ETHICAL HACKING EXPLOIT DEVELOPMENT (EHXD)

DESCRIÇÃO

(Nível Intermediário/Avançado)

Carga Horária: 40 hrs

O curso Ethical Hacking Exploit Development (**EHXD**) faz um estudo detalhado do processo de criação e desenvolvimento de *exploits* (*userland*) para aplicações no Sistema Operacional Windows, com foco na arquitetura x86. Ideal para os profissionais da área de segurança da informação que buscam novos desafios e procuram entender melhor como os ataques funcionam, como são encontradas falhas nos programas e sistemas, como são construídos *exploits* que exploram essas vulnerabilidades e como um atacante consegue contornar determinadas proteções.

O treinamento é 100% prático (*hands-on*) e serão exploradas aplicações reais em sistemas operacionais modernos por meio de uma bateria de exercícios e laboratórios. O aluno encontrará cenários dos mais simples aos mais complexos que vão desenvolver capacidades não somente da parte ofensiva (Pentesting, Red Team) como também da parte defensiva (SOC, CSIRT, Blue Team). Com a evolução constante das ameaças cibernéticas, é de fundamental importância conhecer e se manter atualizado sobre os principais ataques e como eles funcionam com o objetivo de proteger sua organização.

Durante o treinamentos serão abordados temas desafiadores, desde de *buffer overflow* clássico até como se dá o processo de *fuzzing* para encontrar pontos de falhas (*bugs*) em binários, criação de *exploits* que se aproveitam da Estrutura de Tratamento de Exceção do Windows (SEH *Overwrite*), análise de binário e injeção manual de código malicioso em aplicações legítimas (*backdooring*), processo de criação de *exploits* 0-Day (procedimentos e metodologia), *exploits* com a utilização de *Egghunter*, criação de *exploits* em ambientes restritos, criação e codificação manual de *shellcode*, introdução a técnica *Return Oriented Programming* (ROP) para contornar a proteção *Data Execution Prevention* (DEP), entre outros.

Para aqueles interessados em entender a funda a natureza dos *exploits* e querem dar um passo adiante, além de simplesmente utilizarem ferramentas automatizadas, esse é o treinamento perfeito, ministrado por profissionais capacitados, com didática sob medida e material de alta qualidade, onde cada tópico é descrito com riqueza de detalhes, permitindo que o aluno reproduza as atividades de forma fácil e intuitiva.

MÓDULO 0x00

1. Revisão de exploração de *Buffer Overflow* (BOF) clássico baseado na pilha (*vanilla*)
   - Análise de uma aplicação real
   - Utilização do Immunity Debugger com o plugin mona.py
   - Identificação do ponto de entrada vulnerável
   - Criação de uma Prova de Conceito (PoC) simples
   - Análise da corrupção de memória da aplicação via Immunity Debugger
   - Como sobrescrever o endereço de retorno de forma precisa via BOF
   - Controlando o EIP e o fluxo de execução da aplicação
   - Identificando endereços de memória e instruções de interesse
   - Localizando espaço para a inserção do *Shellcode*
   - Criação do *Shellcode* evitando badchars
   - Construção do *Exploit* e execução de Shellcode
2. Explorar a Estrutura de Tratamento de Exceção do Windows (SEH *Overwrite*)
   - Entender o mecanismo do *Structured Exception Handler* (SEH)
   - Como o Windows trata uma exceção de uma aplicação
   - Como se aproveitar do SEH no processo de criação de *exploits* mais estáveis
   - Sobrescrevendo a região da pilha e a cadeia do SEH
   - Desencadear uma exceção e sobrescrever, de forma precisa, a cadeia do SEH
   - Controlar o EIP via SEH *Overwrite* e dominar a execução da aplicação
   - Identificar endereços de memória e instruções de interesse (Assembly x86) que apontem para o *Shellcode* injetado – POP POP RET :)
   - Criação do *Shellcode* evitando badchars
   - Manipular a cadeia do SEH para executar o Shellcode
   - Construção do *Exploit* final e obtenção do *shell*
3. **Laboratório**

MÓDULO 0x01

1. Construção manual de um *Backdoor* em uma aplicação legítima
   - Análise de uma aplicação real
   - Utilização do Immunity Debugger com o plugin mona.py
   - Utilização do *debugger* x64dbg para análise do binário
   - Identificação de *Code Caves* no binário para inserção do código malicioso
   - Manipulação do *Entry Point* do binário
   - Inserção do *Shellcode* (*backdoor*)
   - Desvio do fluxo de execução normal do programa
   - Salto para o código do *backdoor*
   - Execução do código malicioso
2. Cifra manual do B*ackdoor* para evasão de Antivírus
   - Identificação de outros *Code Caves* no binário
   - Configuração de múltiplos desvios na execução do programa
   - Construção de código (Assembly x86) para cifrar o *Shellcode* (*backdoor*)
   - Criação do binário com *backdoor* cifrado em disco
   - Execução do *backdoor* em memória
3. **Laboratório**

