buffer ´e quando dados são capturados de um dispositivo de entrada (um teclado ou mouse, por exemplo) e enviados `a um dispositivo de saída(monitor, impressora).
- Buffers também são utilizados em aplicações onde a taxa de dados recebidos ´e maior do que a taxa em que os dados são processados, como por exemplo em uma fila de atendimento: Enquanto um banco, por exemplo, tem 5 terminais de atendimento, pessoas são atendidas em grupos de 5. Caso o número de pessoas a serem atendidas seja maior que cinco, estas que chegaram por ´ultimo formam uma fila enquanto esperam o termino de um atendimento para serem atendidas. A fila funciona como um buffer, armazenando as pessoas em um lugar próximo de seu destino, enquanto ainda não podem se locomover ate ele.

BUFFER OVERFLOW

O que é Overflow?

- O conceito básico de um overflow ´e um transbordamento, ou seja, um overflow ocorre quando o volume de determinado objeto ou substancia ´e maior do que a capacidade de seu compartimento. Um rio raso exposto `a chuvas fortes aumenta consideravelmente seu volume de água, podendo levar o n´nível de ´agua a ser maior que a altura de suas margens, fazendo-o transbordar. Este fato caracteriza um overflow (transbordamento), o que leva a água do rio se espalhar por locais onde a mesma não deveria estar naturalmente.

- FONTE: https://securityinformationnews.files.wordpress.com/2014/02/bufferoverflow.pdf

EGGHUNTERS

- Um **egghunter** é um pequeno pedaço de código que é capaz de pesquisar com segurança o Espaço de Endereço Virtual Virtual Address Space por um "ovo" - uma string curta que significa o início de uma carga útil maior
- Um egg hunter é um shellcode que irá procurar por um padrão específico na memória (o ovo) e executará o shellcode que o segue

ESPAÇO DE ENDEREÇO VIRTUAL

- Na computação, um espaço de endereço virtual ou espaço de endereço é o conjunto de intervalos de endereços virtuais que um sistema operacional disponibiliza para um processo
- **Espaço de endereço virtual** é a coleção / **intervalo** de**endereços** lógicos ou **virtuais** para um processo específico em um sistema de computador. Só para dar uma ideia dos**endereços virtuais** - Quando a CPU executa as instruções do nosso programa, ela precisa buscar a instrução, operandos / dados da memória principal (DRAM).

ALGORITMO: CONCEITO

- Um algoritmo nada mais é do que uma receita que mostra passo a passo os procedimentos necessários para a resolução de uma tarefa. Ele não responde a pergunta “o que fazer?”, mas sim “como fazer”. Em termos mais técnicos, um algoritmo é uma sequência lógica, finita e definida de instruções que devem ser seguidas para resolver um problema ou executar uma tarefa
- Embora você não perceba, utiliza algoritmos de forma intuitiva e automática diariamente quando executa tarefas comuns. Como estas atividades são simples e dispensam ficar pensando nas instruções necessárias para fazê-las, o algoritmo presente nelas acaba passando despercebido.

ALGORITMO: EXEMPLO

- Por exemplo, quando precisa trocar uma lâmpada, você utiliza algoritmos:

```
Início
  Verifica se o interruptor está desligado;
  Procura uma lâmpada nova;
  Pega uma escada;
  Leva a escada até o local;
  Posiciona a escada;
  Sobe os degraus;
  Para na altura apropriada;
  Retira a lâmpada queimada;
  Coloca a lâmpada nova;
  Desce da escada;
  Aciona o interruptor;
    Se a lâmpada não acender, então:
        Retira a lâmpada queimada;
        Coloca outra lâmpada nova
    Senão
        Tarefa terminada;
  Joga a lâmpada queimada no lixo;
 Guarda a escada;
Fim
```

- FONTE: https://www.tecmundo.com.br/programacao/2082-o-que-e-algoritmo-.htm

O que é um bit

- A palavra *bit* vem do inglês e é uma abreviação de *binary digit*. Neste sentido, os computadores utilizam impulsos elétricos, que formam um bit, traduzido pelo código binário como estado de 0 ou 1.
- Uma combinação de bits formam um código de números, chamado pelos engenheiros informáticos de "palavra". Se um bit pode ser 0 ou 1, dois bits podem ser 00, 01, 10 ou 11 e assim por diante.
- Imagine agora um processador capaz de ler "palavras" com possibilidades de combinações muito superiores.

