INTRODUÇÃO AO DESENVOLVIMENTO DE EXPLOITS 2 - LÓGICA

PROF. JOAS ANTONIO

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 o assunto
  - 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
  - Necessário saber o básico do inglês.

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

SOFTWARE E SISTEMA

SISTEMA: SIGNIFICADOS

- Conjunto de partes que se relacionam entre si que forma um todo: **UNITÁRIO, COMPLEXO E HARMÔNICO**
- 1: “conjunto de elementos em interação recíproca”; 2. “conjunto de partes reunidas que se relacionam entre si formando uma totalidade”; 3: “conjunto de elementos interdependentes, cujo resultado final é maior do que soma dos resultados que esses elementos teriam caso operassem de maneira isolada”; 4: “conjunto de elementos interdependentes e interagentes no sentido de alcançar um objetivo ou finalidade”; 5: “grupo de unidades combinadas que formam um todo organizado cujas características são diferentes das características das unidades” e 6: “todo organizado ou complexo; um conjunto ou combinação de coisas ou partes, formando um todo complexo ou unitário orientado para uma finalidade”.

SISTEMA: SIGNIFICADOS

- Em resumo, todo e qualquer sistema é formado por partes menores, em tamanho e complexidade, ou seja, partes menores do que o próprio sistema. Essas partes são chamadas, usualmente, de elementos, de órgãos componentes ou, tão somente, de subsistemas.
- A boa integração dos elementos componentes de um sistema é chamada sinergia, determinando o grau de intensidade que as transformações ocorridas em um de seus elementos influenciarão seus outros elementos.
- A alta sinergia de um sistema implica em cumprir sua finalidade ou atingir seu objetivo. Já, a baixa sinergia pode implicar em mau funcionamento do sistema, podendo causar falhas, insuficiência, baixo desempenho etc.

SISTEMA: SIGNIFICADOS

- Assim, a sinergia de um sistema pode ser entendida como a convergência de suas partes que colaboram para afluir a um mesmo lugar, para alcançar um resultado único.
- Pelos vários significados aqui apresentados, pode-se afirmar que o conceito de sistema é, relativamente, simples.
- Contudo, importantíssimo destacar que as definições para a palavra não param por aí. Isto porque a palavra sistema é empregada em várias áreas do conhecimento.

SISTEMA DE SOFTWARE

- Sistemas de Software geram mapas celestes que apresentam a posição das estrelas, dos planetas, dos asteroides etc. Administradores utilizam Sistemas de Software para auxiliar na gestão de seus negócios. Médicos, em geral, utilizam
- Sistemas de Software para o auxílio quanto ao diagnóstico e tratamento médico. Sistemas de Software são sistemas que servem de auxílio à gestão ou tão somente à utilização de outros tipos de sistema.

SOFTWARE

- software significa: “Inform Qualquer programa ou grupo de programas que instrui o hardware sobre a maneira como ele deve executar uma tarefa, inclusive sistemas operacionais, processadores de texto e programas de aplicação”.
- As definições apresentadas pelo dicionário fazem entender software como um programa que é executado por um equipamento qualquer, em outras palavras, um conjunto de instruções a serem executadas (ou processadas), seja para armazenamento, recuperação, redirecionamento, distribuição, no geral, para qualquer forma de manipulação de um dado ou informação.

DADO VS INFORMAÇÃO

- Inicia-se esta diferenciação por suas definições mais genéricas. De acordo com o Dicionário Michaelis de Português Online, dado significa: sm 1 Mat Elemento, princípio ou quantidade conhecida que serve de base à solução de um problema. 2 Ponto de partida em que assenta uma discussão. 3 Princípio ou base para se entrar no conhecimento de algum assunto.
- No caso da palavra informação, o mesmo dicionário apresenta, dentre algumas, as seguintes definições: sf (lat informatione) 1 Ato ou efeito de informar. 2 Transmissão de notícias. 3 Comunicação. 4 Ação de informar-se. 5 Instrução, ensinamento. 6 Transmissão de conhecimentos. 7 Indagação. 8 Opinião sobre o procedimento de alguém. 9 Parecer técnico dado por uma repartição ou funcionário. 10 Investigação. 11 Inquérito.

DADO VS INFORMAÇÃO

- Em suma, a partir de dados é possível o provimento de uma série de informações, cuja relevância e aplicabilidade dependerão daquele(s) que a recebe(m). Observe a lista abaixo. Ela exemplifica uma série de informações, sendo elas:
- Os dias e finais de placa do sistema de rodízio de veículos na cidade de São Paulo;
- Os resultados da descoberta de uma grande investigação científica;
- Os resultados dos jogos do Campeonato Espanhol de Futebol de 2014/2015;
- O total de alunos matriculados em cursos técnicos em todo o território nacional brasileiro nos últimos cinco anos;
- O total de técnicos formados em todo o território nacional brasileiro nos últimos cinco anos;
- O número de abertura de contas celulares nos últimos três anos e
- O percentual de vendas realizadas via internet na última década;

LÓGICA DA PROGRAMAÇÃO

LÓGICA DA PROGRAMAÇÃO

- Estudar a lógica de programação que dá suporte ao raciocínio que o ser humano precisa desenvolver para resolver um problema de Processamento de Dados; sem esse raciocínio, a programação não tem sentido.
- Escolher uma Linguagem de Programação e tentar escrever comandos não produzirá os resultados esperados. O estudo dos principais conceitos de lógica de Programação o ajudará a se tornar um programador melhor.

ALGORITMOS

- Um Algoritmo é uma sequência de passos que visam a atingir objetivos bem definidos em um determinado período de tempo. É composto por instruções claras e bem definidas, com o objetivo de resolver o problema; é um caminho que leva à solução; uma norma de solução a ser trilhada.
- Sempre que executado, sob as mesmas condições, produz o mesmo resultado; uma vez construído, poderá ser traduzido em qualquer Linguagem de Programação, agregado às funcionalidades disponíveis no ambiente de desenvolvimento. Costumamos chamar esse processo de Codificação.

