Universidade Federal de Minas Gerais

Cibersegurança - Projeto de programação #2

Junho 2023

1 Introdução

O objetivo deste projeto é obter experiência prática na identificação de vulnerabilidades no código e na montagem de ataques de estouro de buffer (*buffer overflow*). Nas Partes 1 e 2, você recebe o código-fonte de cinco programas exploráveis que devem ser instalados com 'setuid root' em uma máquina virtual que fornecemos. A máquina virtual foi criada pela equipe da Universidade de Stanford, por isto você deve usar o login e a senha informados neste documento. Você terá que identificar uma vulnerabilidade (estouro de buffer, *double free*, vulnerabilidade de string de formato, etc.) em cada programa. Você escreverá um *exploit* para cada um que executa o programa vulnerável com argumento elaborado, fazendo com que ele salte para uma sequência de exploração. Em cada instância, o resultado produzirá um *shell root*, mesmo que o ataque tenha sido executado por um usuário sem privilégios. Na Parte 3, você usará um fuzzer para encontrar uma vulnerabilidade em um programa chamado bsdtar, parte de uma biblioteca amplamente usada chamada libarchive. Você fará o download e criará a versão vulnerável do libarchive em sua VM e, em seguida, executará o fuzzer para encontrar uma entrada que cause a falha do programa.

2 Ambiente

Você executará seus exploits em uma máquina virtual (VM) emulada usando QEMU. Isso serve a dois propósitos. Em primeiro lugar, os programas vulneráveis contêm vulnerabilidades reais e exploráveis e nós desaconselhamos instalá-los com 'setuid root' em sua máquina. Em segundo lugar, tudo, desde a versão específica do compilador até o sistema operacional e as versões da biblioteca instaladas, afetará a localização exata do código na pilha. A VM fornece um ambiente idêntico àquele em que a tarefa será testada para avaliação.

2.1 Iniciando a VM

Para iniciar sua VM, primeiro você precisará instalar o QEMU. Siga as instruções no site do QEMU aqui (`https://www.qemu.org/download/`). Os usuários

do MacOS precisarão primeiro instalar o Homebrew. Você pode encontrar instruções sobre como fazer isso aqui (`https://brew.sh/`).

Em seguida, você pode iniciar sua VM executando o script `run qemu.sh` incluído. Pode levar um ou dois minutos para que sua VM seja iniciada. Depois disso, você verá um prompt de login no terminal em que iniciou a VM. Você pode fazer login e usar sua VM a partir desse terminal. Sua VM possui um único usuário com o nome de usuário "user" e a senha "cs155". Como alternativa, você pode usar o ssh para fazer login na sua VM. Você pode usar o script ./ssh para qemu.sh incluído ou executar o comando

```
$ssh -p 5555 usuário@localhost
```

Você pode ter uma experiência de terminal mais suave usando sua VM sobre ssh porque o QEMU não usa toda a janela do terminal.

2.2 Usando a VM

A VM está configurada com Debian GNU/Linux 11 (alvo), com ASLR (randomização de endereço) desativada. Possui uma única conta de usuário "user" com senha "cs155", mas você pode se tornar temporariamente o usuário root usando o sudo. Os exploits serão executados como "usuário" e devem gerar um shell de linha de comando (/bin/sh) executado como "root". A VM vem com um conjunto de ferramentas pré-instaladas (curl, wget, openssh, gcc, vim etc), mas fique à vontade para instalar software adicional. Por exemplo, para instalar o editor emacs, você pode executar:

```
$ sudo apt-get install emacs
```

Quando você executar a VM pela primeira vez, ela terá um servidor OpenSSH em execução para que você possa fazer login em sua máquina host, bem como transferir arquivos usando, por exemplo, ssh, scp e sshfs. A VM já contém o código inicial no diretório proj1.

Para desligar sua VM, você pode executar:

```
$ sudo systemctl poweroff
```

3 Partes 1 e 2

As partes 1 e 2 pedem que você desenvolva exploits para cinco diferentes programas alvos vulneráveis.

