HTTP-Dissector

Bruna Almeida Osti
Rafael Cortez Sanches
17 de Junho de 2020

Resumo

1 Introdução

O HTTP é um protocolo de transferência de hipertexto, ou seja, é um protocolo de comunicação da camada de aplicação utilizado para transferência de hipermídia (1). Esse protocolo funciona na forma de cliente-servidor pois as requisições são enviadas por um agente-usuário que pode ser um navegador Web, no qual cada requisição é enviada para um servidor que irá lidar com isso e retornar uma resposta (2).

O protocolo HTTP, além de não oferecer criptografia, também não garante que os dados não possam ser interceptados, coletados, modificados ou retransmitidos e nem que você esteja se comunicando exatamente com o site desejado. Por estas características, ele não é indicado para transmissões que envolvem informações sigilosas, como senhas, números de cartão de crédito e dados bancários, e deve ser substituído pelo HTTPS, que oferece conexões seguras (3).

Neste trabalho implementaremos um dissecador de pacotes, que funciona decodificando os pacotes transferidos durante sessões do protocolo HTTP e remontando-os, além disso, estudaremos um pouco a respeito dos ataques vinculados ao dissecador e modos de evitá-los.

2 Desenvolvimento

Primeiramente pesquisamos como funciona a requisição de pacotes no protocolo HTTP, como é o formato do header e o padrão do payload. Após o entendimento do funcionamento do protocolo, partimos em busca de alguma biblioteca que oferecesse suporte a leitura de pacotes do formato ".pcap", no qual foi escolhido a biblioteca "scapy" para o desenvolvimento do trabalho, pois a documentação desta biblioteca é bem completa, oferecendo suporte não só a leitura do arquivo como também sua manipulação integral (4).

O protocolo HTTP funciona enviando requisições e recebendo respostas do servidor, essas requisições podem ser métodos do tipo GET, POST, HEAD, PUT, TRACE, OPTIONS e DELETE (2). Cada uma desses tipos de requisições tem padrões de respostas, é importante o funcionamento de cada tipo de requisição como descrito na Tabela 1.

Método	Descrição	Corpo da mensagem
GET	Pegar um documento do servidor	Não
HEAD	Pegar só os headers do documento do servidor	Não
POST	Enviar dados para o servidor processar	Sim
PUT	Guardar no corpo a requisição do servidor	Sim
TRACE	Seguir a messagem através dos proxys dos servidores	Não
OPTIONS	Determinar quais métodos podem operar em um servidor	Não
DELETE	Remover um documento do servidor	Não

Tabela 1: Tipos de requisições (2)

O padrão seguido para mensagens enviadas seguem a seguinte estrutura mostrada abaixo, no qual o método se refere ao tipo de requisição feita ao servidor, a versão se refere à versão do protocolo. Por outro lado, na resposta a versão do protocolo vem primeiro e em seguida o status do requerimento, se retornou com erros ou foi bem sucedido e a frase resposta. Ambos apresentam headers, mas o corpo da mensagem é dispensável em alguns casos, como visto

na Tabela 1.

Requisição:

```
<método> <URL requerida> <versão>
<headers>
<corpo da mensagem>
```

Resposta:

```
<versão> <status> <frase de resposta>
<headers>
<corpo da mensagem>
```

Para o desenvolvimento inicial foi utilizado o arquivo ".pcap" disponível pelo professor, no qual verificamos que as requisições aconteciam em sessões diferentes, portanto teríamos que verificar em cada sessão se a camada HTTP estava disponível.

Posteriormente selecionamos apenas o header HTTP utilizando expressões regulares para pegar o padrão das requisições, se houver alguma requisição do tipo GET ou POST, extraímos o payload e remontamos os pacotes utilizando a biblioteca "zlib" para descompressão.

3 Testes

PCAP é uma interface de aplicação (API) para captura de tráfego escrita em C. Existem vários *wrappers* dessa biblioteca para outras linguagens, bem como programas que a utilizam, como Wireshark e tcpdump. PCAP também é o nome do padrão para arquivos de captura de tráfego gerados por essa biblioteca.

Essa seção apresenta métodos utilizados para gerar um arquivo PCAP, o qual é consumido pela aplicação HTTP Dissector. Também são expostos detalhes sobre os arquivos extraídos de um dado PCAP previamente fornecido.