ALGORITMOS

- O Algoritmo é importante, pois representa fielmente o raciocínio lógico e permite resumir os passos necessários na resolução do problema. Dessa forma, você não irá se distrair com detalhes computacionais que podem ser acrescentados mais tarde e focará no que é importante, a lógica da construção do Algoritmo.
- Outro fator importante é que o Algoritmo é independente da Linguagem, permitindo que ele seja codificado em diferentes Linguagens, incorporando as funcionalidades disponíveis nos diversos ambientes.

ALGORITMOS: CONSTRUINDO

- Primeiro, deve-se identificar o problema e o objetivo do Algoritmo, fazendo a leitura do enunciado ou da questão. Em seguida, você deve localizar as entradas de dados, ou seja, as informações que serão fornecidas e, a partir delas, verificar os cálculos ou processamento. A identificação das saídas é a próxima etapa, tendo foco nos resultados.
- Com esses dados em mãos, você determina quais passos são necessários para transformar as entradas nas saídas desejadas. Depois de estabelecidos os passos de transformação, você irá construir o Algoritmo e definir os testes.

ALGORITMOS: CONSTRUINDO

- Passos para Construção de Algoritmo

1. Identificar objetivo;
2. Identificar as “entradas de dados”;
3. Identificar as “saídas de dados”;
4. Determinar o que deve ser feito para transformar entradas em saídas: - Observar a regras; - Obedecer a limitações inclusive do computador; - Determinar ações possíveis de serem realizadas;
5. Construir o Algoritmo;
6. Testar a solução.

DADOS PRIMITIVOS

- Em computação existem apenas 4 tipos de dados primitivos, algumas linguagens subdividem esses tipos de dados em outros de acordo com a capacidade de memória necessária para a variável. Mas de modo geral, os tipos de dados primitivos são:
- **INTEIRO**: Representa valores numéricos negativo ou positivo sem casa decimal, ou seja, valores inteiros.
- **REAL**: Representa valores numéricos negativo ou positivo com casa decimal, ou seja, valores reais. Também são chamados de ponto flutuante.
- **LÓGICO**: Representa valores booleanos, assumindo apenas dois estados, VERDADEIRO ou FALSO. Pode ser representado apenas um bit (que aceita apenas 1 ou 0).
- **TEXTO**: Representa uma sequencia de um ou mais de caracteres, colocamos os valores do tipo TEXTO entre " " (aspas duplas).

DADOS PRIMITIVOS

- Algumas linguagens de programação, dividem esses tipos primitivos de acordo com o espaço necessário para os valores daquela variável. Na linguagem Java por exemplo, o tipo de dados inteiro é dividido em 4 tipos primitivos: **byte**, **short**, **int** e **long**. A capacidade de armazenamento de cada um deles é diferente.
- **byte**: é capaz de armazenar valores entre -128 até 127.
- **short**: é capaz de armazenar valores entre – 32768 até 32767.
- **int**: é capaz de armazenar valores entre –2147483648 até 2147483647.
- **long**: é capaz de armazenar valores entre – 9223372036854775808 até 9223372036854775807.

OPERADORES

- Os operadores são utilizados nas expressões, por exemplo, em 3 + 2, os números 3 e 2 são relacionados por um operador representado pelo sinal + , que significa adição.
- Os operadores se classificam em aritméticos, lógicos e literais. Essa divisão depende do tipo de expressão em que serão inseridos. Existe, ainda, outro tipo de operador, que é o relacional, no qual se comparam informações e o resultado é um valor lógico.
- adição (+)
- subtração (-)
- multiplicação (*)
- divisão (/)
- módulo - resto da divisão - (%)

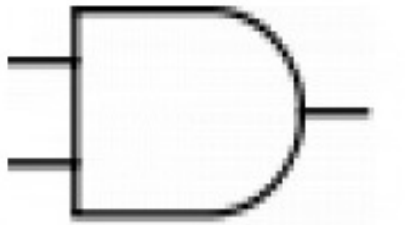
OPERADORES LÓGICOS

- As operações lógicas trabalham sobre valores booleanos, tanto os valores de entrada como o de saída são desse tipo. Os operadores lógicos são: E, OU, NÃO, NÃO-E, NÃO-OU, OU-EXCLUSIVO E NÃO-OU-EXCLUSIVO. Abaixo uma explicação de cada um.

OPERADORES LÓGICOS

Operador E (AND)

O Operador "E" ou "AND" resulta em um valor VERDADEIRO se os dois valores de entrada da operação forem VERDADEIROs, caso contrário o resultado é FALSO. Abaixo a **tabela-verdade** da operação E.

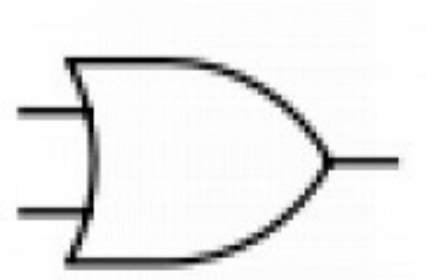


VALOR 1	VALOR 2	OPERAÇÃO E
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

OPERADORES LÓGICOS

Operador OU (OR)

O Operador "OU" ou "OR" resulta em um valor VERDADEIRO se ao menos UM dos dois valores de entrada da operação for VERDADEIRO, caso contrário o resultado é FALSO. Abaixo a **tabela-verdade** da operação OU.

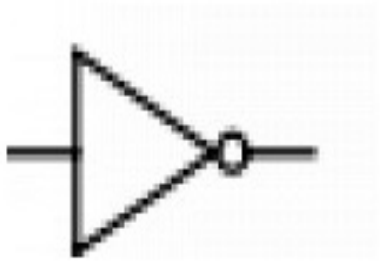


VALOR 1	VALOR 2	OPERAÇÃO OU
VERDADEIRO	**VERDADEIRO**	**VERDADEIRO**
VERDADEIRO	FALSO	**VERDADEIRO**
FALSO	**VERDADEIRO**	**VERDADEIRO**
FALSO	FALSO	FALSO

OPERADORES LÓGICOS

Operador NÃO (NOT)

O Operador "NÃO" ou "NOT" é o único operador que recebe como entrada apenas um valor, e sua função é simplesmente inverter os valores. Ou seja, se o valor de entrada for VERDADEIRO, o resultado será FALSO e se o valor de entrada for FALSO, o resultado será VERDADEIRO. Abaixo a **tabela-verdade** da operação NÃO.

