Universidade Federal de Pernambuco

Centro de Informática

Bacharelado em Sistemas de Informação

Engenharia reversa em firmwares de roteadores SOHO

Patrick Roberto Braz Costa

Trabalho de Graduação

Recife

6 de dezembro de 2018

Universidade Federal de Pernambuco

Centro de Informática

Patrick Roberto Braz Costa

Engenharia reversa em firmwares de roteadores SOHO

Trabalho apresentado ao Programa de Bacharelado em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: *Fernando José Castor de Lima Filho*

Recife

6 de dezembro de 2018

dedico este trabalho aos meus pais

Agradecimentos

Agradeço a todas pessoas que participaram da minha experiência na graduação, companheiros de classe, professores e em especial ao meu orientador. Aos meus colegas de trabalho que tanto contribuíram para o meu crescimento profissional, em particular ao cara que me deu a pílula vermelha, me fazendo mergulhar de cabeça no submundo da computação. Agradeço especialmente a minha namorada que me ajudou não só em todos os dias deste trabalho, mas desde que nos conhecemos. Por fim, gostaria de agradecer principalmente aos meus pais que sempre me proveram com mais do que o necessário para se viver, estiveram do meu lado nos momentos bons e ruins, e me apoiaram sempre que quis me aventurar.

Yes, I am a criminal. My crime is that of curiosity.

—THE MENTOR (The Hacker Manifest)

Resumo

Roteadores SOHO são a porta de entrada para a rede doméstica e por isso são frequentemente alvos de ataque. Infelizmente, os fabricantes além de não desenvolverem *firmwares* seguros, não costumam prover correções para as vulnerabilidades conhecidas. Ademais, o código fonte dos *firmwares* não é disponibilizado publicamente, o que torna a descoberta e a correção das falhas uma tarefa complicada para a comunidade. Consequentemente, a análise de segurança em *firmwares* de roteadores é uma tarefa desafiadora, exigindo uma engenharia reversa do dispositivo. Este trabalho contém uma revisão bibliográfica das técnicas para engenharia reversa em roteadores SOHO, aplicando-as em um dispositivo, no intuito de descobrir vulnerabilidades.

Palavras-chave: engenharia reversa, roteador, firmware, vulnerabilidade

Abstract

SOHO routers are the entry point in home networks and, therefore, are targets of attacks from the Internet. Unfortunately, vendors do not develop secure firmwares and neither provide patches for known vulnerabilities. Besides that, the source code of firmwares are usually not publicly released, what makes it hard for the comunity to audit them. Consequently, the security analysis on these devices is challenging, requiring a reverse engineering. This work contains a bibliographic review of the techniques for reverse engineering SOHO routers and shows how to apply them in a device, in order to find vulnerabilities.

Keywords: reverse engineering, router, firmware, vulnerability

Sumário

Lista de Figuras

CAPÍTULO 1

Introdução

Segundo a UNCTAD (2017), o Brasil é o quarto país do mundo em número de usuários da Internet, com cerca de 120 milhões de pessoas conectadas, ficando apenas atrás de Estados Unidos, Índia e China. Apesar dos benefícios da inclusão digital, esta realidade traz novos riscos, uma vez que dados pessoais trafegam cada vez mais nas redes de computadores.

Encontrar roteadores com vulnerabilidades conhecidas na Internet não é uma tarefa onerosa. Poornachandran et al. (2015) escanearam a Internet, em busca de dispositivos vulneráveis, e encontraram diversos roteadores utilizando servidores DNS maliciosos. Serviços como o Shodan[1], fornecem facilmente informações sobre dispositivos acessíveis publicamente na rede mundial de computadores. Em uma análise de larga escala, Costin et al. (2014) descobriu diversos dispositivos com diferentes tipos de *backdoors*. Algumas vezes na forma de credenciais *telnet* fixadas, outras vezes ativado pelo fornecimento de uma cadeia de caracteres "mágica"em uma requisição ou pacote UDP.

Valendo-se de vulnerabilidades conhecidas, alguns *malwares* ganharam notoriedade pela quantidade de infecções em roteadores SOHO. Um deles é o VPNFilter, responsável por infectar mais de 500 mil roteadores ao redor do mundo (GOODIN et al., 2018). Bohio (2015), além de revisar diversos *malwares* da história que atacaram roteadores, analisou um *malware* específico para a plataforma MIPS. Com o roteador sob controle, é possível praticar diversas formas de ataques contra os usuários da rede ou simplesmente utilizar-se do *link* de Internet para a prática de atividades ilícitas. Tais condutas podem gerar consequências ao proprietário, como o caso de um cidadão alemão que foi multado por deixar sua Wi-fi desprotegida (BBC, 2010).

Como constatado por Costin e Zaddach (2013) é relativamente fácil encontrar vulnerabilidades em dispositivos embarcados, uma vez que os fabricantes focam mais em segurança

[1] https://www.shodan.io/

por obscuridade do que em profundidade. Muitas vulnerabilidades podem ser encontradas simplesmente utilizando ferramentas e *frameworks* existentes. Tendo em vista o cenário preocupante em que se encontra a segurança dos roteadores, é preciso que as vulnerabilidades em questão sejam publicadas e corrigidas, de forma a impulsionar uma melhora nos produtos existentes e no processo de produção dos fabricantes.

Infelizmente, o código fonte dos *firmwares* de roteadores não é publicamente disponibilizado na Internet e em alguns casos, nem mesmo o arquivo binário é fornecido. Portanto, para analisar o *firmware* em busca de vulnerabilidades, geralmente é necessário a engenharia reversa do dispositivo. Para isso, existem diversas técnicas catalogadas que têm como objetivo extrair informações existentes em um dispositivo e entender sobre seu comportamento.

Este trabalho traz uma abordagem empírica, desenvolvida através de uma revisão bibliográfica e um experimento, para realização da engenharia reversa do *firmware* de um roteador. O estudo utiliza como exemplo o repetidor de sinal *wireless* NPLUG[2], ofertado pela empresa brasileira Intelbras.

A escolha deste dispositivo deve-se primeiro ao fato do mesmo ter sido adquirido previamente pelo autor. No entanto, para realização da análise descrita neste trabalho, o acesso físico ao dispositivo seria perfeitamente dispensável. Além disso, este é um dispositivo que funciona sob princípios muito similares ao de um roteador, porém, apresentando uma complexidade menor, o que facilita sua utilização como prova de conceito. Por último, este é um dispositivo de fácil acesso para o usuário brasileiro, sendo encontrado facilmente em várias lojas do país.

1.1 Justificativa e objetivo

A segurança dos roteadores é de grande importância para os usuários da Internet no mundo todo. Uma falha de segurança em um roteador pode levar usuários maliciosos a obterem acesso à rede interna, contendo diversos outros dispositivos tais como celulares, câmeras, *tablets* e computadores. Portanto, é de interesse comum que as vulnerabilidades sejam descobertas cor-

[2]Intelbras NPLUG http://www.intelbras.com.br/empresarial/wi-fi/para-sua-casa/repetidor-de-sinal/nplug

rigidas pelo fabricante.

Alguns estudos foram publicados descrevendo técnicas específicas para se analisar roteadores em busca de vulnerabilidades, porém a literatura em português mostrou-se escassa. Este trabalho sumariza as estratégias utilizadas, de modo a introduzir o assunto em nossa literatura.

Ademais, o objetivo deste trabalho é investigar as várias técnicas utilizadas para realizar a engenharia reversa em *firmwares* de roteadores, no intuito de desenvolver uma abordagem que possa ser utilizada em qualquer dispositivo da mesma classe, para a descoberta de vulnerabilidades.

1.2 Conceitos

1.2.1 Firmware

De acordo com o trabalho de Costin e Zaddach (2013), o termo *firmware* foi cunhado por Ascher Opler, em uma publicação da revista *Datamation* de 1967, originalmente utilizado para referir-se ao microcódigo de CPU existente entre o *hardware* e software. Nos dias atuais, este termo é utilizado de forma mais abrangente, como por exemplo, para referir-se à BIOS, carregadores de sistemas operacionais (*bootloaders*) e sistemas de controle de embarcados.

Neste trabalho, qualquer menção ao termo *firmware* será referida ao *software* responsável por configurar e controlar roteadores. Este programa pode existir em formatos não-documentados, sem qualquer informação sobre como inicializá-lo. Costin e Zaddach (2013) classificam a complexidade de um *firmware* em **3** (três) categorias:

- Completo
  - Tipicamente um *firmware* Linux ou Windows que carrega um sistema operacional completo. A aplicação irá, geralmente, rodar em modo usuário, entretanto, módulos de *kernel* customizados podem ser utilizados.
- Integrado

- Sistema operacional inexistente ou proprietário. A aplicação geralmente roda com privilégios de *kernel*.

■ Atualizações parciais

- A imagem do *firmware* não contém todos os arquivos necessários para o funcionamento, somente uma atualização dos arquivos de interesse.