3.1 Alvos

O diretório targets/ no tarball de atribuição (que já foi copiado para a VM para você) contém o código-fonte para os programas alvos vulneráveis, bem como um 'Makefile' para construí-los e instalá-los na VM. Especificamente, para instalar os programas alvos, como o "usuário" não root:

```
$ cd targets $
make $
sudo make install
```

Isso compilará todos os programas alvo, definirá o sinalizador de pilha executável em cada um dos executáveis resultantes e os instalará com 'setuid root' em `/tmp`. Seus exploits devem assumir que os programas alvo estão instalados em `/tmp/` como `/tmp/target1`, `/tmp/target2`, etc.

Observação: quando você reiniciar sua máquina, os arquivos no diretório /tmp serão excluídos automaticamente. Isso significa que você precisa executar sudo make install no diretório proj1/targets para reinstalar os programas alvo.

3.1.1 Explorando o código

O diretório `xploits/` no tarball contém um esqueleto de código para os exploits que você escreverá, chamados `xploit1.c`, `xploit2.c`, etc., para corresponder aos alvos. Também está incluído o arquivo de cabeçalho `shellcode.h`, que fornece um shellcode na variável estática

```
static const char* shellcode
```

3.1.2 Alvo 5

O quinto alvo tem sua pilha marcada como não executável. Isso significa que você não poderá pular para um shellcode que você colocou na memória. Você precisará usar Programação Orientada ao Retorno (ROP) para atacar este alvo. Para sua conveniência, você pode usar o script `./find gadgets.py` para localizar gadgets nesse destino. O script pesquisará um determinado executável e encontrará todos os gadgets em potencial nele. É seu trabalho usar esses gadgets para explorar o alvo 5.

4 Parte 3

Na Parte 3, você aprenderá como encontrar vulnerabilidades de segurança usando um fuzzer chamado American Fuzzy Lop. Fuzzing é uma técnica para encontrar vulnerabilidades em um programa, conforme vimos em aula, executando o programa com dados aleatórios até que ele falhe. Usaremos o afl-fuzz, um dos fuzzers mais bem-sucedidos e amplamente utilizados atualmente. afl-fuzz já foi

instalado na VM. Você pode ler mais sobre o aflfuzz e como ele funciona em `http://lcamtuf.coredump.cx/afl/`.

4.1 Fuzzing libarchive

Você investigará a libarchive (`https://www.libarchive.org/`), uma biblioteca de arquivo e compactação amplamente utilizada. Ela fornece um programa chamado bsdtar que oferece funcionalidade semelhante ao programa GNU tar. Por exemplo, você pode usar o bsdtar para extrair um arquivo `.tar.gz` da mesma forma que o tar normal:

```
$ bsdtar -xf <algum arquivo>.tar.gz
```

No diretório proj1/fuzz/, baixe e extraia o código-fonte para libarchive versão 3.1.2:

```
$ curl -O http://www.libarchive.org/downloads/libarchive-3.1.2.tar.gz
$ tar -xf libarchive-3.1.2.tar.gz
```

Isso deve fornecer um diretório chamado libarchive-3.1.2/. Nesse diretório, execute:

```
$ CC=afl-clang ./configure --prefix=$HOME/proj1/fuzz/install
```

Isso configurará o libarchive para que seja construído usando o compilador afl-fuzz, afl-gcc, e para que ele se instale no diretório `install/` em vez de em todo o sistema. Você pode então construir e instalar o libarchive, incluindo o bsdtar:

```
$ make
$ make install
```

(Observe que não estamos usando o sudo.) Depois disso, o bsdtar deve ser instalado em `install/bin/`. O diretório `fuzz/testcases/` contém um test-case semente que o `afl-fuzz` modificará para tentar travar o bsdtar. Execute `afl-fuzz` neste caso de teste executando o seguinte comando no diretório `fuzz/`

```
$ afl-fuzz -i testcases -o results install/bin/bsdtar -O -xf @@
```

Este comando instrui o afl-fuzz a executar bsdtar e fornecer os argumentos -O -xf @@. A opção -O (O maiúsculo, não numeral-0) diz ao bsdtar para não gravar nenhum arquivo no disco. afl-fuzz substituirá @@ pelo nome da entrada para testar uma falha, então a parte -xf @@ fará com que bsdtar tente extrair o arquivo de entrada gerado por afl-fuzz. O resultado é que o afl-fuzz gerará vários casos de teste com base nas sementes no diretório `testcases/` e, em seguida, executará o bsdtar em cada arquivo gerado até que o bsdtar falhe.