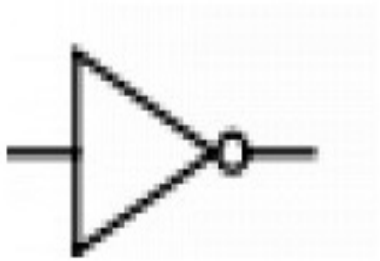


VALOR DE ENTRADA	OPERAÇÃO NÃO
VERDADEIRO	FALSO
FALSO	VERDADEIRO

OPERADORES LÓGICOS

Operador NÃO (NOT)

O Operador "NÃO" ou "NOT" é o único operador que recebe como entrada apenas um valor, e sua função é simplesmente inverter os valores. Ou seja, se o valor de entrada for VERDADEIRO, o resultado será FALSO e se o valor de entrada for FALSO, o resultado será VERDADEIRO. Abaixo a **tabela-verdade** da operação NÃO.



VALOR DE ENTRADA	OPERAÇÃO NÃO
VERDADEIRO	FALSO
FALSO	VERDADEIRO

OPERADORES LÓGICOS

Operador NÃO-E (NAND)

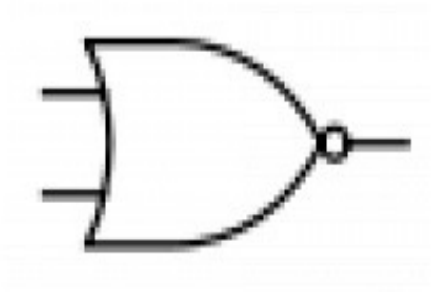
O Operador "NÃO-E" ou "NAND" é o contrário do operador E (AND), ou seja, resulta em VERDADEIRO, se ao menos um dos dois valores for FALSO, na verdade este é o operador E (AND) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-E.

VALOR 1	VALOR 2	OPERAÇÃO NAND
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	VERDADEIRO

OPERADORES LÓGICOS

Operador NÃO-OU (NOR)

O Operador "NÃO-OU" ou "NOR" é o contrário do operador OU (OR), ou seja, resulta em VERDADEIRO, se os dois valores forem FALSO, na verdade este é o operador OU (OR) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-OU.

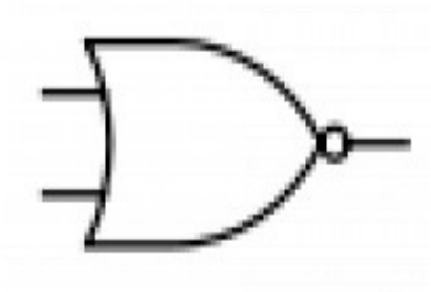


VALOR 1	VALOR 2	OPERAÇÃO NOR
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	VERDADEIRO

OPERADORES LÓGICOS

Operador NÃO-OU (NOR)

O Operador "NÃO-OU" ou "NOR" é o contrário do operador OU (OR), ou seja, resulta em VERDADEIRO, se os dois valores forem FALSO, na verdade este é o operador OU (OR) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-OU.

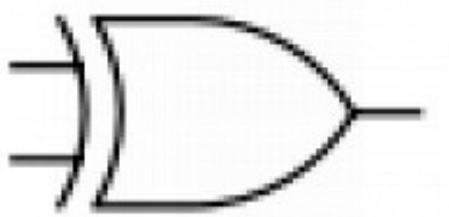


VALOR 1	VALOR 2	OPERAÇÃO NOR
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	VERDADEIRO

OPERADORES LÓGICOS

Operador OU-EXCLUSIVO (XOR)

O Operador "OU-EXCLUSIVO" ou "XOR" é uma variação interessante do operador OU (OR), ele resulta em VERDADEIRO se apenas um dos valores de entrada for VERDADEIRO, ou seja, apenas se os valores de entrada forem DIFERENTES. Abaixo a **tabela-verdade** da operação OU-EXCLUSIVO.



VALOR 1	VALOR 2	OPERAÇÃO XOR
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

OPERADORES LÓGICOS

Operador NÃO-OU-EXCLUSIVO (XNOR)

O Operador "NÃO-OU-EXCLUSIVO" ou "XNOR" é o contrário do operador OU-EXCLUSIVO (XOR), ou seja, resulta VERDADEIRO se os valores de entrada forem IGUAIS. Observe a tabela abaixo:

VALOR 1	VALOR 2	OPERAÇÃO XNOR
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	VERDADEIRO

ESTRUTURA DE CONDIÇÃO

- **Estrutura de seleção** (*expressão condicional* ou ainda *construção condicional*) é, na ciência da computação, uma estrutura de desvio do fluxo de controle presente em linguagens de programação que realiza diferentes computações ou ações dependendo se a seleção (ou condição) é verdadeira ou falsa, em que a expressão é processada e transformada em um valor booleano. Nas linguagens de programação, usamos as palavras em inglês para expressar uma estrutura de seleção, como *if*, *else if* e *else*

ESTRUTURA DE CONDIÇÃO

- **Estrutura de seleção** (*expressão condicional* ou ainda *construção condicional*) é, na ciência da computação, uma estrutura de desvio do fluxo de controle presente em linguagens de programação que realiza diferentes computações ou ações dependendo se a seleção (ou condição) é verdadeira ou falsa, em que a expressão é processada e transformada em um valor booleano. Nas linguagens de programação, usamos as palavras em inglês para expressar uma estrutura de seleção, como *if*, *else if* e *else*

ESTRUTURA DE REPETIÇÃO

- Dentro da lógica de programação é uma estrutura que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador.
- São utilizadas, por exemplo, para repetir ações semelhantes que são executadas para todos os elementos de uma lista de dados, ou simplesmente para repetir um mesmo processamento até que a condição seja satisfeita.

ESTRUTURA DE REPETIÇÃO

While:

- É dentre as 3 a mais simples.
- Repete um bloco de código enquanto uma condição permanecer verdadeira
- Caso a condição seja falsa, os comandos dentro do while não serão executados e a execução continuará com os comandos após o while
- A repetição do while é controlada por uma condição que verifica alguma variável. Porém para que o while funcione corretamente é importante que essa variável sofra alteração dentro do while. Ex: um contador.
- Após entrar dentro da repetição, o bloco de comandos sempre será executado, mesmo que dentro do bloco a variável que está controlando a execução seja alterada.