MÓDULO 0x02

1. Criação de *exploits* utilizando a técnica de Egghunter
   - Análise de uma aplicação real
   - Utilização do Immunity Debugger com o plugin mona.py
   - Identificação do ponto de entrada vulnerável
   - Análise da corrupção de memória da aplicação via *debugger*
   - Localização de espaço de memória adequado para a inserção do *Shellcode*
   - Contornando limitação de instruções que podem ser utilizadas para manipular o binário
   - Utilização de saltos condicionais em Assembly x86
   - Identificação de locais alternativos para alojar *shellcodes* maiores
   - Como se aproveitar de *buffers* pequenos com a utilização do Egghunter
2. Emprego do Egghunter
   - Analise do código do Egghunter
   - Fundamentos de *System Calls* do Windows
   - Entendendo o mecanismo otimizado de procura em memória
   - Configurar o Egghunter para procurar pelo *Shellcode* final
   - Construção do *Exploit* e obtenção do *shell*
2. Aplicando o Egghunter em SO de 64 bits
   - Portabilidade para Windows 10
   - Adaptando o Egghunter para x64
3. **Laboratório**

MÓDULO 0x03

1. Metodologia de criação de Exploit 0-Day
   - Estudo da aplicação real a ser explorada
   - Análise do protocolo de rede utilizado pela aplicação
   - Identificação de pontos de falha na implementação do protocolo
   - Entender os mecanismos de *Fuzzing*
   - Aplicar técnicas de *Fuzzing* para identificar um ponto de entrada vulnerável
   - Configurar uma ferramenta de *Fuzzing* para corromper a aplicação
   - Análise do processo de corrupção de memória via *debugger*
   - Sobrescrever a cadeia do SEH da aplicação
2. Aproveitar a condição de SEH *Overwrite*
   - Sobrescrevendo a região da pilha e a cadeia do SEH
   - Acionar uma exceção e sobrescrever, de forma precisa, a cadeia do SEH
   - Escrita parcial do endereço de retorno
   - Controlar o EIP via SEH *Overwrite* e dominar a execução da aplicação
   - Identificar endereços de memória e instruções de interesse (Assembly x86)
   - Utilizar propriedades do protocolo de rede para facilitar a construção do *exploit*
   - Trabalhar com restrições de tamanho do *payload* injetado
   - Entender o mecanismo de saltos negativos em Assembly x86
   - Aplicar múltiplos saltos para se atingir o código desejado
   - Criação do *Shellcode* evitando badchars
   - Construção do *Exploit* final e obtenção do *shell*
3. **Laboratório**

MÓDULO 0x04

1. Criação de Exploits em ambientes restritos
   - Estudo de uma aplicação web real a ser explorada
   - Análise da interação entre cliente/servidor via protocolo HTTP
   - Aplicar técnicas de *Fuzzing* para identificar pontos de entrada vulneráveis no cabeçalho HTTP utilizado pela aplicação
   - Configurar uma ferramenta de *Fuzzing* para corromper a aplicação
   - Analisar o comportamento da aplicação, via *debugger* e via inspeção de tráfego de rede, durante a execução do processo de *Fuzzing* do cabeçalho HTTP
   - Identificação de vários pontos de falha da aplicação web
   - Determinação do ponto de falha mais adequado para ser explorado
2. SEH *Overwrite* com limitações
   - Sobrescrevendo a região da pilha e a cadeia do SEH
   - Ativar uma exceção e sobrescrever, de forma precisa, a cadeia do SEH
   - Controlar o EIP via SEH *Overwrite* e dominar a execução da aplicação
   - Identificar endereços de memória e instruções de interesse (Assembly x86)
   - Identificação de badchars para o cenário explorado
   - Determinação de um conjunto muito restrito de caracteres que podem ser utilizados
   - Mapeamento dos obstáculos a serem superados na criação do *Exploit*
   - Identificação de um endereço de memória adequado para controlar o EIP
   - Utilização de saltos condicionais de Assembly x86
   - Aplicar a técnica de execução do Shellcode por estágios/etapas
2. Egghunter codificado manualmente
   - Adotar a execução do *Shellcode* por fases via Egghunter
   - Identificar o local adequado para inserção do código do Egghunter e do *Shellcode*
   - Ajustes do P*ayload*, no cabeçalho HTTP, para disponibilizar mais espaço para o código do Egghunter e para o *Shellcode* final
   - Utilizar a técnica de expansão de bytes, durante a manipulação (*parse*) do cabeçalho HTTP pela aplicação, para prover mais espaços para o Egghunter Codificado
   - Manipular o registrador ESP para se aproveitar das instruções PUSH/POP (*stack pivot*)
   - Utilização de instruções aritméticas básicas e outras instruções simples de Assembly x86 para codificar o Egghunter de forma manual, evitando os badchars
   - Construção parcial do *Exploit* para testar a decodificação do Egghunter em memória
   - Ajuste de Pilha (*stack pivot*) para utilizar uma área de memória do binário com permissões adequadas
   - Criação do *Shellcode* para executar um *reverse shell*
   - Construção do *Exploit* final e obtenção do *shell*
3. **Laboratório**