3.1 Geração de arquivo de tráfego PCAP

Em um sistema Linux, arquivos PCAP podem ser gerados por meio da aplicação *tcpdump*.

```
# tcpdump –w test.pcap tcp port 80
```

Uma vez que o acesso à interface de rede requer privilégios de superusuário, o comando acima deve ser executado com *sudo* caso o usuário não possua privilégios.

No exemplo acima, assume-se que os servidores Web envolvidos nas transações estão escutando a porta TCP padrão para o protocolo HTTP (80). Caso isso não seja verdade, o parâmetro deve ser ajustado de acordo.

Uma vez que o comando *tcpdump* estiver executando, deve-se navegar por sites que não utilizam SSL (HTTPS) para que pacotes HTTP possam ser capturados a partir da interface de rede padrão. A captura pode ser encerrada por um sinal SIGINT (Ctrl + C) no terminal. Os pacotes capturados estarão todos no arquivo "test.pcap".

3.2 Arquivo PCAP fornecido

Para testar a capacidade de extrair binários executáveis do Windows a partir de pacotes HTTP, foi utilizado o arquivo PCAP fornecido para o experimento anterior dessa disciplina (*Network Traffic Analysis*). Ele contém trafego capturado durante um ataque de cavalo de tróia, no qual são obtidos dois executáveis maliciosos do Windows por meio do protocolo HTTP.

3.3 Artefatos obtidos

O Dissector não obtém somente arquivos baixados via HTTP, mas o conteúdo de todas as mensagens HTTP trocadas, incluindo as mensagens POST. A tabela 2 relaciona todos os conteúdos obtidos a partir do PCAP analisado.

Os binários maliciosos aparecem nas mensagens HTTP como sendo uma imagem JPEG (5) e uma planilha do Microsoft Excel (2), mas rodando o comando *file* do Linux é possível perceber que eles na verdade são executáveis PE32 do Windows.

ID	Tipo	Comentário
0	HTML	GET com o texto ”[SelecaoBancoErro]”
1	WWW-FORM-URLENCODED	Dados pessoais da vítima
2	MS-EXCEL	Binário malicioso PE32
3	WWW-FORM-URLENCODED	Dados pessoais da vítima
4	HTML	GET com o texto ”[SelecaoBancoErro]”
5	JPEG	Binário malicioso PE32
6	PLAIN	Vários códigos (hash?) separados em linhas
7	PLAIN	Vários códigos (hash?) separados em linhas
8	WWW-FORM-URLENCODED	Dados pessoais da vítima

Tabela 2: Arquivos extraídos

Os binários extraídos foram submetidos ao site ”VirusTotal” (5) para checagem, o qual confirmou a suspeita de que esses eram executáveis maliciosos, conforme visto no trabalho anterior da disciplina.

4 Ataques vinculados

Nesta seção iremos abordar um pouco a respeito dos tipos de ataques realizados ao protocolo HTTP e algumas contramedidas para evitar esses tipos de ataques.

4.1 Tipos de ataques

4.1.1 Spoofing

O Spoofing é uma técnica de disfarce/falsificação da comunicação de uma fonte desconhecida como sendo segura e confiável. Esse tipo de ataque redireciona o tráfego da internet para sites maliciosos que visam roubar informações ou distribuir malwares (6).

A falta de criptografia no HTTP permite que um servidor Web seja personificado mais facilmente, uma vez que não há checagem de autenticidade em nenhuma das partes envolvidas na comunicação cliente/servidor.

4.1.2 Sniffing

Esse tipo de ataque corresponde ao roubo/interceptação de dados quando estão sendo transmitidos pela rede sem criptografia, um invasor pode analisar a rede e obter informações para eventualmente causar o travamento ou a corrupção da rede ou ler as comunicações que estão trafegando na rede (7).

4.2 Contramedidas

4.2.1 HTTPS

Uma medida muito utilizada atualmente para proteção dos dados é o protocolo HTTPS, que é uma extensão segura do HTTP com certificado TLS *(Transport layer Security).* O objetivo do TLS é tornar segura a transmissão de informações sensíveis com dados pessoais, é uma alternativa à transferência de dados na qual a conexão ao servidor não é criptografada pois torna mais difícil a interceptação desses dados (8).