O fuzzer pode funcionar por vários minutos antes de encontrar uma falha. Depois disso, pressione Ctrl-C para parar o AFL.

5 Escrever

Passe alguns minutos investigando o acidente. Use o GDB para obter um backtrace no momento da falha. Tente descobrir qual é a vulnerabilidade no código-fonte. No arquivo `fuzz/README`, inclua seu backtrace do GDB e descreva brevemente a vulnerabilidade (duas ou três frases; não mais que 200 palavras).

6 Entregáveis

A tarefa é dividida em três partes:

- **Parte 1:** consiste nos alvos 1 e 2.
- **Parte 2:** consiste de investigar os outros três alvos.
- **Parte 3:** (que você enviará junto com a Parte 2) consiste em investigar vulnerabilidades em um programa do mundo real (bsdtar).

Para cada envio, você precisará fornecer um tarball gzipado (.tar.gz) gerado pela execução de `make submission` a partir do diretório de nível superior da origem da atribuição (proj1/). Este tarball conterá o conteúdo do diretório `sploits/`, as falhas encontradas durante o fuzzing e o README no diretório `fuzz/`. Certifique-se de que, se você extrair seu tarball de envio:

- No diretório extraído `xploits/`, executando o `make` sem argumentos, ele retorna os executáveis do `xploit1` a `xploit5` no mesmo diretório.
- No diretório extraído `fuzz/`, executando a versão vulnerável do bsdtar em qualquer um dos os casos de teste de travamento, deve-se reproduzir o travamento.
- No diretório `fuzz/` extraído, deve haver um arquivo README que descreva a vulnerabilidade habilidade encontrada via fuzzing.

OBSERVAÇÃO: seguir o formato de envio é importante. Ajude-nos!
Você precisa enviar o arquivo tarball pelo Moodle.

6.1 Dicas

- `gdb` é uma ferramenta poderosa para esta atividade.
- Se você rodar `gdb` e o alvo travar, você precisará reiniciar o `gdb` e rodar o `xploit` novamente.
- O comando `info frame` do `gdb` dará informações úteis sobre a pilha de execução do programa.

7 Configuração passo-a-passo

Baixe a VM a partir do link no Moodle e extraia. O tarball contém um arquivo `disk.img`, que é uma imagem de disco QEMU QCOW. Inicie a máquina usando o script de execução `qemu.sh` incluído. O script apenas invoca a seguinte linha

```
qemu-system-x86_64 disk.img -m 2G -nic user,hostfwd=tcp::5555-:22 -nographic
```

A opção -m 2G inicia a VM com 2 Gigabytes de memória. A opção -nic user,hostfwd=tcp::5555-:22 encaminha o tráfego da Internet na porta 5555 do seu computador para a porta 22 da VM. Isso permite que você faça ssh em sua VM. A opção -nographic faz com que o QEMU encaminhe a saída da sua VM para o seu terminal. Você pode iniciar sua VM sem essa opção. Se fizer isso, o QEMu criará uma nova janela para sua VM. Consulte a documentação do QEMU para obter mais informações sobre isso em `https://www.qemu.org/docs/master/`

Depois que a VM inicializar, faça o login com o nome de usuário "user" e a senha "cs155". Seu diretório pessoal agora conterá a pasta "proj1" que contém os alvos e o código inicial para o projeto. A VM deve ser configurada para encaminhar o tráfego na porta 5555 de sua máquina para a porta 22 da VM. Isso permite que você faça ssh em sua VM com o seguinte

```
$ ssh -p 5555 usuário@localhost
```

Uma vez logado, crie e instale os alvos:

```
$ cd proj1/targets
$ make && sudo make install
Senha: cs155
```

Escreva, construa e teste seus exploits:

```
$ cd ../xploits
 ...editar, testar...
$ make
$ ./xploit1
```

Se, a qualquer momento, você desejar uma nova cópia do código inicial, faça o download do tarball da tarefa no Moodle e extraia:

```
$ tar xzf proj1.tar.gz
```

Em seguida, repita as etapas de compilação e instalação acima.