32 e 64 bits

- Estes valores indicam a capacidade de transmissão de informação entre a CPU e a memória RAM do dispositivo.

	32 bits	64 bits
O que é	Processador capaz de trabalhar até 32 bits por vez.	Processador capaz de trabalhar conjuntos de até 64 bits por vez.
Interação com a RAM	Utiliza até 4GB da memória RAM.	Pode utilizar mais de 4GB de memória RAM.
Quantidade de informação processada	32 bits são 4.294.967.295 combinações do código binário.	64 bits processa o dobro de informação.

32 e 64 bits

32 bits

- Um processador de 32 bits, por exemplo, teria a capacidade de processar de 0 até 4.294.967.295 números, ou de −2.147.483.648 até 2.147.483.647 na codificação de dois complementos.
- O processador armazena os dados do que precisa acessar em formatos de “endereços” em números, que serão distribuídos entre os limites de valores acima. Para isso, o processador de 32 bits pode utilizar até 4GB da memória RAM.
- Se a memória RAM do computador exceder 4GB, é necessário ter um processador de 64 bits para poder usufruir de mais memória.

32 e 64 bits

64 bits

- Os processadores mais modernos são capazes de trabalhar até 64 bits por vez. Isto quer dizer que a “palavra” lida pelo processador pode ser duas vezes o tamanho daquela em um processador de 32 bits.
- O potencial de um processador de 64 bits melhora significativamente o desempenho do computador, e está presente na maior parte dos dispositivos hoje em dia.
- Atualmente a maioria dos sistemas operacionais, no entanto, funcionam em processos de 32 bits. Para contornar isto, os computadores modernos, de 64 bits, vêm com uma extensão chamada “x86-64”, que simula o processamento em 32 bits.

CONCEITOS AVANÇADOS

STACK e HEAP: CONCEITOS E DIFERENÇAS

- Um *stack* ou pilha, neste contexto, é uma forma otimizada para organizar dados na memória alocados em sequência e abandonados (sim, normalmente não há desalocação) em sequência invertida a da entrada
- Um *heap* ou é a organização de memória mais flexível que permite o uso de qualquer área lógica disponível
- O sistema operacional ao carregar um programa na memória disponibiliza ao programa um espaço de endereçamento. Esse espaço é a memória disponível para aquele programa. O *Heap,* ou área de alocação dinâmica, é um espaço reservado para variáveis e dados criados durante a execução do programa (*runtime*). Vamos dizer que o *Heap* é a memória global do programa
- Já a pilha de funções (*stack*) é uma área da memória que aloca dados/variáveis ou ponteiros quando uma função é chamada e desalocada quando a função termina. Podemos dizer então que representa a memória local àquela função.

HEAP

- O *heap* é uma área de alocação dinâmica de variáveis. Se um programa utiliza uma lista encadeada por exemplo, ele aloca essa estrutura que cresce dinamicamente no *Heap.* Na linguagem C/C++, para alocar memória no *Heap*, utilizamos as funções *malloc(), calloc(), realloc()* e *new (*para C++*).* Para desalocar variáveis no *Heap* usamos as funções *free()* e *delete (*para C++*)*

- Ao desalocar memória do *Heap* a área volta a estar disponível para novas alocações. À medida que muitas alocações/desalocações ocorrem no *Heap* ele sofre muita fragmentação. Isso gera impacto na performance e na eficiência de como o programa aloca memória. Memória alocadas no *Heap* permitem leitura e escrita.