ESTRUTURA DE REPETIÇÃO

Do While:

- Muito parecido com o while, porém tem uma diferença crucial: condição é verificada após executar o bloco de comandos.
- Há uma bloco de comandos e logo depois uma verificação. Assim caso a variável condicional for alterada dentro do bloco de comandos, isso afetará a validação da condição.
- A escolha entre while e do while é mínima, então dependerá do bom senso do programador, que optará pela estrutura que deixar o algoritmo mais simples e legível.

ESTRUTURA DE REPETIÇÃO

For:

- O For é utilizado para executar um conjunto de comandos executado por um número X de vezes.
- É passada uma situação inicial, uma condição e uma ação a ser executada a cada repetição.
- Uma variável é inicializada com uma valor inicial.
- Essa variável é utilizada para controlar a quantidade de vezes em que o conjunto de comandos será executado.
- E ao final do conjunto de comandos a variável sempre sofrerá uma alteração, aumentando ou diminuindo de acordo com a lógica utilizada.

ESTRUTURA DE REPETIÇÃO

Foreach:

- O FOREACH é uma simplificação do operador FOR.
- Permite acessar cada elemento individualmente iterando sobre toda a coleção sem a necessidade de informação de índices.

APRENDA!

Aprenda a lógica com portugol ou visualg:

https://www.youtube.com/watch?v=6OIADpFImtc

https://www.youtube.com/watch?v=8mei6uVttho&list=PLHz_AreHm4dmSj0MHol_aoNYCSGFqvfXV&index=1

#Curso em video

LINGUAGEM DE PROGRAMAÇÃO - DESENVOLVIMENTO DE EXPLOIT

LINGUAGEM DE PROGRAMAÇÃO

A **linguagem de programação** é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

LINGUAGEM DE PROGRAMAÇÃO: ALTO NIVEL

- **Linguagem de programação de alto nível** é como se chama, na Ciência da Computação de linguagens de programação, uma linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana. Desse modo, as linguagens de alto nível não estão diretamente relacionadas à arquitetura do computador. O programador de uma linguagem de alto nível não precisa conhecer características do processador, como instruções e registradores. Essas características são abstraídas na linguagem de alto nível.
- Por se tratar de uma classificação subjetiva, isto é, sem limites bem definidos, não é possível afirmar que "determinada linguagem pode ser mais *humana* que outra". Apesar disso, por questão de praticabilidade e objetividade, a classificação geralmente se limita em "linguagem de alto nível" e "linguagem de baixo nível".

LINGUAGEM DE PROGRAMAÇÃO: ALTO NIVEL (exemplos)

- ASP
- ActionScript
- C/C++
- C#
- Pascal/Object Pascal
- Euphoria
- Java
- JavaScript
- Lua
- MATLAB
- PHP
- Python
- R
- Ruby
- Tcl
- Basic/Visual Basic

LINGUAGEM DE PROGRAMAÇÃO: BAIXO NÍVEL

- **Linguagem de programação de baixo nível** trata-se de uma linguagem de programação que segue as características da arquitetura do computador. Assim, utiliza somente instruções que serão executadas pelo processador, em contrapartida as linguagens de alto nível que utilizam de instruções abstratas. Nesse sentido, as linguagens de baixo nível estão diretamente relacionadas com a arquitetura do computador.

LINGUAGEM DE PROGRAMAÇÃO: BAIXO NÍVEL (exemplos)

- Binário
- Assembly

CÓDIGO DE MÁQUINA

- Um programa em **código de máquina** consiste de uma sequência de bytes que correspondem a instruções a serem executadas pelo processador. As instruções do processador, chamadas de opcodes, são representadas por valores em hexadecimal.

CÓDIGO DE MÁQUINA

- Para se programar em código de máquina, deve-se obter os códigos de instruções do processador utilizado contendo opcodes, operandos e formatos de cada instrução.
- Por esse motivo foi criada uma linguagem de programação chamada Assembly, composta de códigos mnemônicos que expressam as mesmas instruções do processador, embora escritos em acrônimos da língua inglesa, tais como *mov* ou *rep*, em vez de opcodes.

LINGUAGENS PARA ESTUDO

- Algumas linguagens de programação são exclusivas para determinados assuntos, mas outras linguagens são bem interessantes de estudar e aprender, afinal elas ajudam a compreender a funcionalidade de sistemas, softwares e até mesmo em desenvolver em um nível abstrato maior.
- Além disso, para desenvolver exploits é essencial o conhecimento em pelo menos nessas 4 linguagens.
- Então segue a lista do que estudar:

1. C e C++
2. Python
3. Assembly

LINGUAGEM C

- **C** é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada por Organização Internacional para Padronização (ISO), criada em 1972 por Dennis Ritchie na empresa AT&T Bell Labs para desenvolvimento do sistema operacional Unix (originalmente escrito em Assembly).
- C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para C. C tem influenciado muitas outras linguagens de programação (por exemplo, a linguagem Java), mais notavelmente C++, que originalmente começou como uma extensão para C.

LINGUAGEM PYTHON

- ***Python*** é uma linguagem de programação de alto nível,interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991.Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. O padrão *de facto* é a implementação CPython.
- A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e *frameworks* desenvolvidos por terceiros.

LINGUAGEM ASSEMBLY

- ***Assembly*** ou **linguagem de montagem** é uma notação legível por humanos para o código de máquina que uma arquitetura de computador específica usa, utilizada para programar códigos entendidos por dispositivos computacionais, como microprocessadores e microcontroladores. O código de máquina torna-se legível pela substituição dos valores em bruto por símbolos chamados mnemónicos.

- OBS: Clique nos hiperlinks para saber o significado de cada conceito.