O TLS utiliza tanto criptografia assimétrica para assegurar a autenticidade, quanto a simétrica para a confidencialidade e integridade (8). O TLS é a única e eficaz maneira de obter segurança de dados em comércio eletrônico, quando há um Certificado Digital instalado no web site, um ícone de um cadeado aparece no navegador e o endereço começa com https:// ao invés de http:// informando que os dados serão criptografados (9).

Quando o navegador se conecta a um servidor HTTPS, o servidor responde com seu certificado. O navegador verifica se o certificado é válido, portanto checa se as informações do proprietário correspondem ao nome do servidor que o usuário solicitou, se o certificado é assinado por uma autoridade de certificação confiável. Entretanto, se alguma dessas condições não for atendida, o usuário será informado sobre o problema (8).

Quando o HTTP é usado, ocorre as três vias de handshakes de mesmo modo do protocolo TCP, após isso a comunicação fica criptografada para trocar chaves de criptografia, a comunicação então pode prosseguir, pois essas etapas iniciais acontecem em questão de milissegundos (10).

Depois que o handshake HTTPS é concluído, todas as comunicações entre o cliente e o servidor são criptografadas. Isso inclui o URL completo, dados (texto sem formatação ou binário), cookies e outros cabeçalhos (10).

4.2.2 VPN

A VPN (Virtual Private Network) fornece aos usuários uma rede virtual privada para que possam se conectar de forma segura na internet. A essência da VPN é manter suas informações em sigilo. Seu funcionamento é dado pelo roteamento da conexão à internet do seu dispositivo através do servidor privado da VPN escolhido em vez do provedor de serviços (ISP), para que quando os dados forem transmitidos venham da VPN e não do seu computador (11).

A VPN atua como uma espécie de intermediária à medida que você se conecta à Internet, ocultando seu endereço IP e protegendo sua identidade. Além disso, se seus dados forem interceptados de alguma forma, eles serão ilegíveis até chegarem ao seu destino final, pois além de servir como intermédia também criptografa os dados para serem transmitidos (11).

A conexão sem utilizar uma VPN expõe os usuários a uma infinidade de vulnerabilidades, pois quando conectado a uma rede local o ISP consegue manter registrado tudo que o usuário está fazendo, enquanto que quando conectado em pontos de acessos não confiáveis os usuários podem acabar acessando pontos de acessos falsos que foram criados por pessoas não tão bem intencionadas (11).

5 Considerações Finais

Com esse experimento é possível perceber a fragilidade do protocolo HTTP para troca de informações confidenciais. Um atacante que tomar controle de um roteador de borda de um *Autonomous System* (AS) pode ter acesso a uma quantidade irrestrita de tráfego e obter o conteúdo das mensagens por meio de um *dissector*, como foi feito nessa prática. Isso também pode ser feito para outros protocolos que não utilizam criptografia para garantir confidencialidade, como o antigo *Remote Shell* (RSH).

Por esse motivo, protocolos desse tipo foram substituídos ao longo do tempo por novas versões, as quais utilizam criptografia para garantir confidencialidade e autenticidade. Nesse contexto, o HTTPS substituiu o HTTP e o SSH substituiu o RSH. Uma vez que as ferramentas de análise de tráfego estão cada vez mais rápidas devido à evolução do hardware, qualquer pacote não criptografado está sujeito a ser interceptado e processado. O ideal é que protocolos com criptografia sejam adotados sempre que possível em aplicações que utilizem a Internet.

Referências

[1] M. web docs, "Uma visão geral do http," jun 2020.

[2] D. Gourley, *HTTP : the definitive guide*. Beijing Sebastopol, CA: O'Reilly, 2002.

[3] Cert.br, "Cartilha de segurança para internet," jun 2020.

[4] P. Biondi and the Scapy community, "Welcome to scapy's documentation!," jun 2020.

[5] VirusTotal, “Virustotal homepage,” jun 2020.

[6] ForcePointing, “O que é spoofing?,” jun 2020.

[7] HTTPDebugger, “Analisador de http e analisador de protocolo,” jun 2020.

[8] D. Silva, “Segurança: Entenda como funciona o protocolo https,” jun 2020.

[9] HostingerTutoriais, “O que é ssl / tls e https?,” jun 2020.

[10] LOVE2DEV, “Como o https funciona para mantê-lo seguro e como difere do http,” jun 2020.

[11] NameCheap, “How does vpn work?,” jun 2020.