STACK

- A pilha de funções também é uma área disponibilizada dentro do espaço de endereçamento do processo. Essa área funciona como uma estrutura de dados LIFO (*last in first out*). Quando uma função é chamada durante a execução de um programa, um bloco de memória é empilhado no topo da pilha de funções. Nesse bloco existem referências para todas as variáveis criadas ou apontadas dentro da função chamada. Ao término da execução da função, esse bloco é desempilhado/desalocado.
- O *runtime* de cada linguagem junto ao sistema operacional irá controlar o tamanho máximo dessa pilha. Imagine um programa recursivo que faz uma chamada para uma função ***A*** dentro da própria função ***A***. Isso irá fazer inúmeros empilhamentos nessa pilha de funções. Normalmente os limites dessas pilhas são muito grandes e nem sempre é necessário se preocupar com elas.
- **FONTE:** https://blog.pantuza.com/artigos/heap-vs-stack

HEAP-SPRAYING

- No mundo da segurança, pulverização do heap (dynamic memory allocation) é uma técnica utilizada em explorações a fim de facilitar a execução de código arbitrário. O termo também é usado para descrever a parte do código-fonte de um exploit que implementa esta técnica. No código, em geral que pulveriza (traduzido à letra) a pilha, tenta colocar uma determinada sequência de bytes (shell code por exemplo) num local pré-determinado da memória de um processo alvo, fazendo com que este "encha" grandes blocos na heap do processo e preencher também nesses blocos os bytes com os valores correctos. Geralmente se aproveita o facto de que esses blocos heap estão mais ou menos no mesmo local, cada vez que o "spray heap" é executado
- Os sprays Heap para browsers, normalmente são implementados em JavaScript e pulverizar a pilha através da criação de grandes cadeias de caracteres Unicode, com o mesmo carácter repetido várias vezes, começando com uma sequência de um caracteres e concatenando-se sucessivamente. Desta forma, o comprimento da string pode crescer exponencialmente até o comprimento máximo permitido pelo mecanismo de script. Quando o comprimento da string desejada seja atingido, shellcode é colocado no final da cadeia. A pilha de pulverização código faz cópias do longa sequência com shellcode e memoriza-as em uma matriz, até o ponto onde a memória tem sido bastante pulverizado para cobrir a área que o exploit assume como alvo. Ocasionalmente, o VBScript é usado no Internet Explorer para criar strings usando a função String.
- DEMO: https://www.youtube.com/watch?v=MqDCn0HoTSw

ASLR

- Address space layout randomization (**ASLR**) é uma técnica de segurança da informação que previne ataques de execução arbitrária de código
- Na intenção de prevenir que um agente mal intencionado, que obteve o controle de um programa em execução em um determinado endereço de memória, pule deste endereço para o de uma função conhecida carregada em memória - a fim de executá-la - a ASLR organiza aleatoriamente a posição de dados chaves no espaço de endereçamento do programa, incluindo a base do executável e a posição da stack, do heap, e de bibliotecas
- ASLR foi originalmente desenvolvido e publicado pelo projeto PaX em Julho de 2001, incluindo um patch para o kernel Linux em Outubro de 2002. Quando aplicado ao kernel, chama-se **KASLR**, de *Kernel address space layout randomization*.

DEP

- **A Prevenção de Execução de Dados** (**DEP**) é um recurso de segurança que pode ajudar a evitar danos ao seu computador contra vírus e outras ameaças à segurança.Programas prejudiciais podem tentar atacar o Windows tentando executar (também conhecido como **executar**) código de locais de memória do sistema reservados para o Windows e outros programas autorizados.
- Este recurso destina-se a impedir a execução de códigos de uma região da memória não-executável em um aplicativo ou serviço. No que ajuda a evitar decorrentes explorações que armazenam código via um vazamento de informações de um buffer, por exemplo. A DEP é executado em dois modos, no de hardware: a DEP é configurada para computadores que podem salvar páginas de memória como não-executável; e na de software: a DEP, tem uma configuração de prevenção limitada para computadores que não têm suporte de hardware para a DEP, como dito anteriormente. A DEP quando configurada para a proteção por software, não protege contra execução de código em páginas de dados, mas em vez de outro tipo de ataque.
- A DEP foi adicionada no Windows XP Service Pack 2 e está incluído no Windows XP Tablet PC Edition da versão 2005, Windows Server 2003 Service Pack 1 e o Windows Vista. Versões posteriores dos sistemas operacionais citados também oferecem suporte ao DEP.