1.2.2 Roteadores SOHO e suas vulnerabilidades

Os roteadores caseiros, popularmente chamados de roteadores SOHO (*small office/home office*), são dispositivos focados em rotear o tráfego de rede, em uma casa ou pequena empresa. São conhecidos por disponibilizarem interfaces web para seu gerenciamento, além de funcionalidades extras como *firewall*, VPN e QoS. Estudos comprovam que esses dispositivos, apesar de fornecerem funcionalidades convenientes, costumam pecar no quesito segurança. Szewczyk e Macdonald (2017) destacaram os riscos que os provedores de Internet banda-larga expõem seus usuários na Internet.

Devido a limitações de hardware, o código escrito para *firmwares* de roteadores geralmente utiliza linguagens com baixo nível de abstração, como C e Assembly, sem o uso de *frameworks*. Portanto, é geralmente mais suscetível a falhas de programação que culminam no surgimento de vulnerabilidades. Gourdin et al. (2011) destacou os desafios de criar interfaces web seguras para dispositivos embarcados e propôs um *framework* específico para esta finalidade.

Além disso, não é incomum encontrar notícias sobre dispositivos contendo trechos de código, intencionalmente mal escritos, com intuito malicioso, também conhecidos como *backdoors*. Um caso famoso de *backdoor* foi publicado por Heffner (2013), onde um roteador popularmente utilizado na época permitia, a qualquer requisição contendo a *string* **xmlset_roodkcableoj28840ybtide** no campo *User-agent*, acessar as funcionalidades de sua área autenticada.

1.2.3 Engenharia Reversa

Segundo Eilam (2005), o conceito de Engenharia Reversa surgiu bem antes da tecnologia que conhecemos hoje e consiste no processo de extrair conhecimento de algo criado pelo ser humano. Geralmente, esta atividade é exercida com o objetivo de adquirir conhecimento sobre o funcionamento de um artefato quando esta informação está inacessível. Tratando-se de tecnologia, a engenharia reversa pode ser feita nas mais diversas camadas existentes. Em um *hardware*, esta poderia ser feita por exemplo, tentando de alguma forma interceptar os sinais lógicos de um circuito, extraindo dados de componentes, ou conectando-se a uma interface serial. Em um *software*, a engenharia reversa poderia ser feita por meio de técnicas como descompressão, desofuscação e descompilação de um programa. O objetivo principal é proporcionar o entendimento do *software*, mesmo quando não se tem acesso ao código fonte original.

A engenharia reversa em roteadores SOHO, possui um papel importante na descoberta de vulnerabilidades. Estes dispositivos são classificados como COTS (*commercial-off-the-shelf*), ou seja, produzidos sem customizações, visando a redução de custos, diretamente para o consumidor final. Este tipo de produto tende a ficar obsoleto e sem suporte, levando a existência falhas não corrigidas que ficam expostas à Internet. Como destacado por Papp, Ma e Buttyan (2015), a segurança e estabilidade dos dispositivos de rede SOHO são sujeitos à confiabilidade e qualidade de seus *firmwares*.

1.3 Estrutura do documento

Este trabalho está estruturado da seguinte forma. O Capítulo **2** descreverá a metodologia utilizada para a revisão bibliográfica, incluindo os mecanismos de busca e critérios de seleção. O Capítulo **3** irá conter todo o processo utilizado para realização da engenharia reversa no *firmware* de um roteador. Por fim, o Capítulo **4** apresenta uma conclusão do trabalho, bem como possíveis direções para trabalhos futuros. Os trabalhos relacionados serão mencionados ao longo deste documento, não havendo portanto uma área específica para eles. Não obstante, é possível encontrar uma lista com todos os artigos utilizados na pesquisa, ordenados cronologicamente por data de publicação no Apêndice **A**.

CAPÍTULO 2

Metodologia

O desenvolvimento da metodologia de engenharia reversa em roteadores foi feito através de uma revisão bibliográfica, no intuito de enumerar as principais técnicas e ferramentas utilizadas até o momento, além das dificuldades encontradas no processo. Foram definidos critérios de inclusão e exclusão para filtrar os estudos encontrados através de mecanismos de busca.

- Critérios de inclusão:
  - Conter informações sobre a engenharia reversa de *firmwares*;
  - Escrito em inglês.
- Critérios de exclusão:
  - Documento inacessível;
  - Documento com mais de 20 páginas.

Primeiramente, dois engenhos de busca acadêmicos foram utilizados para encontrar artigos relacionados com o tema. A fim de limitar a quantidade de artigos para serem revisados, foi definida uma *string* de busca que pudesse englobar o maior número de artigos relacionados ao tema, descartando falso positivos.

Engenho de busca	**URL**
ACM Digital Library	https://dl.acm.org/
IEEE Xplore Digital Library	https://ieeexplore.ieee.org/

Uma vez que estamos tratando de *firmwares* de roteadores, os dois termos foram dados como obrigatórios. A inclusão do termo "*embedded*" no lugar de "*routers*", apesar de retornar

artigos relacionados, trouxe também muitos artigos com foco distinto do proposto neste trabalho, e portanto, não foi utilizada. No entanto, artigos que tratem de *firmwares* de embarcados não foram completamente descartados, tendo em vista que poderiam ser incluídos na etapa seguinte. O termo "*vulnerability*" foi utilizado para trazer a busca ao contexto de segurança, evitando assim diversos artigos que possam tratar apenas de desenvolvimento e performance. O termo "*reversing*" não foi utilizado, pois restringia artigos de interesse desse estudo. Como resultado, o número de artigos encontrados e as *strings* utilizadas estão descritos na tabela a seguir:

Engenho de busca	**Termo de busca utilizado**	**Resultados**
ACM Digital Library	("firmware") +("router") +("vulnerability")	82
IEEE Xplore Digital Library	(((firmware) AND router) AND vulnerability)	7

O título e resumo dos resultados da busca foram analisados de acordo com os critérios de inclusão e exclusão. Somente **11** (onze) artigos encontrados nos engenhos de busca acima foram mantidos na revisão.

As referências dos trabalhos selecionados anteriormente foram utilizadas para buscar novos artigos. Estas buscas foram feitas com o auxílio da ferramenta Google Scholar[1]. Os novos trabalhos acadêmicos encontrados tiveram o título e resumo lidos, filtrando-os de acordo com os critérios definidos. Como resultado, foram adicionados **23** (vinte e três) novos artigos à pesquisa, totalizando **34** (trinta e quatro) trabalhos acadêmicos.

Além de artigos acadêmicos, a literatura cinzenta foi considerada pela quantidade de material relacionado disponível informalmente na Internet. Os artigos selecionados foram lidos por completo e estão listados no Apêndice A.

[1]Google Scholar https://scholar.google.com.br/

CAPÍTULO 3

A engenharia reversa do firmware de um roteador

Neste capítulo será destrinchado o processo de engenharia reversa de um roteador com o objetivo de descobrir vulnerabilidades. A ordem das etapas descritas a seguir foi escolhida arbitrariamente pelo autor, de forma que o conhecimento sobre o dispositivo será construído gradativamente.

3.1 Análise inicial

Independente da posse do *firmware* de um roteador é possível analisar o dispositivo de forma superficial em busca de vulnerabilidades. Este processo é feito através do estudo da documentação, análise da interface web e escaneamento de portas.