MÓDULO 0x05

1. Proteções do Windows contra a utilização e execução de *exploits*
   - Mecanismos de Proteção (ASLR, DEP, SafeSEH, SEHOP, CFG, *Exploit Guard*)
   - Entender os efeitos do *Data Execution Prevention* (DEP)
   - Formas de contornar o DEP
   - *Return Oriented Programming* (ROP)
   - APIs do Windows que podem ser utilizadas para contornar o DEP
   - *VirtualAlloc, VirtualProtect, WriteProcessMemory*
   - Entender o que são Gadgets
   - Como encontrar Gadgets
   - Como encadear os Gadgets, de forma adequada, para realizar uma tarefa específica
2. Utilizar o ROP para contornar o DEP
   - Análise de uma aplicação real
   - Testar o funcionamento de um *Exploit* sem e com o DEP habilitado
   - Utilizar formas automatizadas de localização de Gadgets
   - Escolher a API do Windows mais adequada (*VirtualAlloc/VirtualProtect*)
   - Análise da cadeia de Gadgets (ROP Chain) gerada automaticamente
   - Identificar o endereço da API do Windows (*VirtualAlloc/VirtualProtect*)
   - Ajustar alguns Gadgets e Argumentos do ROP Chain
   - Criação do *Shellcode* para executar um *reverse shell*
   - Construção do *Exploit* final e obtenção do *shell*
3. **Laboratório**

MÓDULO 0x06

1. Desafios
2. Laboratórios Extras
3. Visão geral de temas mais avançados em Desenvolvimento de *Exploits*

PRÉ-REQUISITOS

- Conhecimentos básicos em Sistema Operacional Windows (linha de comando)
- Conhecimentos básicos em Sistema Operacional Linux (linha de comando)
- Conhecimentos básicos de Rede de Computadores, Serviços e Protocolos de Rede
- Conhecimento básico de programação/lógica de programação
- Conhecimento básico de linguagens de programação (preferencialmente, Python e C)
- Familiaridade com a distribuição Kali Linux
- Familiaridade com a ferramenta Metasploit
- Familiaridade com instruções básicas de Assembly x86
- Familiaridade com o Immunity Debugger
- Conhecimento de como funciona um Buffer Overflow baseado na pilha (*vanilla*)
- Familiaridade com ferramentas de Virtualização – VMWare e VirtualBox
- Recomendável ter realizado o curso Ethical Hacking Memory Corruption (**EHMEC**)

PÚBLICO ALVO

- Especialistas em Segurança da Informação
- Analista de Segurança da Informação
- Pentesters
- Membros de Red Team
- Desenvolvedores e Pesquisadores de *Exploits*
- Analista de Malwares
- Desenvolvedores de *softwares*
- Membros de CSIRT
- Pesquisadores da área de Segurança da Informação
- Profissionais de TI com interesse e afinidade na área de Segurança da Informação
- Entusiastas de Segurança da Informação

MATERIAL NECESSÁRIO

- Os alunos devem utilizar suas próprias estações de trabalho (Windows, Linux ou Mac OS), com a capacidade de executar até 02 (duas) Máquinas Virtuais (VM) simultaneamente.
- Configuração mínima de 8GB de RAM, 40 GB de espaço livre em disco, porta USB e placa de rede (RJ45 ou *wireless*) para acesso à Internet.
- Desejável 02 (dois) monitores para incremento na produtividade.
- Software de Virtualização: VMWare ou VirtualBox, preferencialmente, a versão mais atualizada.

MATERIAL RECEBIDO

- As Máquinas Virtuais com as aplicações utilizadas nos exercícios
- Slides do Curso no formato PDF
- Certificado de Conclusão do Curso no formato PDF (com a carga horária e ementa)

CAPACIDADES ALCANÇADAS

No final do curso, o aluno estará apto a:

- Utilizar *Debuggers* para análise de binários do Windows
- Dominar instruções da Linguagem Assembly x86 aplicadas a *Exploit Dev*
- Analisar binários do Windows em nível de linguagem Assembly x86
- Manipular e fazer *patch* em binários de Windows para inserção de *backdoor*
- Entender o processo de corrupção de memória em aplicações do Windows
- Entender e utilizar técnicas de *Fuzzing* para identificar pontos de falha em binários
- Analisar e identificar pontos de entrada vulneráveis em binários do Windows
- Compreender, escrever, manipular e utilizar *Shellcode*
- Compreender vulnerabilidades do tipo *Buffer Overflow*
- Entender a estrutura de tratamento de exceção do Windows (SEH)
- Compreender e explorar vulnerabilidades do tipo SEH *Overwrite*
- Entender o processo de criação de *exploits* 0-Day em binários do Windows
- Criar *exploits* para aplicações em Windows que utilizam a técnica de Egghunter
- Criar *exploits* em cenários restritos (*small buffers*, badchars, instruções limitadas)
- Entender os principais mecanismos de defesa contra *exploits* do Windows
- Entender e utilizar técnicas básicas de contorno de mitigações de segurança
- Compreender e utilizar a técnica *Return Oriented Programming* (ROP)

LAIOS FELIPE BARBOSA