ROP

- **A ROP** (**Return-Oriented Programming** - **Programação Orientada a Retorno**) é uma técnica de exploração de segurança de computador que permite que um invasor execute código na presença de defesas de segurança, como proteção de espaço executável e assinatura de código
- Nessa técnica, um invasor obtém o controle da pilha de chamadas para sequestrar o fluxo de controle do programa e, em seguida, executa sequências de instruções de máquina cuidadosamente escolhidas que já estão presentes na memória da máquina, chamadas "gadgets". Cada gadget normalmente termina em uma instrução de retorno e está localizado em uma sub - rotina dentro do programa existente e / ou código de biblioteca compartilhada. Em conjunto, esses dispositivos permitem que um invasor realize operações arbitrárias em uma máquina, empregando defesas que impedem ataques mais simples.
- FONTE: https://en.wikipedia.org/wiki/Return-oriented_programming

CALL STACK

- Na ciência da computação , uma **pilha de chamadas** é uma estrutura de dados de pilha que armazena informações sobre as sub - rotinas ativas de um programa de computador. Este tipo de pilha também é conhecido como uma **pilha de execução** , **pilha do programa** , **pilha controle** , **pilha de tempo de execução** , ou **pilha máquina**, e é muitas vezes abreviado para apenas " **a pilha** ". Embora a manutenção da pilha de chamadas seja importante para o funcionamento adequado da maioria dos softwares , os detalhes normalmente são ocultos e automáticos nas linguagens de programação de alto nível. Muitos computadores conjuntos de instruções fornecem instruções especiais para manipular pilhas.
- Uma pilha de chamadas é usada para várias finalidades relacionadas, mas a principal razão para uma delas é manter o controle do ponto no qual cada sub-rotina ativa deve retornar o controle quando terminar a execução.
- FONTE: https://en.wikipedia.org/wiki/Call_stack

ASSINATURA DE CÓDIGO

- **A assinatura de código** é o processo de assinar digitalmente executáveis e scripts para confirmar o autor do software e garantir que o código não tenha sido alterado ou corrompido desde que foi assinado. O processo emprega o uso de um hash criptográfico para validar autenticidade e integridade.
- A assinatura de código pode fornecer vários recursos valiosos. O uso mais comum de assinatura de código é fornecer segurança ao implantar; em algumas linguagens de programação, ele também pode ser usado para ajudar a evitar conflitos de namespace. Quase toda implementação de assinatura de código fornecerá algum tipo de mecanismo de assinatura digital para verificar a identidade do autor ou sistema de compilação e uma soma de verificação para verificar se o objeto não foi modificado. Ele também pode ser usado para fornecer informações de versão sobre um objeto ou para armazenar outros metadados sobre um objeto.
- A eficácia da assinatura de código como um mecanismo de autenticação para o software depende da segurança das chaves de assinatura subjacentes. Como ocorre com outras tecnologias de infraestrutura de chave pública (PKI), a integridade do sistema depende de os editores protegerem suas chaves privadas contra acesso não autorizado. As chaves armazenadas no software em computadores de uso geral são suscetíveis de comprometimento. Portanto, é mais seguro e prática recomendada armazenar chaves em dispositivos de hardware criptográficos seguros e invioláveis, conhecidos como módulos de segurança de hardware ou HSMs .
- FONTE: https://en.wikipedia.org/wiki/Code_signing

EIP

- EIP é um registrador em arquiteturas x86 (32 bits). Ele contém o "Ponteiro de instrução estendida" para a pilha. Em outras palavras, ele diz ao computador onde ir ao lado para executar o próximo comando e controla o fluxo de um programa.
- FONTE: https://security.stackexchange.com/questions/129499/what-does-eip-stand-for