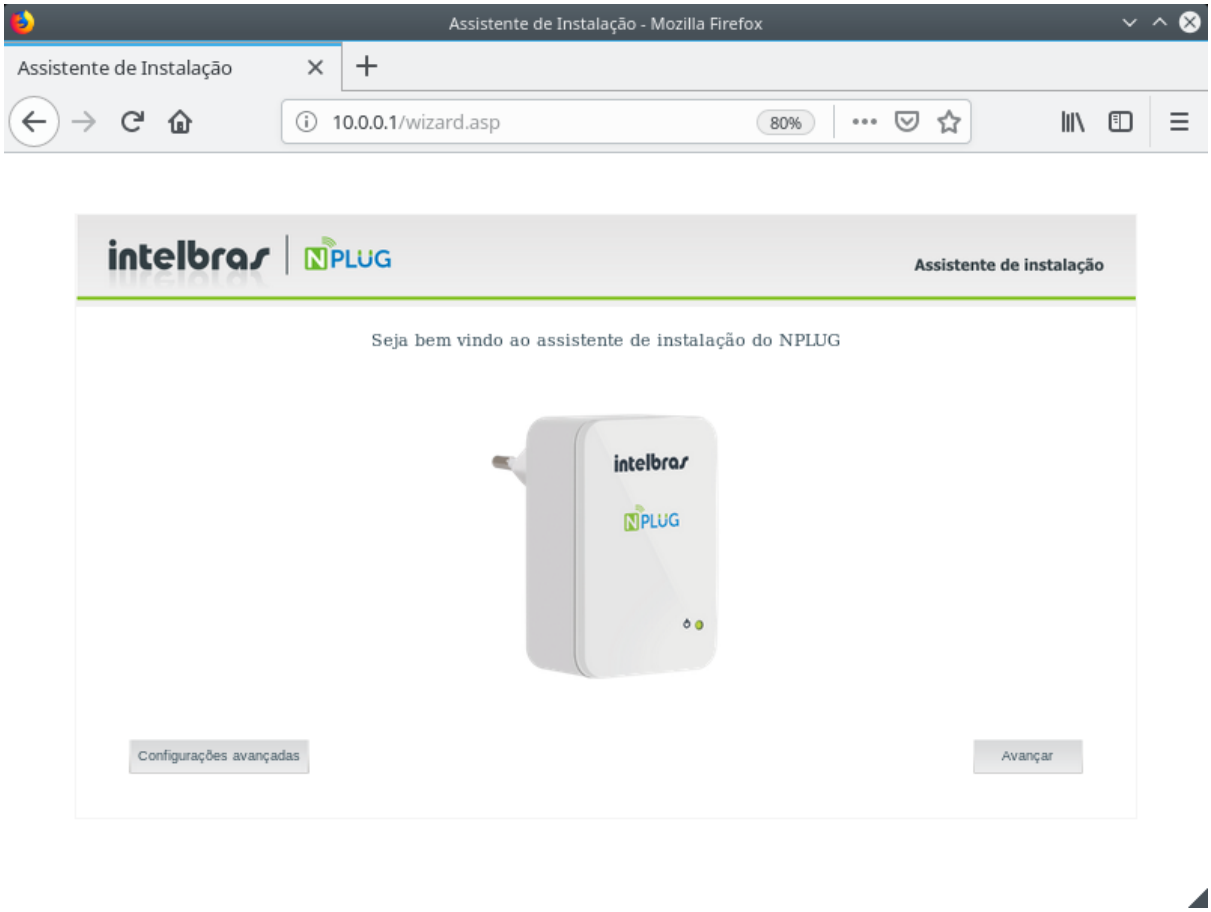
3.1.1 Estudando a documentação

Para agilizar o entendimento do objeto de estudo, o primeiro passo para a engenharia reversa de um roteador é procurar informações sobre o dispositivo, seja no site do fabricante ou através de engenhos de busca. A documentação de um roteador, embora geralmente seja deficiente (GHITA; FURNELL, 2018), pode ser uma fonte rápida de informações a respeito das configurações vindas de fábrica, tais como senhas padrão e serviços existentes no dispositivo. De acordo com a documentação do NPLUG, o dispositivo não possui nenhum tipo de autenticação por padrão. Sendo necessário configurar uma senha manualmente pela interface web para se obter o mínimo de segurança.

Uma forma de se obter informações sobre a estrutura interna do dispositivo sem possuir ou abrir o dispositivo, é utilizar o número identificador do produto em sites como o FCC ID

Search[1]. Dispositivos homologados pelo governo americano possuem o chamado FCC ID, no Brasil temos como equivalente o número de certificação da Anatel. Com essa identificação, muitas vezes, é possível encontrar manuais, especificações e até fotos internas do produto.

Figura 3.1: Tela inicial de configuração do dispositivo.
FONTE: O autor (2018)



3.1.2 Análise de caixa-preta

Após adquirir um contexto geral do dispositivo, é comum partir para a chamada análise de caixa-preta. Esta etapa consiste em investigar o roteador sem conhecimento do seu funcionamento interno. O primeiro passo é conectar o dispositivo ao computador utilizando um cabo

[1]FCC ID Search https://fccid.io/

Ethernet, iniciar uma ferramenta para análise de tráfego, como o tcpdump[2] (linha de comando) ou o Wireshark[3] (interface gráfica) e ligar o roteador. Analisar o tráfego de rede neste instante, permite observar os pacotes que são transmitidos pelo dispositivo no momento em que é iniciado. Caso o dispositivo disponibilize algum acesso remoto, recomenda-se utilizar a linha de comando para executar uma análise mais aprofundada, como descrito posteriormente. Em alguns dispositivos, o acesso via linha de comando não é documentado, portanto, recomenda-se escanear todas as portas do dispositivo em busca de serviços TCP ou UDP utilizando a ferramenta Nmap[4]. A ferramenta Nmap foi utilizada para escanear o NPLUG, no entanto não foram encontrados serviços expostos além de um servidor HTTP.

Os roteadores SOHO são conhecidos por disponibilizarem uma interface web para facilitar a configuração por parte do usuário. Nesta interface é possível habilitar as funcionalidades que podem não vir ativadas por padrão, por exemplo, o acesso via linha de comando. Muitas das vulnerabilidades existentes em roteadores podem ser encontradas realizando uma análise da aplicação web disponibilizada pelo dispositivo. Existe uma extensa gama de vulnerabilidades web relatadas em roteadores tais como *cross-site scripting* (XSS), *cross-site request forgery* (CSRF), deficiências no mecanismo de autenticação e até execução remota de comandos (RCE). Bojinov, Bursztein e Boneh (2009) apresentaram uma classe de XSS onde o código malicioso não é inserido através da web, mas por meio de um canal não-convencional, tal como NFS ou SNMP. Este tipo de ataque passou a ser chamado de *cross channel scripting* (XCS). Bojinov et al. (2009) avaliaram 21 sistemas embarcados em busca de vulnerabilidades em interfaces web, reportando mais de 40 vulnerabilidades para o CERT.

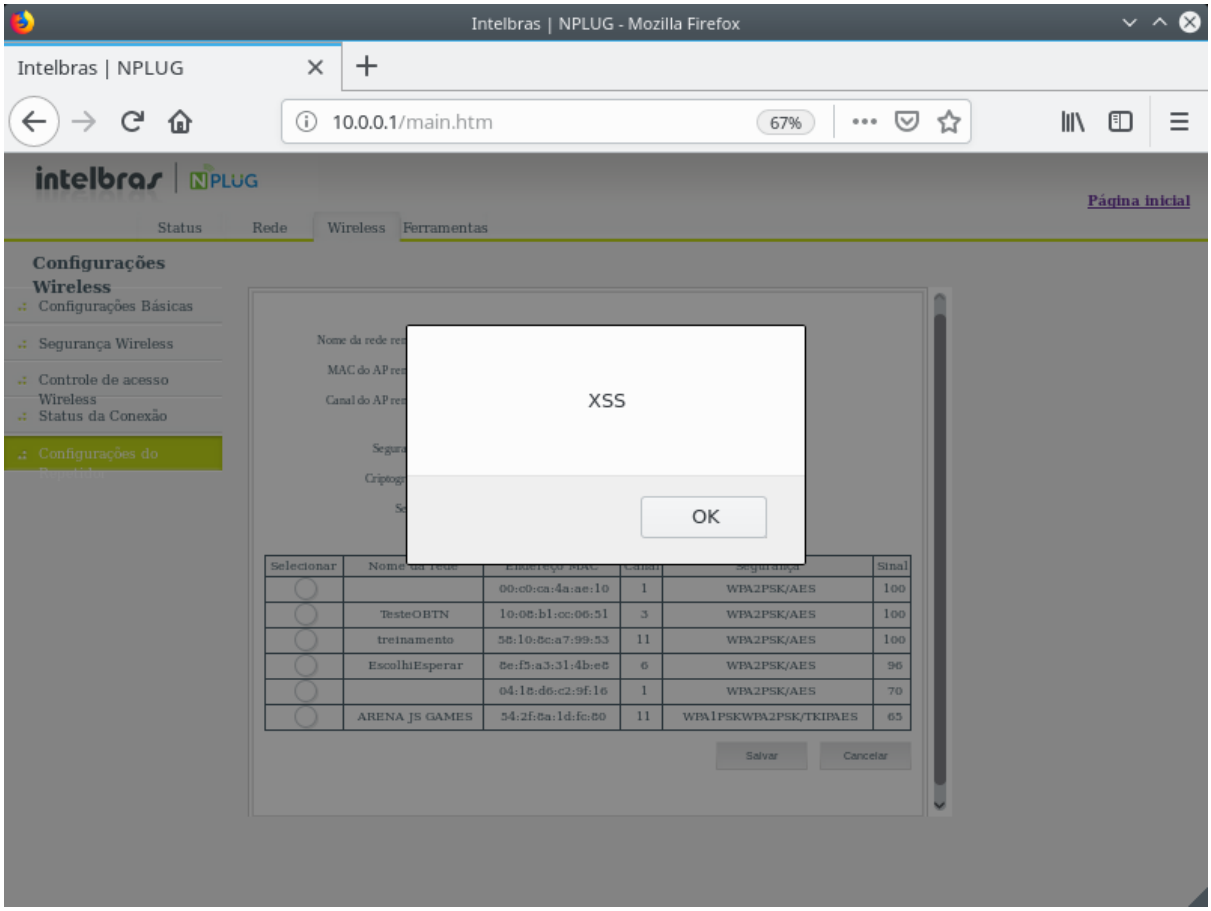
Para fazer a análise da aplicação web é recomendado a utilização de um *proxy* HTTP, tais como mitmproxy[5] e Burp Suite[6]. Essas ferramentas permitem que as requisições HTTP sejam interceptadas e modificadas. No caso do Burp, é possível automatizar requisições e realizar um escaneamento por vulnerabilidades conhecidas.