LINGUAGEM C: APRENDA

- https://www.youtube.com/watch?v=FH7YrE0RjWE
- https://www.aulaead.com/courses/curso-gratis-linguagem-c
- https://www.alura.com.br/cursos-online-programacao/c
- https://www.youtube.com/watch?v=oZeezrNHxVo = PAPO BINÁRIO

LINGUAGEM PYTHON: APRENDA

- https://www.youtube.com/playlist?list=PLHz_AreHm4dlKP6QQCekuIPky1CiwmdI6
- https://www.youtube.com/watch?v=j94IGZmwtYI&list=PLesCEcYj003QxPQ4vTXkt22-E11aQvoVj
- https://github.com/dsacademybr/PythonFundamentos
- https://github.com/search?q=python+curso

LINGUAGEM ASSEMBLY: APRENDA

- https://www.youtube.com/watch?v=hxguUS5HxvM&list=PLxTkH01AauxRm0LFLlOA9RR5O6hBLqBtC
- https://www.youtube.com/watch?v=2giDgeXpJE0
- https://www.youtube.com/watch?v=AZan-9s49tg&list=PLZ8dBTV2_5HS6FDb_A7G9fRtjlXM98o6k

CONCEITO DE ENGENHARIA REVERSA

ENGENHARIA REVERSA

- Em termos amplos, engenharia reversa é o processo de entender como algo funciona através da análise de sua estrutura e de seu comportamento. É uma arte que permite conhecer como um sistema foi pensado ou desenhado sem ter contato com o projeto original.
- Não restrinjamos, portanto, a engenharia reversa à tecnologia. Qualquer um que se disponha a analisar algo de forma minuciosa com o objetivo de entender seu funcionamento a partir de seus efeitos ou estrutura está fazendo engenharia reversa. Podemos dizer, por exemplo, que Carl G. Jung (conhecido como o pai da psicologia analítica) foi um grande engenheiro reverso ao introduzir conceitos como o de inconsciente coletivo através da análise do comportamento humano.
- No campo militar e em época de guerras é muito comum assegurar que inimigos não tenham acesso às armas avançadas: aviões, tanques e outros dispositivos, pois é importante que adversários não desmontem esses equipamentos, não entendam seu funcionamento e, consequentemente, que não criem versões superiores deles ou encontrem falhas que permitam inutilizá-los com mais facilidade. Na prática, é evitar a engenharia reversa.
- {% hint style="info" %} Vale a pena assistir ao filme Jogo da Imitação (*Imitation Game*) que conta a história do criptoanalista inglês Alan Turing, conhecido como o pai da ciência de computação teórica, que quebrou a criptografia da máquina nazista Enigma utilizando engenharia reversa. {% endhint %}

ENGENHARIA REVERSA: SOFTWARE

- Este livro foca na engenharia reversa de software, ou seja, no processo de entender como uma ou mais partes de um programa funcionam, sem ter acesso a seu código-fonte. Focaremos inicialmente em programas para a plataforma x86 (de 32-bits), rodando sobre o sistema operacional Windows, da Microsoft, mas vários dos conhecimentos expressos aqui podem ser úteis para engenharia reversa de software em outros sistemas operacionais, como o GNU/Linux e até mesmo em outras plataformas, como ARM.

ENGENHARIA REVERSA: SOFTWARE

- Assim como o *hardware, o software* também pode ser desmontado. De fato, existe uma categoria especial de *softwares* com esta função chamados de *disassemblers*, ou desmontadores. Para explicar como isso é possível, primeiro é preciso entender como um programa de computador é criado atualmente. Farei um resumo aqui, mas entenderemos mais a fundo em breve.
- A parte do computador que de fato executa os programas é o chamado processador. Nos computadores de mesa (*desktops*) e *laptops* atuais, normalmente é possível encontrar processadores fabricados pela Intel ou AMD. Para ser compreendido por um processador, um programa precisa falar sua língua: a **linguagem (ou código) de máquina**.
- Os humanos, em teoria, não falam em linguagem de máquina. Bem, alguns falam, mas isso é outra história. Acontece que para facilitar a criação de programas, algumas boas almas começaram a escrever programas onde humanos escreviam código (instruções para o processador) numa linguagem mais próxima da falada por eles (Inglês no caso). Assim nasceram os primeiros **compiladores**, que podemos entender como programas que "traduzem" códigos em linguagens como **Assembly** ou **C** para código de máquina.

ENGENHARIA REVERSA: SOFTWARE

- Um programa é então uma série de instruções em código de máquina. Quem consegue olhar pra ele desta forma, consegue entender sua lógica, mesmo sem ter acesso ao código-fonte que o gerou. Isso vale para praticamente qualquer tipo de programa, seja ele criado em linguagens onde a compilação ocorre separada da execução, como C, C++, Pascal, Delphi, Visual Basic, D, Go e até mesmo em linguagens onde a compilação ocorre junto à execução, como Python, Ruby, Perl ou PHP. Lembre-se que o processador só entende código de máquina e para ele não importa qual é o código fonte, ou se a linguagem é compilada(análise e execução separadas) ou interpretada(análise e execução juntas). Então é importante notar que, para o processador poder executar, "tudo tem que acabar em linguagem de máquina". Qualquer um que a conheça será capaz de inferir qual lógica o programa possui.

ENGENHARIA REVERSA: APLICAÇÃO

Análise de malware

- Naturalmente, os criadores de programas maliciosos não costumam compartilhar seus códigos-fonte com as empresas de segurança de informação. Sendo assim, analistas que trabalham nessas empresas ou mesmo pesquisadores independentes podem lançar mão da engenharia reversa afim de entender como essas ameaças digitais funcionam e então poder criar suas defesas.

Análise de vulnerabilidade

- Alguns *bugs* encontrados em *software* podem ser exploráveis por outros programas. Por exemplo, uma falha no componente SMB do Windows permitiu que a NSA desenvolvesse um programa que dava acesso a qualquer computador com o componente exposto na Internet. Para encontrar tal vulnerabilidade, especialistas precisam conhecer sobre engenharia reversa, dentre outras áreas.

ENGENHARIA REVERSA: APLICAÇÃO

Correção de *bugs*

- Às vezes um *software* tem um problema e por algum motivo você ou sua empresa não possui mais o código-fonte para repará-lo ou o contrato com o fornecedor que desenvolveu a aplicação foi encerrado. Com engenharia reversa, pode ser possível corrigir tal problema.

Mudança e adição de recursos

- Mesmo sem ter o código-fonte, é possível também alterar a maneira como um programa se comporta. Por exemplo, um programa que salva suas configurações num diretório específico pode ser instruído a salvá-las num compartilhamento de rede.
- Adicionar um recurso é, em geral, trabalhoso, mas possível.