CONTROL FLOW

- Na ciência da computação , o fluxo de controle (ou fluxo de controle) é a ordem na qual instruções individuais , instruções ou chamadas de função de um programa imperativo são executadas ou avaliadas. A ênfase no fluxo de controle explícito distingue uma linguagem de programação imperativa de uma linguagem de programação declarativa
- Dentro de uma linguagem de programação imperativa , uma instrução de fluxo de controle é uma instrução, cuja execução resulta na escolha de qual dos dois ou mais caminhos seguir. Para linguagens funcionais não estritas , funções e construções de linguagem existem para alcançar o mesmo resultado, mas elas geralmente não são chamadas de instruções de fluxo de controle
- Um conjunto de instruções é geralmente estruturado como um bloco , que além de agrupar, também define um escopo léxico
- Interrupções e sinais são mecanismos de baixo nível que podem alterar o fluxo de controle de maneira semelhante a uma sub-rotina, mas geralmente ocorrem como resposta a algum estímulo ou evento externo (que pode ocorrer de forma assíncrona), em vez da execução de uma linha. declaração de fluxo de controle
- No nível de linguagem de máquina ou linguagem de montagem , as instruções de fluxo de controle geralmente funcionam alterando o contador de programa . Para algumas unidades de processamento central (CPUs), as únicas instruções de fluxo de controle disponíveis são instruções de ramificação condicionais ou incondicionais , também denominadas saltos.
- FONTE: https://en.wikipedia.org/wiki/Control_flow

CONTROL FLOW GUARD

- O Flow Flow Guard (CFG) é um recurso de segurança de plataforma altamente otimizado que foi criado para combater vulnerabilidades de corrupção de memória. Ao restringir rigorosamente o local de onde um aplicativo pode executar o código, fica muito mais difícil para as explorações executar código arbitrário por meio de vulnerabilidades, como estouro de buffer.
- FONTE: https://docs.microsoft.com/en-us/windows/desktop/secbp/control-flow-guard

/GS (VERIFICAÇÃO DE SEGURANÇA DO BUFFER)

- Detecta alguns estouros de buffer que substituem o endereço de retorno da função, o endereço do manipulador de exceção ou determinados tipos de parâmetros. Causar um estouro de buffer é uma técnica usada por hackers para explorar o código que não impõe restrições de tamanho do buffer.
- FONTE: https://docs.microsoft.com/pt-br/cpp/build/reference/gs-buffer-security-check?view=vs-2019

SMAP

- **Supervisor Mode Access Prevention**) é um recurso de algumas implementações de CPU , como a microarquitetura Intel Broadwell, que permite que os programas do modo supervisor configurem opcionalmente mapeamentos de espaço de memória para que o acesso aos mapeamentos do supervisor cause uma interceptação. Isso torna mais difícil para programas mal-intencionados "enganar" o kernel para usar instruções ou dados de um programa de espaço do usuário.
- FONTE: https://en.wikipedia.org/wiki/Supervisor_Mode_Access_Prevention

KERNEL LINUX

- O sistema Linux é o kernel do sistema, ou seja, um software responsável por controlar as interações entre o hardware e outros programas da máquina. O kernel traduz as informações que recebe ao processador e aos demais elementos eletrônicos do computador. O kernel é, portanto, uma série de arquivos escritos em linguagem C e Assembly, que formam o núcleo responsável por todas as atividades executadas pelo sistema operacional.
- FONTE: https://www.escolalinux.com.br/blog/kernel-do-linux-o-que-e-e-para-que-serve

Conteúdo para estudo e consulta:

- https://www.kaspersky.com.br/blog/exploits-problem-explanation/6010/
- https://www.welivesecurity.com/br/2016/07/29/exploits/
- https://en.wikipedia.org/wiki/Zero-day_(computing)
- https://www.welivesecurity.com/br/2016/09/01/zero-day-termos-de-seguranca/
- https://www.quora.com/What-is-the-difference-between-virtual-address-space-and-virtual-memory
- https://www.diferenca.com/32-bits-e-64-bits/
- https://www.processtec.com.br/artigos/qual-a-diferenca-entre-32-bits-e-64-bits
- https://securityinformationnews.files.wordpress.com/2014/02/bufferoverflow.pdf
- https://pt.stackoverflow.com/questions/3797/o-que-s%C3%A3o-e-onde-est%C3%A3o-o-stack-e-heap
- https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/heap.html
- https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html
- https://www.portugal-a-programar.pt/forums/topic/38005-heap-spray-mais-uma-forma-de-explorar-o-heap-dynamic-memory-allocation/
- https://www.youtube.com/watch?v=Ec4UEtO7dPI – TÉCNICAS HEAP-SPRAY
- https://pt.wikipedia.org/wiki/Data_Execution_Prevention
- https://www.youtube.com/watch?v=yS9pGmY_xuo – ROP
- https://pt.wikipedia.org/wiki/Return-oriented_programming
- https://en.wikipedia.org/wiki/Control_flow

Conteúdo para estudo e consulta 2:

- https://pt.wikibooks.org/wiki/Assembly_x86/Registos
- http://alax.info/blog/656
- https://stackoverflow.com/questions/6607410/understanding-buffer-security-check-gs-compiler-option-in-msvc
- https://pt.wikipedia.org/wiki/Linux_(n%C3%BAcleo)
- https://web.archive.org/web/20160803075007/https://www.ncsi.com/nsatc11/presentations/wednesday/emerging_technologies/fischer.pdf SMEP
- https://en.wikipedia.org/wiki/Control_register#cite_note-4

PRÁTICA: BÁSICA

CONSIDERAÇÕES

- A parte prática requer total dedicação na área, então se você está começando e não tem conhecimentos em PenTest eu recomendo deixar para depois;
- Colocar a prática nas próximas páginas, iria demandar muita coisa, imagine mostrar diversos casos?
- Mas para que isso não se torne uma desculpa para encerrar o livro logo, eu vou deixar diversos conteúdos, principalmente estudo de casos para que vocês possam colocar em prática e analisar;
- Lembrando que os conceitos passados, são totalmente superficial, por isso recomendo se aprofundar, afinal cada conceito demandaria só um ebook para cada um.

PRÁTICA 1: Buffer Overflow

- https://www.youtube.com/watch?v=59_gjX2HxyA –Buffer overflow para PenTesters - Ricardo Longatto
- https://www.youtube.com/watch?v=oGZ01rvbfwE – Introdução ao Buffer Overflow – Helvio Junior
- https://www.youtube.com/watch?v=wLi-dGphpdg&t=202s – Entendendo Buffer Overflow – Helvio Junior
- http://www.cis.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf
- https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture21.pdf

SHELLCODE DATABASE

- http://shell-storm.org/shellcode/

PRÁTICA 2: Developer Exploit Basic

- https://medium.com/bugbountywriteup/windows-expliot-dev-101-e5311ac284a
- https://0x00sec.org/t/getting-cozy-with-exploit-development/5311
- https://www.anitian.com/a-study-in-exploit-development-part-1-setup-and-proof-of-concept/
- https://medium.com/@jain.sm/shell-code-exploit-with-buffer-overflow-8d78cc11f89b

PRÁTICA 3: Ignorando DEP com ROP

- https://bytesoverbombs.io/bypassing-dep-with-rop-32-bit-39884e8a2c4a
- https://samsclass.info/127/proj/rop.htm
- https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/
- http://securitydynamics.blogspot.com/2015/11/an-easy-guide-to-bypass-dep-using-rop.html

PRÁTICA 4: Ignorando DEP e ASLR

- https://0x00sec.org/t/bypass-data-execution-protection-dep/6988
- https://www.exploit-db.com/docs/english/17914-bypassing-aslrdep.pdf