Durante a análise da interface web do NPLUG, foi possível notar a existência de ao

[2]tcpdump http://www.tcpdump.org/
[3]Wireshark https://www.wireshark.org/
[4]Nmap Security Scanner https://nmap.org/
[5]mitmproyx https://mitmproxy.org/
[6]Burp Suite https://portswigger.net/burp

menos um ponto vulnerável à injeção de código JavaScript[7]. Este ponto foi encontrado na funcionalidade "Configurações do repetidor" e pode ser explorado quando um usuário realiza uma busca por redes *wireless* próximas. Um atacante poderia utilizar-se desta vulnerabilidade enviando um pacote para o *broadcast* contendo um SSID malicioso.

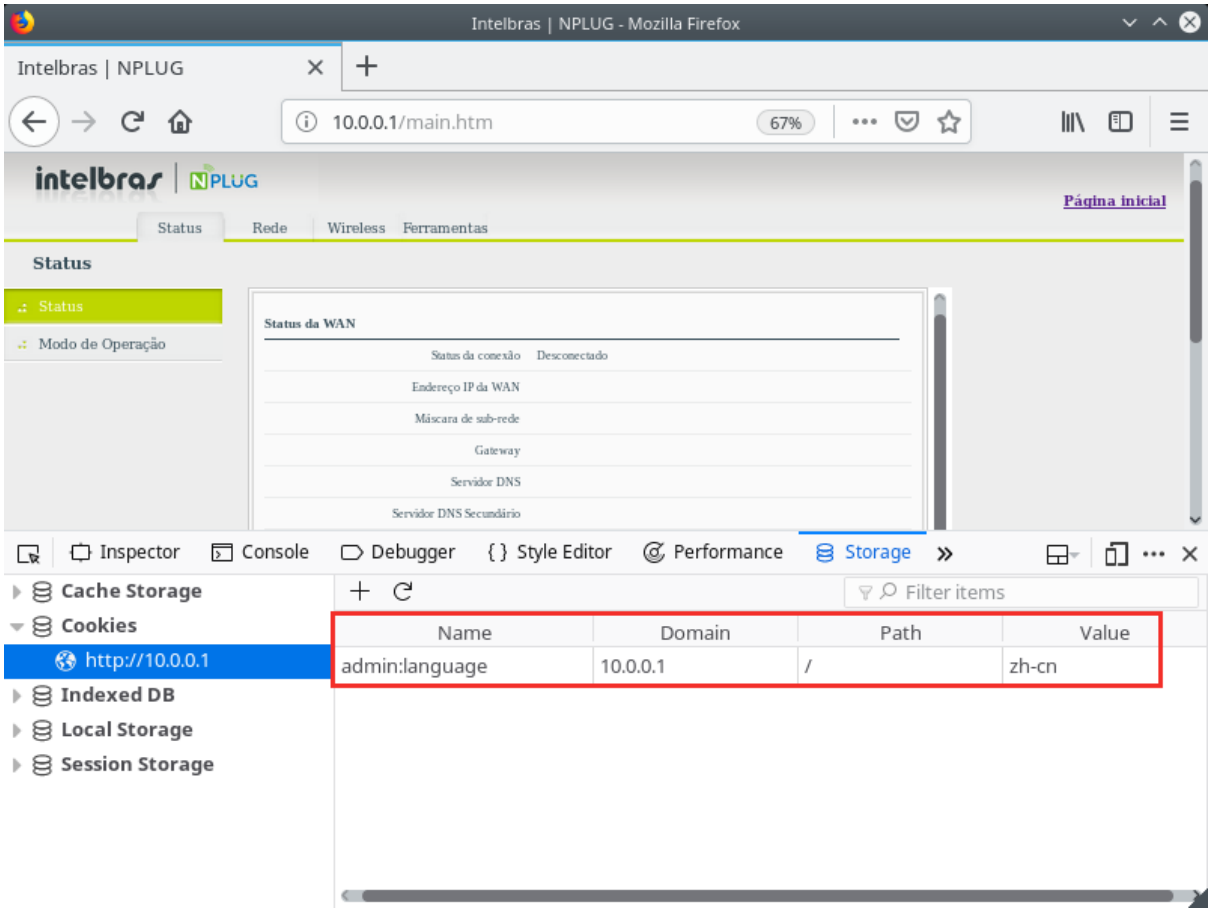
Figura 3.2: Código JavaScript sendo executado na sessão do navegador.
FONTE: O autor (2018)



[7]-2018-17337 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-17337

Uma vez que um XSS (ou XCS) é explorado no navegador da vítima, torna-se possível executar qualquer ação permitida pelo JavaScript do navegador, como por exemplo, sequestrar a sessão do usuário ou alterar configurações do repetidor. No caso do NPLUG, esta vulnerabilidade não é necessária caso o atacante esteja localizado na mesma rede. Um usuário conectado na rede do NPLUG é capaz de acessar as configurações do dispositivo ainda que o mesmo esteja protegido por senha, uma vez que não existente um gerenciamento de sessão apropriado[8].

Figura 3.3: Ausência de gerenciamento de sessão.
FONTE: O autor (2018)



[8]CVE-2018-12455 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12455

3.2 Aquisição do *firmware*

A forma mais fácil de adquirir o *firmware* de um roteador é através do site do fabricante. No entanto, é comum que os *firmware*s de alguns dispositivos não sejam disponibilizados, como forma de dificultar a engenharia reversa. Algumas vezes, embora o fabricante não forneça o *firmware* publicamente, existe a possibilidade de encontrá-lo em sites de terceiros.

Quando o *firmware* não é encontrado para *download* na Internet a opção restante é extraí-lo do próprio dispositivo. O primeiro passo para esta tarefa seria procurar uma porta serial na PCB (*Printed Circuit Board*), geralmente na forma de 4 pinos (ou furos) juntos na placa. É importante a utilização de um multímetro para identificar a função de cada pino antes de conectar-se a eles.

Frenquentemente, fabricantes utilizam a interface UART para depurar dispositivos embarcados. A UART (*Universal Asynchronous Receiver-Transmitter*) é uma interface que atua como uma porta serial e, em conjunto com um conversor UART/USB, permite interações através de um emulador de terminal, como o Screen[9] ou o Minicom[10]. Para conectar-se com sucesso ao dispositivo, é necessário configurar um *baud rate* adequado que trata-se de uma taxa de transmissão dos dados. Outra opção é usar um dispositivo conhecido como Buspirate[11], que permite a comunicação por diversos tipos de protocolos como SPI, JTAG e I²C.

3.3 Extraindo o sistema de arquivos

Uma vez em posse do *firmware* do roteador, é necessário entender o arquivo em questão para utilizar as técnicas certas para analisá-lo. Este processo, em alguns casos, pode não ser trivial, devido ao uso de algoritmos de compressão desconhecidos. Para este estudo, iremos utilizar apenas ferramentas em ambientes Linux.

O *firmware* do NPLUG foi adquirido através do site do fabricante e encontrava-se no

[9] https://help.ubuntu.com/community/Screen
[10] Minicom https://help.ubuntu.com/community/Minicom
[11] Buspirate https://en.wikipedia.org/wiki/Bus_Pirate

formato **zip**. Após descompactá-lo, encontrou-se uma imagem de sistema operacional na arquitetura MIPS. O comando **file** exibe informações sobre um arquivo escolhido, tais como: formato, arquitetura para a qual foi compilado e se as bibliotecas foram incluídas estaticamente ou não.

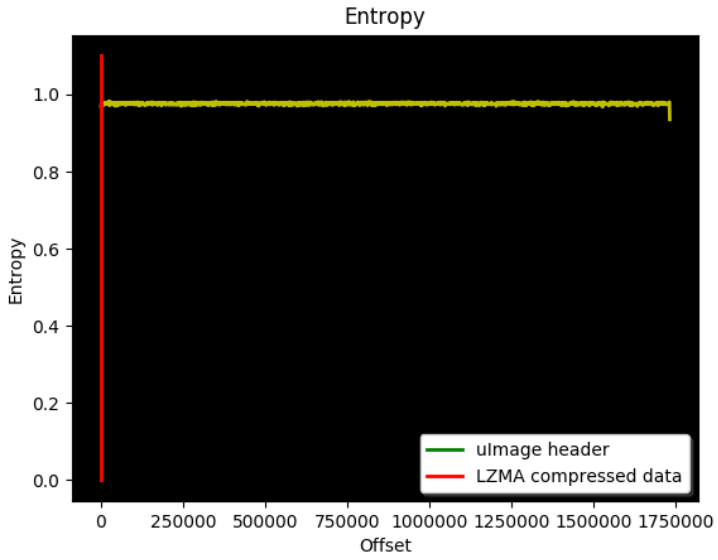
```
$  file fw_nplug_1_0_0_14.zip
fw_nplug_1_0_0_14.zip: Zip archive data, at least v2.0 to extract

$  unzip fw_nplug_1_0_0_14.zip Archive:
 fw_nplug_1_0_0_14.zip inflating: CHANGELOG NPLUG 1_0_0_14.pdf
extracting: fw_NPLUG_1_0_0_14.bin

$  file fw_NPLUG_1_0_0_14.bin
fw_NPLUG_1_0_0_14.bin: u-boot legacy uImage, Linux Kernel Image,
   Linux/MIPS, OS Kernel Image (lzma), 1731916 bytes, Wed Oct 12 17:28:28
   2016, Load Address: 0x80000000, Entry Point: 0x802CB000, Header CRC:
   0xEC8AD091, Data CRC: 0x3A3E5EAC
```

O comando **strings** exibe na tela as cadeias de caracteres imprimíveis de um arquivo. Além de permitir encontrar palavras que passem uma noção do seu conteúdo, este comando pode servir como uma forma rápida de checar se o arquivo está comprimido ou cifrado. Usar o comando **strings** nesta etapa do processo irá retornar majoritariamente dados não legíveis. Isto porque o arquivo em questão trata-se de uma imagem de sistema operacional. Um arquivo **uImage** consiste de uma imagem de *kernel* encapsulada, em conjunto com um cabeçalho que descreve, entre outras coisas, se o *kernel* está comprimido ou não e seu ponto de entrada. Neste caso, é possível visualizar que a imagem está comprimida utilizando o algoritmo LZMA.