ENGENHARIA REVERSA: APLICAÇÃO

(Anti-)pirataria

- *Software* proprietário costuma vir protegido contra pirataria. Você já deve ter visto programas que pedem número de série, chave de registro, etc. Com engenharia reversa, os chamados *crackers* são capazes de quebrar essas proteções. Por outro lado, saber como isso é feito é útil para programadores protegerem melhor seus programas.

ENGENHARIA REVERSA: EXEMPLOS

- Um bom exemplo de uso da engenharia reversa é o caso da equipe que desenvolve o LibreOffice: mesmo sem ter acesso ao código fonte, eles precisam entender como o Microsoft Office funciona, a fim de que os documentos criados nos dois produtos sejam compatíveis. Outros bons exemplos incluem:
- o **Wine**, capaz de rodar programas feitos para Windows no GNU/Linux;
- o **Samba** que permite que o GNU/Linux apareça e interaja em redes Windows;
- o **Pidgin** que conecta numa série de protocolos de mensagem instantânea;
- e até um sistema operacional inteiro chamado **ReactOS**, que lhe permite executar seus aplicativos e drivers favoritos do Windows em um ambiente de código aberto e gratuito.
- Todos estes são exemplos de implementações em software livre, que tiveram de ser criadas a partir da engenharia reversa feita em programas e/ou protocolos de rede proprietários.

FONTE: LIVRO MENTE BINÁRIA

APRENDA ENGENHARIA REVERSA

APRENDA!

- QUE TAL APRENDER ENGENHARIA REVERSA?
- AFINAL, PARA DESENVOLVER EXPLOITS É NECESSÁRIO CONHECER DE ENGENHARIA REVERSA PARA QUEBRAR A SEGURANÇA DE UM SOFTWARE E IR ATRÁS DE BRECHAS
- ENTÃO A SEGUIR VOU DEIXAR MATERIAIS PARA ENTENDER UM POUCO:

1. https://github.com/mentebinaria/fundamentos-engenharia-reversa
2. https://github.com/rumbleh/engenharia-reversa
3. https://github.com/uniciv/Introducao-Engenharia-Reversa
4. https://github.com/felipesanches/FISL2015_HardwareLivre_e_EngenhariaReversa
5. https://www.youtube.com/watch?v=IkUfXfnnKH4
6. https://www.youtube.com/watch?v=PG510bhFgXY
7. https://www.youtube.com/watch?v=af0kbx8KuWo
8. https://github.com/wtsxDev/reverse-engineering

APRENDA!

LABORATÓRIOS

- https://medium.com/bugbountywriteup/tagged/reverse-engineering
- https://medium.com/bugbountywriteup/tokyowesterns-ctf-4th-2018-writeup-part-1-78558397cb7b
- http://www.cs.utexas.edu/~harold/rev.html
- https://shellterlabs.com/pt/account/login/
- https://picoctf.com/
- https://www.hackthebox.eu/

BINARY EXPLOIT

BINARY EXPLOIT

- É a arte de explorar arquivos binários, ou seja, já compilados, para chegar em um objetivo, seja:

1. Encontrar uma falha na aplicação
2. Um meio de burlar o software
3. Quebrar a segurança para realizar outras explorações
4. Manipular a funcionalidade da aplicação

- Isso tudo é ligado a Buffer Overflow que você pode aprender no volume 1 desse livro.

BINARY EXPLOIT: CONCEITO

- A exploração binária é o processo de subverter um aplicativo compilado, de forma que viole alguns limites de confiança de uma maneira que seja vantajosa para você, o invasor. Neste módulo, vamos nos concentrar na corrupção de memória. Ao abusar de vulnerabilidades que corrompem memória no software, geralmente podemos reescrever informações críticas sobre o estado do aplicativo de uma maneira que nos permita elevar privilégios dentro do contexto de um aplicativo específico (como um servidor de desktop remoto) ou executar cálculos arbitrários ao seqüestrar o fluxo de controle e executar o código de nossa escolha.
- Se você está tentando encontrar erros nos programas C compilados, é importante saber o que você está procurando. Comece identificando onde os dados que você envia para o programa são usados. Se seus dados estiverem armazenados em um buffer, anote o tamanho deles. Programar em C sem erros é muito difícil e o CERT C Coding Standard cataloga muitas das maneiras pelas quais os erros podem ocorrer. Prestar atenção às APIs comumente mal utilizadas pode ser um caminho rápido para o sucesso.
- Depois que uma vulnerabilidade é identificada, ela deve ser usada para comprometer a integridade do programa, no entanto, existem várias maneiras de atingir esse objetivo. Para programas como servidores da web, obter as informações de outro usuário pode ser o objetivo final. Em outros, alterar suas permissões pode ser útil, por exemplo, alterar as permissões de um usuário local para o administrador.

APRENDA!

APRENDA

- https://tcode2k16.github.io/blog/posts/picoctf-2018-writeup/binary-exploitation/
- https://www.youtube.com/watch?v=cNN54833LiU
- https://www.youtube.com/watch?v=akCce7vSSfw
- https://www.youtube.com/watch?v=iyAyN3GFM7A&list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN
- https://old.liveoverflow.com/binary_hacking/

APRENDA!

LABORATÓRIOS

- https://github.com/offensive-security/exploitdb-bin-sploits
- https://github.com/scwuaptx/HITCON-Training
- https://github.com/Billy-Ellis/Exploit-Challenges
- https://github.com/r0hi7/BinExp
- https://github.com/jmpews/pwn2exploit

ROP EXPLOIT

ROP:CONCEITO

- Programação orientada a retorno (também chamada de“ empréstimo de parte à la krahmer ”) é uma técnica de exploração de segurança de computador na qual o invasor usa o controle da pilha de chamadas para executar indiretamente instruções de máquina escolhidas a dedo ou grupos de instruções de máquina imediatamente antes da instrução de retorno nas sub-rotinas do código de programa existente, de maneira semelhante à execução de um interpretador de código encadeado.

- "Como todas as instruções executadas são de áreas de memória executável no programa original, isso evita a necessidade de injeção direta de código e contorna a maioria das medidas que tentam impedir a execução de instruções da memória controlada pelo usuário".

- https://www.rapid7.com/resources/rop-exploit-explained/

APRENDA!