PRÁTICA 5: Heap overflow

- https://www.lume.ufrgs.br/bitstream/handle/10183/36924/000819136.pdf
- https://www.esentire.com/blog/vulnerability-analysis-a-closer-look-at-cve-2015-7547/
- https://github.com/fjserna/CVE-2015-7547
- https://www.sans.edu/student-files/presentations/heap_overflows_notes.pdf
- https://pdfs.semanticscholar.org/a8af/1fd5811fc3b1c9f42a060f6d629ca8d88e46.pdf
- https://www.usenix.org/legacy/event/woot08/tech/full_papers/daniel/daniel_html/index.html
- https://blog.rapid7.com/2019/06/12/heap-overflow-exploitation-on-windows-10-explained/

PRÁTICA 6: Deep Analysis CVE-2016-3820

- https://www.fortinet.com/blog/threat-research/deep-analysis-of-cve-2016-3820-remote-code-execution-vulnerability-in-android-mediaserver.html

PRÁTICA 7: ASLR Bypass

- https://sploitfun.wordpress.com/2015/05/08/bypassing-aslr-part-i/
- https://www.youtube.com/watch?v=Pht6y4p63SE

PRÁTICA 8: AVAST 4.7 ESCALAÇÃO DE PRIVILÉGIOS

- https://www.exploit-db.com/exploits/12406
- https://gist.github.com/mgeeky/023c6f121686cc7ead426a78f8148eb6

PRÁTICA 9: SHELLCODE Examples

- https://blog.rapid7.com/2018/05/03/hiding-metasploit-shellcode-to-evade-windows-defender/
- https://www.youtube.com/watch?v=HSlhY4Uy8SA – LIVE OVERFLOW

PRÁTICA 10: EGGHUNTERS Exploit

- https://medium.com/@rafaveira3/exploit-development-kolibri-v2-0-http-server-egg-hunter-example-1-5e435aa84879
- https://www.exploit-db.com/docs/english/18482-egg-hunter---a-twist-in-buffer-overflow.pdf
- https://www.fuzzysecurity.com/tutorials/expDev/4.html

PRÁTICA 11: CFG Bypass

- https://improsec.com/tech-blog/bypassing-control-flow-guard-in-windows-10
- https://cansecwest.com/slides/2017/CSW2017_HenryLi_How_to_find_the_vulnerability_to_bypass_the_ControlFlowGuard.pdf
- https://www.blackhat.com/docs/asia-17/materials/asia-17-Li-Cross-The-Wall-Bypass-All-Modern-Mitigations-Of-Microsoft-Edge.pdf
- https://www.blackhat.com/docs/us-15/materials/us-15-Zhang-Bypass-Control-Flow-Guard-Comprehensively-wp.pdf

CASES STUDY

- https://github.com/dyjakan/exploit-development-case-studies
- https://medium.com/magebit/magento-web-exploit-case-studies-bac57add8c0e
- http://www.cs.colostate.edu/~cs635/Windows_of_Vulnerability.pdf
- CVE-2017-8601: https://github.com/tunz/js-vuln-db

https://github.com/qazbnm456/awesome-cve-poc

https://www.exploit-db.com/exploits/42246

AWE MATERIAL COURSE

- https://www.offensive-security.com/documentation/advanced-windows-exploitation.pdf

CONCLUSÃO

- Se aprofundar em Desenvolvimento de Exploit exige muito tempo, além dos conhecimentos em engenharia reversa, algoritmos e lógica da programação
- Obviamente o conteúdo passado não foi nem 10% que permite a exploração completa desse assunto, porém eu espero que tenha passado pelo menos uma base
- Caso o seu desejo é se aprofundar mais, eu recomendo a procura de casos de estudo, artigos e palestras
- Não se esqueça de sempre se aprofundar nos fundamentos de cada assunto estudado
- De resto agradeço pela leitura ☺

FINISH

CURTAS AS SEGUINTES PÁGINAS:

CYBER SECURITY UP
EXPERIENCE SECURITY
H4K SECURITY
CAVALEIROS DA INFORMÁTICA
aCCESS