O Binwalk[12] é uma ferramenta que realiza a análise de um binário em busca de assinaturas de formatos de arquivo conhecidos. Além disso, podem-se fazer uma análise visual da entropia do arquivo, como demonstra a Figura 3.4, o que permite estimar a quantidade de conteúdo legível no arquivo em seu estado atual e a região em que se encontra.

[12]Binwalk https://github.com/ReFirmLabs/binwalk

Figura 3.4: Análise da entropia no arquivo *fw_NPLUG_1_0_0_14.bin*.
FONTE: O autor (2018)

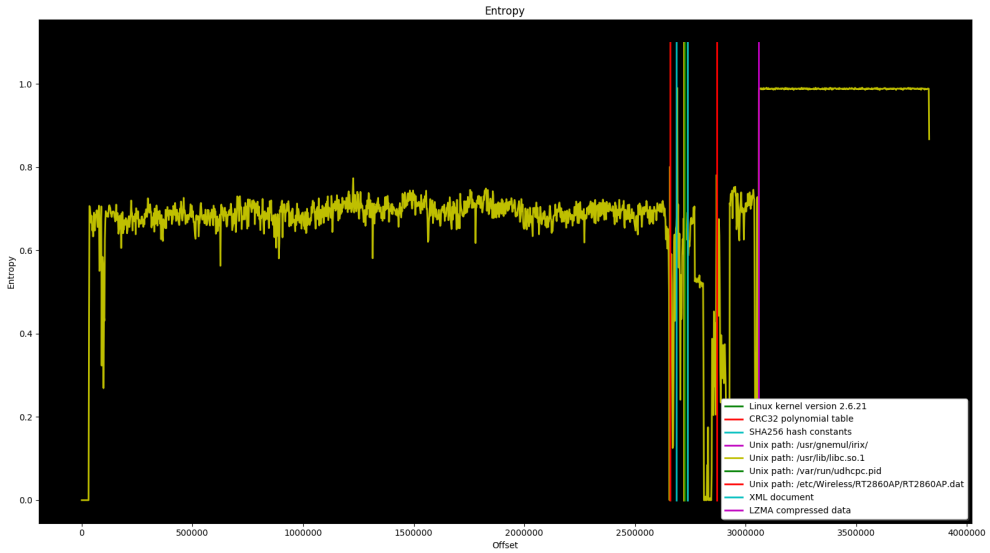



```
$  binwalk -e fw_NPLUG_1_0_0_14.bin

[...]

$  cd _fw_NPLUG_1_0_0_14.bin.extracted/
$  ls
40  40.7z
$  binwalk 40

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
2658360       0x289038        Linux kernel version 2.6.21
2659360       0x289420        CRC32 polynomial table, little endian
2686144       0x28FCC0        SHA256 hash constants, little endian
2720824       0x298438        Unix path: /usr/gnemul/irix/
2721676       0x29878C        Unix path: /usr/lib/libc.so.1
2727580       0x299E9C        Unix path: /var/run/udhcpc.pid
2736880       0x29C2F0        Unix path:
```

```
   /etc/Wireless/RT2860AP/RT2860AP.dat
2737968       0x29C730        XML document, version: "1.0"
2870768       0x2BCDF0        CRC32 polynomial table, little endian
3059712       0x2EB000        LZMA compressed data, properties: 0x5D,
dictionary size: 1048576 bytes, uncompressed size: 2943488 bytes
```

Note que a ferramenta gerou uma pasta contendo dois arquivos. O número **40** é referente à posição em hexadecimal em que a assinatura foi encontrada. O Binwalk extrai o arquivo encontrado e automaticamente faz o descompactamento no diretório. Repetindo o processo de análise visual da entropia, desta vez no arquivo de nome **40**, pode-se visualizar na Figura 3.5 alguns trechos do arquivo apresentando um baixo grau de desordem. Isto pode significar trechos do binário com conteúdo legível que pode ser de alguma valia para análise. No entanto, como nosso objetivo é extrair o sistema de arquivos iremos continuar com o processo.

Figura 3.5: Análise da entropia no arquivo *40*.
FONTE: O autor (2018)

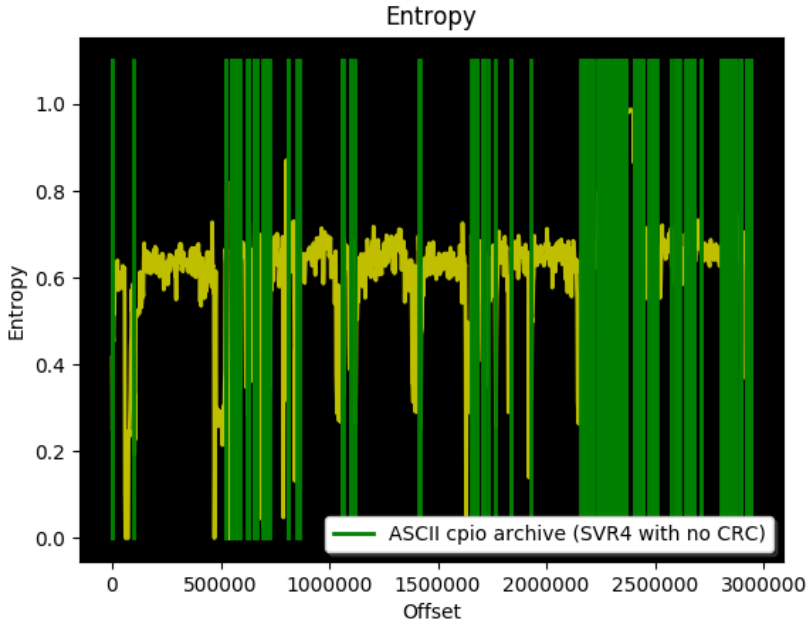


```
$  binwalk -e 40

[...]

$  cd _40.extracted/
$  ls
2EB000  2EB000.7z
$  binwalk -e 2EB000

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ASCII cpio archive (SVR4 with no CRC), file
   name:
"/lib", file name length: "0x00000005", file size: "0x00000000"
116           0x74            ASCII cpio archive (SVR4 with no CRC), file
   name:
"/lib/libc.so.0", file name length: "0x0000000F", file size: "0x00000014"
264           0x108           ASCII cpio archive (SVR4 with no CRC), file
   name:
"/lib/libresolv.so", file name length: "0x00000012", file size:
   "0x00000014"

[...]
```

Utilizando a ferramenta Binwalk no arquivo **2EB000** foram encontrados diversos arquivos em formato legível que correspondem ao *filesystem* do *firmware*. Com isso, acessando o diretório de nome **cpio-root**, foram encontradas as pastas da raiz do sistema, contendo uma estrutura similar à encontrada em distribuições Linux populares.

```
$  ls
bin  dev   etc   etc_ro  home  init lib  media
mnt  proc  sbin  sys     tmp   usr  var
```

Figura 3.6: Análise da entropia no arquivo *2EB000*.
FONTE: O autor (2018)



Este processo de extração recursiva poderia ser automatizado com o próprio Binwalk. Após a extração do sistema, é necessário analisar os arquivos e diretórios disponíveis para entender mais sobre o funcionamento do *firmware*. Em alguns casos, não encontra-se a raiz do *filesystem* de um sistema operacional porque o fabricante pode disponibilizar somente arquivos para atualizar parte do *firmware*, como as páginas de uma interface web. É comum navegar por todas as pastas para entender de modo geral o sistema. Geralmente a pasta **bin** é uma das primeiras a serem analisadas no sistema de arquivos do roteador, no intuito de saber quais binários ele possui.

```
$  ls
ash       default.cfg  iwpriv     netctrl       ps              tquser
ate       dnrd         kill       nvram_clear   ralink_init     udhcpc
ated      echo         ls         nvram_commit  rm              udhcpd
busybox   hostname     miniupnpd  nvram_get     sh
cat       httpd        mkdir      nvram_set     sntp
chmod     inadyn       mount      nvram_show    switch
```

```
cp        iptables     msg         ping          tenda_chck_conn
date      iwconfig     mtd_write   pppd          tenda_deamon
```

A existência do arquivo **httpd** sugere que este seja o binário que inicia o servidor web, responsável pela aplicação de gerenciamento do dispositivo, tornando-o um potencial alvo da pesquisa.