APRENDA

- https://www.youtube.com/watch?v=yS9pGmY_xuo
- https://www.youtube.com/watch?v=MSy0rdi1vbo
- https://www.youtube.com/watch?v=wDosab_Y4Hs

APRENDA!

LABORATÓRIOS

- https://github.com/bkerler/exploit_me
- https://github.com/Gallopsled/pwntools
- https://github.com/Billy-Ellis/Exploit-Challenges

AUXILIARY TOOLS

GDB

- O GNU Debugger, mais conhecido por GDB, é um depurador do GNU. Ele pode ser usado para depuração em sistemas Unix-like e suporta muitas linguagens de programação, como a C, C++, Fortran, Objective-C, Pascal, Java, e parcialmente outras.
- https://www.onlinegdb.com/

GDB: O QUE É

- O GDB (*GNU Project Debugger*) é uma ferramenta para:
  - observar um programa enquanto este executa
  - ver o estado no momento que a execução falha
- Permite:
  - iniciar a execução de um programa
  - executar linha-a-linha
  - especificar pontos de paragem
  - imprimir valores de variáveis

GDB: USO

- O GDB opera sobre ficheiros executáveis (não diretamente sobre o código-fonte)
- Para usar o GDB com um programa em C devemos compilar com opção -g:

```
$ gcc -g -o programa programa.c
```

- A opção -g indica ao compilador para incluir no executável informação extra para debugging

- https://www.dcc.fc.up.pt/~pbv/aulas/progimp/teoricas/introgdb.html

GDB: APRENDA!

- https://www.youtube.com/watch?v=hlA44WNVQYQ
- http://www.lrc.ic.unicamp.br/~luciano/courses/mc202-2s2009/tutorial_gdb.txt
- https://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html

GCC

- O GNU Compiler Collection é um conjunto de compiladores de linguagens de programação produzido pelo projecto GNU para construir um sistema operativo semelhante ao Unix livre.
- https://gcc.gnu.org/

GCC: USO

```c
#Nome do arquivo: prog
int main(){
printf("Hello World");
return 0;
}
```

$ gcc prog.c –o prog

- onde *prog.c* é o nome do arquivo que contém o código. Os outros dois parâmetros, *-o prog*, indicam o arquivo de saída do compilador — o arquivo executável que conterá o programa. Você não verá nenhuma mensagem na tela se a compilação ocorrer sem problemas; o compilador só diz alguma coisa quando ocorrem erros.
- Você precisa especificar o nome do arquivo executável de saída pois o padrão, por razões históricas, é usar o arquivo *a.out*. Em geral, usamos o mesmo nome do arquivo de código, tirando a extensão *.c*. Veja que, ao contrário do Windows, o Linux não precisa da extensão *.exe* para reconhecer um arquivo executável; ele utiliza os atributos de permissão do arquivo para saber se ele é executável, dos quais o gcc já cuida automaticamente.

GCC: USO

- Para executar o programa, a maneira mais “universal” é digitar o seguinte comando no terminal:

$ ./prog

GCC: APRENDA

- http://inf.ufes.br/~pdcosta/ensino/2016-2-estruturas-de-dados/material/GuiaRapido_EDI.pdf
- https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html
- https://www.wikihow.com/Compile-a-C-Program-Using-the-GNU-Compiler-(GCC)
- https://www.youtube.com/watch?v=0iNp9Avtr6c

GHIDRA

- A Agência de Segurança Nacional dos Estados Unidos (NSA) apresentou ao público na conferência da RSA uma nova ferramenta de engenharia reversa conhecida como GHIDRA, que agora está **disponível gratuitamente na versão 9.0 e como um projeto de *open source***. Essa ferramenta, desenvolvida pela NSA, tem sido usada internamente há vários anos como parte do trabalho da agência para a cibersegurança nacional.
- Com o GHIDRA, os profissionais no campo da cibersegurança podem ter ao alcance de suas mãos um instrumento baseado em Java para análise de malwares que permite compreender e descubrir vulnerabilidades em suas redes e sistemas e que funciona como uma alternativa gratuita de uma ferramenta bem conhecida de engenharia reversa, como o IDA Pro. Suas principais características incluem um conjunto de ferramentas para análise de código compilado em diversas plataformas, como Windows, Mac OS e Linux; capacidade de desmontar, montar, descompilar, representar graficamente e executar scripts; e a capacidade de fazer/desfazer ações. Por outro lado, a ferramenta suporta uma ampla variedade de instruções de processador e formatos executáveis e os usuários têm a possibilidade de desenvolver seus próprios plug-ins ou scripts para o GHIDRA através de sua API.

GHIDRA: APRENDA

- https://www.mentebinaria.com.br/treinamentos/curso-de-ghidra-r9/
- https://www.shogunlab.com/blog/2019/04/12/here-be-dragons-ghidra-0.html
- https://www.youtube.com/watch?v=tH9A2zVIzKI

EDB

- O depurador edb é um equivalente em Linux do famoso "depurador Olly" na plataforma Windows. Um dos principais objetivos desse depurador é a modularidade. Algumas de suas características são:
- Interface GUI intuitiva
- As operações usuais de depuração (etapa / etapa / substituição / execução / interrupção)
- Pontos de interrupção condicionais
- O núcleo de depuração é implementado como um plug-in para que as pessoas possam ter substituições drop-in. Obviamente, se uma determinada plataforma tiver várias APIs de depuração disponíveis, você poderá ter um plug-in que implementa qualquer uma delas.
- Análise básica de instruções
- Exibir / despejar regiões de memória
- Inspeção de endereço eficaz
- A visualização de despejo de dados é tabulada, permitindo que você tenha várias visualizações de memória abertas ao mesmo tempo e alterne rapidamente entre elas.
- Importação e geração de mapas de símbolos
- Vários plugins

EDB: APRENDA

- https://tools.kali.org/reverse-engineering/edb-debugger
- https://www.youtube.com/watch?v=yFOW_gAujIs

OLLYDBG

- O OllyDbg é um depurador de análise de nível de montador de 32 bits para o Microsoft Windows. A ênfase na análise de código binário o torna particularmente útil nos casos em que a fonte está indisponível.
- **Recursos:**

- Interface de usuário intuitiva, sem comandos enigmáticos
- Análise de código - rastreia registros, reconhece procedimentos, loops, chamadas de API, comutadores, tabelas, constantes e seqüências de caracteres
- Carrega e depura diretamente DLLs
- Verificação de arquivo de objeto - localiza rotinas de arquivos e bibliotecas de objetos
- Permite etiquetas, comentários e descrições de funções definidas pelo usuário
- Compreende as informações de depuração no formato Borland®
- Salva patches entre as sessões, grava-os de volta no arquivo executável e atualiza os reparos

OLLYDBG: RECURSOS

- Arquitetura aberta - muitos plugins de terceiros estão disponíveis
- Sem instalação - sem lixo nos diretórios do registro ou do sistema
- Depura aplicativos multithread
- Anexa a programas em execução
- O desmontador configurável suporta os formatos MASM e IDEAL
- MMX, 3DAgora! e tipos de dados SSE e instruções, incluindo extensões Athlon
- Suporte completo a UNICODE
- Reconhece dinamicamente as strings ASCII e UNICODE - também no formato Delphi!
- Reconhece construções de código complexas, como chamada para ir para o procedimento
- Decodifica chamadas para mais de 1900 funções API padrão e 400 C
- Fornece ajuda contextual sobre funções da API a partir do arquivo de ajuda externo
- Define pontos de interrupção condicionais, de log, de memória e de hardware
- Rastreia a execução do programa, registra argumentos de funções conhecidas

OLLYDBG: RECURSOS

- Mostra correções
- Rastreia dinamicamente quadros de pilha
- Procura comandos imprecisos e sequências binárias mascaradas
- Pesquisa toda a memória alocada
- Localiza referências ao intervalo constante ou de endereço
- Examina e modifica a memória, define pontos de interrupção e interrompe o programa on-the-fly
- Monta comandos no formato binário mais curto

OLLYDBG: APRENDA!

- https://www.youtube.com/watch?v=xfFVIz5jIeY
- https://www.youtube.com/watch?v=wQhLqdCduQI
- https://www.youtube.com/watch?v=hecjjPVWE_Q
- https://medium.com/@leonardomarciano/engenharia-reversa-3-utilizando-ollydgb-parte-1-4ef716023db5

EXPLOIT DEVELOPER

SOBRE

- Buffer Overflow and Binary Exploitation
- Reverse Engineering
- Bypass prevention

CURSOS

- https://www.youtube.com/watch?v=tVDuuz60KKc
- https://www.youtube.com/watch?v=c7H1W4BmZ6g
- https://www.youtube.com/watch?v=nCdBWJI_5XY
- https://www.youtube.com/watch?v=SwyEDNlWdtw
- https://www.youtube.com/watch?v=3SY2SI60C2s
- https://www.cybrary.it/video/exploit-development-introduction-part-1/

BUFFER OVERFLOW

- https://medium.com/bugbountywriteup/windows-expliot-dev-101-e5311ac284a
- https://github.com/freddiebarrsmith/Buffer-Overflow-Exploit-Development-Practice
- https://0x00sec.org/t/buffer-overflow-exploitation/3846
- https://www.youtube.com/watch?v=59_gjX2HxyA
- https://www.youtube.com/watch?v=H2ZTTQX-ma4
- https://www.youtube.com/watch?v=qjWs___hQcE
- https://www.youtube.com/watch?v=d1U-czwATiM
- https://www.youtube.com/watch?v=1S0aBV-Waeo

BINARY EXPLOITATION

- https://www.youtube.com/watch?v=RoaGM8eRzK4
- https://www.youtube.com/watch?v=iyAyN3GFM7A&list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN
- https://www.youtube.com/watch?v=p7bveH_ocnM

BYPASS PREVETION

- NX + ASLR: http://intx0x80.blogspot.com/2018/04/bypass-aslrnx-part-1.html
- NX: https://www.youtube.com/watch?v=SEW6FlcP_m0
- NX + ASLR2: https://github.com/nnamon/linux-exploitation-course/blob/master/lessons/9_bypass_ret2plt/lessonplan.md
- PLT E GOT = BYPASS ASLR: https://www.ret2rop.com/2018/08/return-to-plt-got-to-bypass-aslr-remote.html
- ASLR+NX+RET2PLT: http://shoxx-website.com/2016/05/exploitation-bypass-aslrnx-with-ret2plt.html
- ASLR: https://www.youtube.com/watch?v=OgMGPxl2fUQ
- NX+ASLR: https://www.youtube.com/watch?v=yFlUSnX5Bqk
- ROP: https://www.youtube.com/watch?v=5FJxC59hMRY
- ASLR: https://www.youtube.com/watch?v=CyazDp-Kkr0
- ROP: https://www.youtube.com/watch?v=gWU2yOu0COk

CONCLUSÃO

CONCLUSÃO

- Para explorar um software com o objetivo de quebra-lo, vai ser necessários diversos estudos profundos sobre sistemas e arquiteturas
- Além disso, aprender a arte de desenvolvimento de exploits é bastante complicado e isso não seria possível apenas em um pequeno livro
- Estudar e percorrer CTFs é essencial para aprender mais ainda sobre as técnicas apresentadas
- Portanto, estude os fundamentos e aprofunde o seu conhecimento em desenvolvimento e sistemas.

REFERÊNCIAS

https://dicasdeprogramacao.com.br/tipos-de-dados-primitivos/
https://www.devmedia.com.br/operadores-logicos-aritmeticos-e-de-atribuicao-do-php/25628
https://dicasdeprogramacao.com.br/operadores-logicos/
https://pt.wikipedia.org/wiki/Estrutura_de_sele%C3%A7%C3%A3o
https://podprogramar.com.br/logica-de-programacao-estruturas-de-repeticao/
https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_de_alto_n%C3%ADvel
https://pt.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina
https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_de_baixo_n%C3%ADvel
https://trailofbits.github.io/ctf/exploits/binary1.html
https://github.com/mentebinaria/fundamentos-engenharia-reversa/blob/master/introducao.md
https://null-byte.wonderhowto.com/how-to/exploit-development-everything-you-need-know-0167801/

FINISH

CURTA AS SEGUINTES PÁGINAS:

CYBER SECURITY UP
EXPERIENCE SECURITY
H4K SECURITY
CAVALEIROS DA INFORMÁTICA
aCCESS