Um ponto de partida para aprender como o dispositivo funciona é buscar por *scripts* que são executados após o *boot* pelo programa *init*. Geralmente os serviços relevantes de um roteador são iniciados por um *script* em **/etc/rcS**, **/etc/init.d/rcS** ou em um *path* similar. No caso do NPLUG, o arquivo que inicia os serviços principais após o *boot* estava localizado no caminho **/etc_ro/rcS**.

```
#!/bin/sh
mount -a
mkdir -p /var/run
cat /etc_ro/motd

#nvram_daemon&
#goahead&
netctrl&

#for telnet debugging #telnetd
#for syslogd #mkdir -p /var/log
```

O *script* inicia montando os volumes necessários e exibindo uma mensagem padrão. Apesar de algumas instruções estarem comentadas, estas dão dicas válidas sobre o dispositivo. A primeira, **nvram_daemon**, apesar de estar comentada, juntamente com a listagem de arquivos da pasta **bin**, sugere que o dispositivo faz uso de uma memória não-volátil (NVRAM). A NVRAM é utilizada para armazenamento das configurações devido a sua persistência de dados, mesmo após o desligamento do dispositivo. Quaisquer alterações que não sejam armazenadas na NVRAM são descartadas ao desligar ou reiniciar o roteador.

A instrução seguinte sugere que o dispositivo utiliza um servidor web para sistemas embarcados chamado GoAhead[13]. Em seguida, é executado um binário **netctrl** em modo *background*. A instrução **telnetd**, apesar de estar comentada, leva a crer que o dispositivo dá a possibilidade de habilitar um serviço Telnet[14] para ser acessado remotamente. Várias outras inferências poderiam ser feitas apenas navegando pelos diretórios do *firmware*, no entanto, iremos prosseguir para uma análise mais aprofundada do binário **httpd** na seção seguinte.

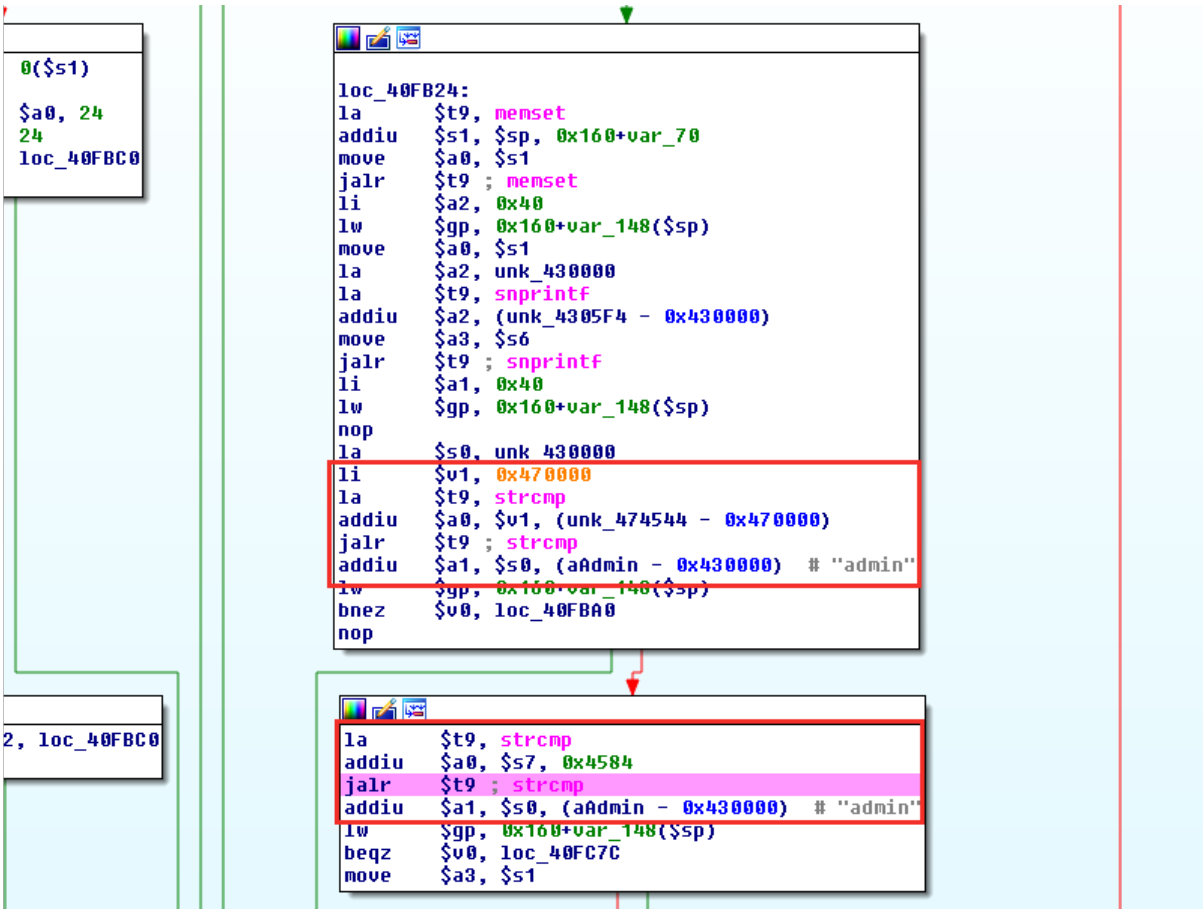
3.4 Análise estática

A análise estática consiste no estudo do chamado código "morto", supondo o possível comportamento das aplicações quando executadas. No cenário dos roteadores SOHO, ela pode ser executada tanto em binários – responsáveis por serviços como HTTP, UPnP e DHCP – quanto em códigos escritos em linguagens com maior nível de abstração. Em alguns casos, é possível extrair somente a interface web e realizar uma análise estática da mesma, em outros, a interface web está acoplada ao servidor em um único binário, dificultando a execução em outro ambiente.

Analisando o arquivo **httpd**, encontrado dentro do sistema de arquivos do *firmware* **fw_NPLUG_1_0_0_14.bin**, foi possível encontrar rapidamente a função responsável por gerenciar os *cookies* de sessão do navegador, devido ao seu nome sugestivo. Nesta função existem algumas comparações com uma constante (admin), o que pode ser um indício da vulnerabilidade CVE-2018-12455.

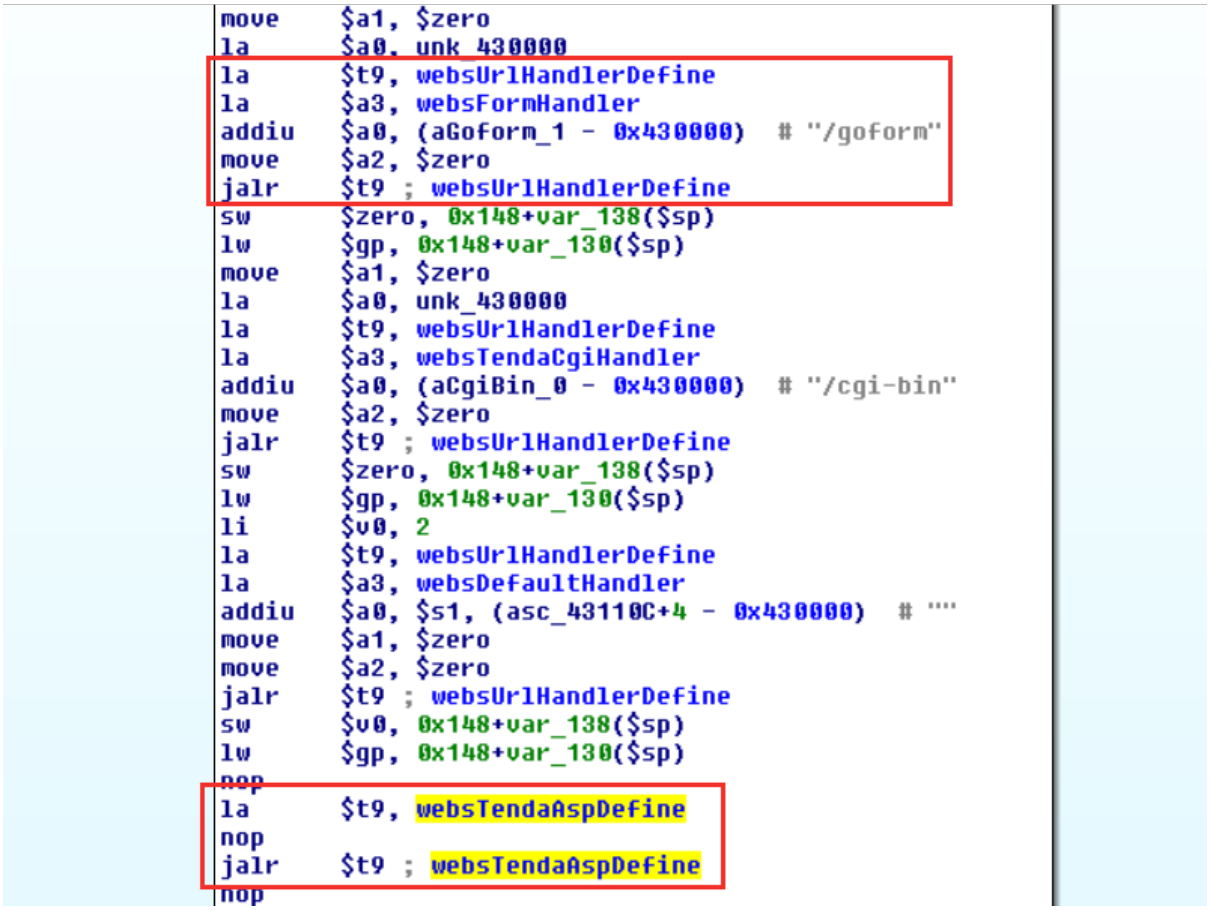
[13]GoAhead https://www.embedthis.com/goahead/
[14]Telnet https://linux.die.net/man/1/telnet

Figura 3.7: Comparação do *cookie* com um valor constante.
FONTE: O autor (2018)



Em seguida, foi encontrado o ponto em que alguns *handlers* são registrados. O *handler* de uma requisição web refere-se, geralmente, a uma função escolhida para gerenciar o comportamento da aplicação quando o *path* ou outros parâmetros da requisição correspondem a um valor pré-definido.

Figura 3.8: Definição de funções para gerenciar requisições.
FONTE: O autor (2018)



Na Figura 3.8 uma função (**websFormHandler**) é definida para ser executada quando o *path* da requisição for igual ao valor **/goform**. Posteriormente, após algumas operações, a função **websAspDefine** é chamada. Novas instruções são executadas e em seguida, outros *handlers* são definidos, utilizando a função **websFormDefine**. Desta vez o que nos chama atenção é o registro do *form* **telnet**. Como visto nos passos anteriores, o NPLUG possui o binário do Telnet, no entanto este serviço não é iniciado no *boot*. Estes fatos levantam a hipótese

de que o serviço pode ser ativado de alguma forma.

Figura 3.9: Definição da função para o *path* /goform/telnet.
FONTE: O autor (2018)

Acessando a função responsável por gerenciar o *path* **/goform/telnet** foi possível encontrar a execução da chamada ao sistema, **system**, contendo como parâmetro o valor **telnetd**, nome do binário do serviço Telnet. Observando a Figura 3.10, nota-se que a aplicação define o usuário do serviço como **admin** e não define uma senha, o que torna o dispositivo ainda mais vulnerável.

Figura 3.10: Aplicação inciando o servidor Telnet.
FONTE: O autor (2018)

```
la      $t9, system
nop
jalr    $t9 ; system
addiu   $a0, (aEchoAdmin00Adm - 0x430000)  # "echo 'admin::0:0:admin:/:/bin/sh'
lw      $gp, 0x20+var_10($sp)
nop
la      $a0, unk_430000
la      $t9, system
nop
jalr    $t9 ; system
addiu   $a0, (aEchoAdminX0Adm - 0x430000)  # "echo 'admin:x:0:admin' > /etc/gro
lw      $gp, 0x20+var_10($sp)
nop
la      $a0, unk_430000
la      $t9, system
nop
jalr    $t9 ; system
addiu   $a0, (aKillall9Telnet - 0x430000)  # "killall -9 telnetd"
lw      $gp, 0x20+var_10($sp)
nop
la      $a0, unk_430000
la      $t9, system
nop
jalr    $t9 ; system
addiu   $a0, (aTelnetd - 0x430000)  # "telnetd"
lw      $gp, 0x20+var_10($sp)
move    $a0, $s0
la      $a1, unk_430000
la      $t9, websWrite
```

Existem ferramentas que podem automatizar o processo de identificação de vulnerabilidades como as apresentadas anteriormente. Costin et al. (2014) foram responsáveis pela criação do Firmware.re[15], um serviço web para extração e análise de imagens de *firmwares* utilizando um *framework* autoral com performance de extração superior as outras ferramentas como BAT, Binwalk e FRAK, segundo o autor.

Shoshitaishvili et al. (2015) desenvolveram um *framework* para análise estática de código binário em *firmwares* de embarcados. A ferramenta foi desenvolvida com o foco em achar vulnerabilidades nos mecanismos de autenticação, sem a necessidade de instrumentar o código do dispositivo. No entanto, o Firmalice requer a inserção de informações, relativas à "Polí-

[15]Firmware.re http://firmware.re/

tica de Segurança" da aplicação, antes da execução da análise de cada *firmware*, tornando a ferramenta pouco escalável.

Dois anos depois, Costin, Zarras e Francillon (2016) apresentaram o primeiro *framework* para uma análise automatizada e escalável de *firmwares* a fim de encontrar vulnerabilidades em dispositivos embarcados. Como resultado do estudo, encontraram falhas em pelo menos 24% das interfaces web que foram emuladas, totalizando 9271 vulnerabilidades em 185 *firmwares*. Costin, Zarras e Francillon (2016), ressaltam que o emulador criado não é perfeito, uma vez que emular um *hardware* desconhecido é praticamente impossível. Além disso, vários problemas foram encontrados para emular os dispositivos, como *kernels* customizados ou inexistentes. O *kernel* é o núcleo do sistema operacional, responsável por controlar os componentes, incluindo sistema de arquivos, memória, interfaces de rede e USB. As funcionalidades dos *firmwares* baseiam-se em interações complexas entre arquivos executáveis, bibliotecas e componentes do *kernel* (PAPP; MA; BUTTYAN, 2015). Portanto, o *kernel* possui um papel fundamental na segurança dos dispositivos, embora seja comumente encontrado em versões desatualizadas.

Eschweiler, Yakdan e Gerhards-Padilla (2016) apresentaram uma abordagem para identificação de similaridades de código binário em múltiplas arquiteturas (x86, x64, ARM, MIPS). Estas semelhanças trazem à discussão o problema do reuso de código entre os fabricantes. Diversas vulnerabilidades, encontradas e publicadas na década passada, continuam circulando em dispositivos vendidos atualmente. Este é o caso de vulnerabilidades encontradas no NPLUG.

3.5 Análise dinâmica

A análise dinâmica possui a vantagem de permitir estudar o *firmware* após a inicialização de variáveis de ambiente e com a presença de alguns arquivos escritos no *boot*. No entanto, emular e depurar (*debug*) o dispositivo é, talvez, uma das tarefas mais difíceis na engenharia reversa de roteadores. Este procedimento é uma forma eficaz de se confirmar algumas vulnerabilidades encontradas na análise estática. Utilizar as interfaces de depuração existentes no *hardware* do dispositivo real seria a maneira mais direta, no entanto, adquirir roteadores para estudo pode ser uma barreira em alguns casos. Felizmente, a análise dinâmica também pode ser executada emulando o dispositivo.

Devido à utilização de arquitetura não convencionais (tais como ARM ou MIPS), faz-se necessário o uso de uma máquina virtual para executar os binários do dispositivo. Para esta etapa, costuma-se utilizar o QEMU[16], um *software* livre que permite emular diversos tipos de arquiteturas de processadores. O sistema de arquivos extraído anteriormente pode ser enviado à máquina virtual no QEMU através do SSH. É possível fazer uso do comando **chroot** para mudar virtualmente o diretório raiz para a pasta contendo o sistema de arquivos do *firmware*.

Executar os programas do *firmware* extraído pode não ser tão simples, uma vez que alguns *firmwares* utilizam recursos específicos do *hardware* hospedeiro, tal qual a NVRAM mencionada anteriormente. No caso da NVRAM, é possível implementar um mecanismo que intercepte as chamadas do sistema para a memória e as responda com um valor adequado. Com serviços rodando no emulador, pode-se realizar análises utilizando ferramentas como Nmap e Metasploit[17]. Uma alternativa proposta por Zaddach et al. (2014) é o *framework* Avatar, capaz de analisar dinamicamente *firmwares* de embarcados. A ferramenta executa o código do *firmware* em uma máquina virtual e encaminha as requisições de I/O para o dispositivo real.

Uma vez executando programas do *firmware*, é de interesse da análise de segurança depurá-los como forma de entender melhor o comportamento e explorar vulnerabilidades mais avançadas. Para isso, é preciso realizar o processo de *cross-compiling* do programa GDB[18]

[16]QEMU https://www.qemu.org/
[17]Metasploit Framework https://www.metasploit.com/
[18]The GNU Project Debugger https://www.gnu.org/software/gdb

(ou outro *debugger*) para permitir sua execução na arquitetura do *firmware*. O GDB pode ser enviado ao dispositivo, real ou emulado, para ser executado em conjunto com o processo alvo da análise.

Chen et al. (2016) propôs o **Firmadyne**, um *framework* para análise dinâmica e automatizada. O **Firmadyne** não possui mecanismos para encontrar vulnerabilidades automaticamente, mas cria o ambiente propício para isso, através da emulação do dispositivo. O *framework* conta ainda com um *crawler* que foi utilizado para fazer o *download* de *firmwares* em sites de fabricantes.

CAPÍTULO 4

Conclusão

Este trabalho apresentou uma revisão bibliográfica sobre a engenharia reversa no *firmware* de roteadores SOHO, no intuito de descobrir vulnerabilidades. As técnicas apresentadas seguiram métodos relatados na literatura, e provaram-se úteis para o estudo do funcionamento de um *firmware*, mesmo sem acesso ao código-fonte. Alguns dos passos foram exemplificados com o uso de um repetidor Wi-fi, no entanto, nem todas etapas puderam ser executadas neste dispositivo devido à peculiaridade de cada imagem. Ademais, seguindo as técnicas relatadas neste documento, é possível encontrar novas vulnerabilidades até mesmo no dispositivo utilizado na pesquisa.

Ainda há a possibilidade de melhoria nas análises estáticas e dinâmicas em *firmwares* de roteadores — como o suporte a múltiplas arquiteturas, redução de falso positivos e negativos, além de técnicas mais avançadas para identificação de vulnerabilidades em binários. Entretanto, é notável que os avanços nas pesquisas da última década proporcionam um ambiente mais favorável ao estudo desses dispositivos. Existe claramente um esforço de pesquisadores em tornar o processo automatizado, de modo que diversos *firmwares* possam ser analisados com menos esforço e que algumas das ferramentas possam ser incorporadas em seu processo de produção.

Os estudos mostram que as segurança dos roteadores SOHO é precária, trazendo como consequência um grande risco aos usuários de todo o mundo. Portanto, conclui-se que a engenharia reversa de roteadores SOHO é uma tarefa não trivial, mas factível e necessária para compelir os fabricantes a não negligenciarem os aspectos de segurança em seus produtos.

APÊNDICE A

Lista completa de artigos selecionados

Autores	Título	Ano
Szewczyk, P.	ADSL Router Forensics Part 1: An introduction to a new source of electronic evidence.	2007
Szewczyk, P.	ADSL Router Forensics Part 2: Acquiring Evidence	2009
Szewczyk, P., Valli, C.	Insecurity by Obscurity: A Review of SoHo Router Literature from a Network Security Perspective	2009
Bojinov, H., et al.	Embedded management interfaces: Emerging massive insecurity	2009
Yin, Z., et al.	Towards Understanding Bugs in Open Source Router Software	2010
Szewczyk, P.	The ADSL Router Forensics Process	2010
B. Bencsáth; et al.	XCS based hidden firmware modification on embedded devices	2011
Jones, Neil	Exploiting Embedded Devices	2012
Zaddach, J., Costin, A.	Embedded devices security and firmware reverse engineering	2013
Safford, D.	Embedded Linux Integrity	2013
Fiebig, T.	Forensic DHCP Information Extraction from Home Routers	2013
Safford, David	Embedded Linux Integrity	2013
Cui, A., et al.	When firmware modifications attack: A case study of embedded exploitation	2013

Costin, Andrei, et al.	A Large-Scale Analysis of the Security of Embedded Firmwares	2014
Zaddach, J., et al.	AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems	2014
Kammerstetter, M., et al.	Prospect: Peripheral Proxying Supported Embedded Code Testing	2014
Gourdin, Baptiste, et al.	Toward Secure Embedded Web Interfaces	2015
Schulte, E. M., et al.	Repairing COTS Router Firmware Without Access to Source Code or Test Suites: A Case Study in Evolutionary Software Repair	2015
Küçüksille, E. U., et al.	Developing a Penetration Test Methodology in Ensuring Router Security and Testing It in a Virtual Laboratory	2015
Pewny, J., et al.	Cross-Architecture Bug Search in Binary Executables	2015
Poornachandran, P., et al.	Internet of Vulnerable Things (IoVT): Detecting Vulnerable SOHO Routers	2015
Shoshitaishvili, Yan, et al.	Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware	2015
Hampton, N., Szewczyk, P.	A survey and method for analysing SoHo router firmware currency	2015
Costin, A., et al.	Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces	2016
Chen, Daming D., et al.	Towards Automated Dynamic Analysis for Linux-based Embedded Firmware	2016
Eschweiler, S., et al.	discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code	2016
Rotenberg, Nadav, et al.	Authentication-Bypass Vulnerabilities in SOHO Routers	2017
Teng, Che-Chun, et al.	Firmware over the air for home cybersecurity in the Internet of Things	2017

Tran, Nghi-Phu, et al.	Towards malware detection in routers with C500-toolkit	2017
L. Thomas, et al.	HumIDIFy: A Tool for Hidden Functionality Detection in Firmware	2017
Szewczyk, P., Macdonald, R.	Broadband Router Security: History, Challenges and Future Implications	2017
Folgado Rueda, Á., et al.	Revisiting SOHO Router Attacks	2017
Visoottiviseth, Vasaka, et al.	Firmaster: Analysis Tool for Home Router Firmware	2018
Cheng, Kai, et al.	DTaint: Detecting the Taint-Style Vulnerability in Embedded Device Firmware	2018

Referências Bibliográficas

BBC. *Wi-fi owner fined for lax security in Germanyj*. 2010. Disponível em: <https://www.bbc.com/news/10116606>.

BOHIO, M. J. Analyzing a backdoor/bot for the mips platform. *SANS Institute*, 2015. Disponível em: <https://www.sans.org/reading-room/whitepapers/malicious/analyzing-backdoor-bot-mips-platform-35902>.

BOJINOV, H.; BURSZTEIN, E.; BONEH, D. Xcs: Cross channel scripting and its impact on web applications. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2009. (CCS '09), p. 420–431. ISBN 978-1-60558-894-0. Disponível em: <http://doi.acm.org/10.1145/1653662.1653713>.

BOJINOV, H. et al. Embedded management interfaces: Emerging massive insecurity. *BlackHat USA*, Citeseer, v. 1, n. 8, p. 14, 2009.

CHEN, D. D. et al. Towards automated dynamic analysis for linux-based embedded firmware. In: *NDSS*. [S.l.: s.n.], 2016.

COSTIN, A.; ZADDACH, J. Embedded devices security and firmware reverse engineering. *BH13US Workshop*, Las Vegas, NV, USA, 2013.

COSTIN, A. et al. A large-scale analysis of the security of embedded firmwares. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. Berkeley, CA, USA: USENIX Association, 2014. (SEC'14), p. 95–110. ISBN 978-1-931971-15-7. Disponível em: <http://dl.acm.org/citation.cfm?id=2671225.2671232>.

COSTIN, A.; ZARRAS, A.; FRANCILLON, A. Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2016. (ASIA CCS '16), p. 437–448. ISBN 978-1-4503-4233-9. Disponível em: <http://doi.acm.org/10.1145/2897845.2897900>.

EILAM, E. *Reversing: Secrets of Reverse Engineering*. New York, NY, USA: John Wiley & Sons, Inc., 2005. ISBN 9780764574818.

ESCHWEILER, S.; YAKDAN, K.; GERHARDS-PADILLA, E. discovre: Efficient cross-architecture identification of bugs in binary code. In: *NDSS*. [S.l.: s.n.], 2016.

GHITA, B.; FURNELL, S. Assessing the usability of wlan security for soho users. *ECU Publications*, 2018.

GOODIN, D. et al. *Hackers infect 500,000 consumer routers all over the world with malware*. Ars Technica, 2018. Disponível em: <https://arstechnica.com/information-technology/2018/05/hackers-infect-500000-consumer-routers-all-over-the-world-with-malware/?comments=1>.

GOURDIN, B. et al. Toward secure embedded web interfaces. In: *Proceedings of the 20th USENIX Conference on Security*. Berkeley, CA, USA: USENIX Association, 2011. (SEC'11), p. 2–2. Disponível em: <http://dl.acm.org/citation.cfm?id=2028067.2028069>.

HEFFNER, C. *Reverse Engineering a D-Link Backdoor*. 2013. Disponível em: <http://www.devttys0.com/2013/10/reverse-engineering-a-d-link-backdoor/>.

PAPP, D.; MA, Z.; BUTTYAN, L. Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In: IEEE. *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*. [S.l.], 2015. p. 145–152.

POORNACHANDRAN, P. et al. Internet of vulnerable things (iovt): Detecting vulnerable soho routers. In: *2015 International Conference on Information Technology (ICIT)*. [S.l.: s.n.], 2015. p. 119–123.

SHOSHITAISHVILI, Y. et al. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. In: *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015. Disponível em: <https://doi.org/10.14722/ndss.2015.23294>.

SZEWCZYK, P.; MACDONALD, R. Broadband router security: History, challenges and future implications. *The Journal of Digital Forensics, Security and Law*, 2017.

UNCTAD. *Information Economy Report*. 2017. Disponível em: <https://unctad.org/en/PublicationsLibrary/ier2017_en.pdf>.

ZADDACH, J. et al. Avatar: A framework to support dynamic security analysis of embedded systems' firmwares. In: *Proceedings 2014 Network and Distributed System Security Symposium*. Internet Society, 2014. Disponível em: <https://doi.org/10.14722/ndss.2014.23229>.

Este volume foi tipografado em LaTeX na classe UFPEThesis (<www.cin.ufpe.br/~paguso/ufpethesis>).