Exploit Wednesdays

AN OVERVIEW OF BINARY PATCH ANALYSIS

CARTER JONES
SECURITY CONSULTANT

Overview

- Who I am
- How I got into security
- Hacking video games
- Binary diffing
- Questions

whoami

- Security consultant @ Cigital
- Security researcher previously
- Student of life (always learning)
- Husband
- Music lover
  - Mostly techno: DnB, Trance, and EDM
- Video game player/hacker

cigital

Who likes video games?

cigital

How I Got Into Security

- Video games: they're fun, but difficult
- There are 2 solutions to this problem:
  1. Practice to eventually get better
  2. Cheat (hack) to get way better right now
- Too lazy/impatient to practice, so I learned how to hack

Hacking Games: Demo

cigital

Hacking Games

- Example Goals
  - Get unlimited health
  - Gain access to restricted parts of the game:
    - See hidden parts of a map
    - Access powers that are above current level
- What other types of hacks would you want to use in a game?

Hacking Games: General Approach

- Modify the game to reach your goal
  - While the game is running (dynamic)
  - Before it is launched/on disk (static)
- Overall process:
  - Find out how the game stores/manipulates data of interest (e.g.: current health value)
  - Change how the game stores/manipulates that data (e.g.: don’t reduce health value)

Hacking Games: Process Review

- Think like an attacker
- Have a goal-based approach
- Have the skills to reach those goals
  - Learn the skills, if needed
- Be creative with the techniques used
- Have fun with it!

Hacking Games: Skills Required

- Reverse engineering/dynamic analysis
  - Find target values in memory
  - Identify code of interest
  - Understand how the code works
  - Modify code flow to suit our needs
- These same skills are used in the security industry every day

How does this apply to the industry?

- The skills needed to hack games are generally required in the following fields:
  - digital forensics
  - malware analysis
  - **vulnerability research**
  - offensive security research (writing exploits)
  - embedded systems analysis
  - mobile research
  - and many more

cigital

Vulnerability Research

- Lots of different types of vulnerability research (the following is just a start):
  - Vulnerability discovery
    - Fuzzing (finds bugs)
    - Analysis of design specifications (finds flaws)
  - **Vulnerability (re)discovery: binary diffing**
  - Exploit development
    - Development of proof of concept code
    - Development of weaponized exploits

Binary Diffing

- Binary diffing (aka: bindiffing) is a method used to analyze the changes that are introduced by patches
- Helps identify exactly what code was fixed in a patch
- Required because patch descriptions from vendors are often intentionally very vague

Binary Diffing: Purpose

- Vulnerabilities that are identified can be used to create:
  - Proof of concepts (aka: PoCs): something that triggers the vulnerability
  - Exploits based on the PoCs
  - Behavioral-based exploit-detection signatures (based on the calls involved when triggering the vulnerability)

Binary Diffing: What is it?

- Binary diffing takes two compiled files:
  - Original file: Before a patch is applied
  - Patched file: After a patch is applied
- These two files are compared to find what exactly was patched
- This is very useful for analyzing security patches, such as those released on Microsoft's Patch Tuesday

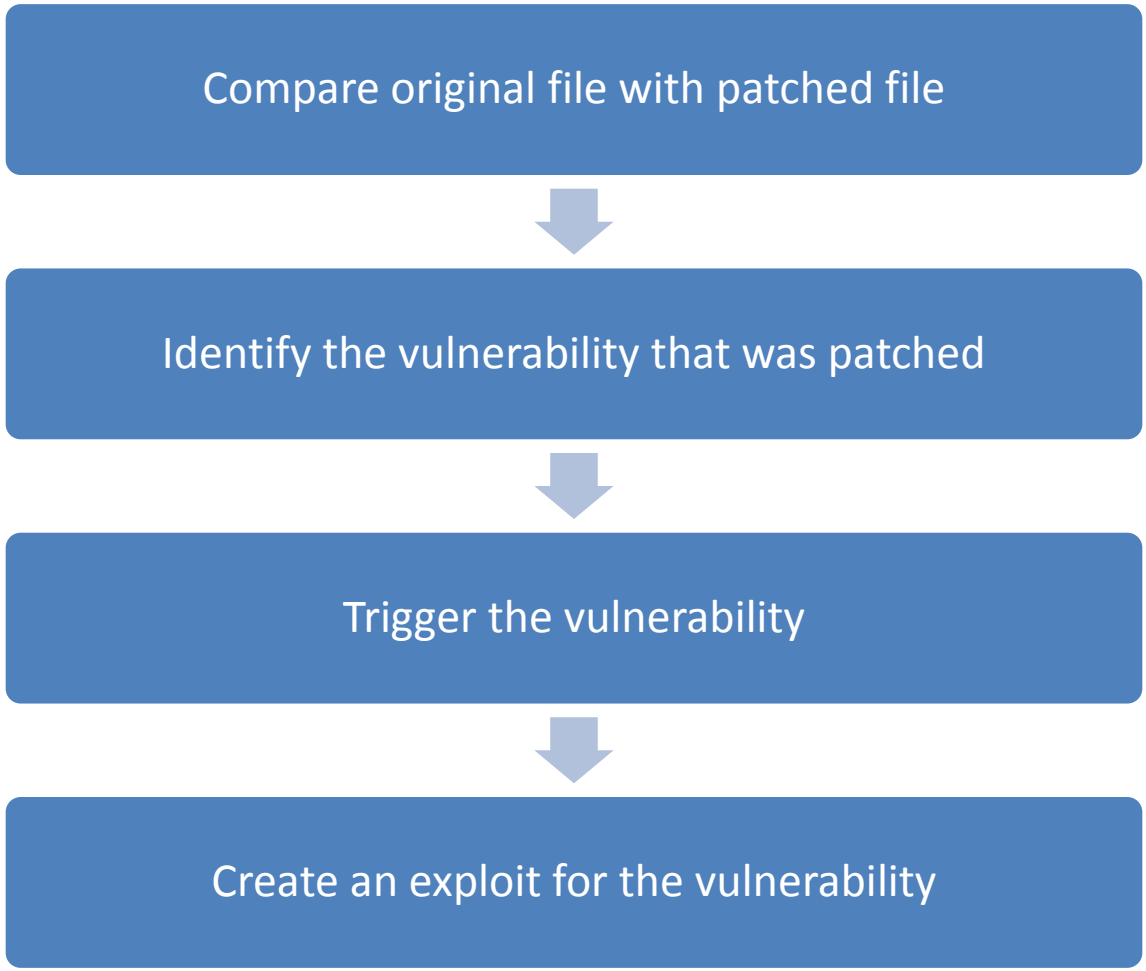
Binary Diffing: Exploit Wednesday

- Patch Tuesday occurs on the second Tuesday of every month
- Attackers analyze these patches using bindiffing techniques and develop exploits
- Sometimes happens within a day
- Patch Tuesday + 1 day = “Exploit Wednesday”

Are there any questions so far?

cigital

Binary Diffing: Overview



cigital

Binary Diffing: A Simple Example

```
push    dword ptr [esi] ; void *
call    ??3@YAXPAX@Z    ; operator delete(void *)
pop     ecx
```


```
push    dword ptr [esi] ; void *
call    ??3@YAXPAX@Z    ; operator delete(void *)
pop     ecx
mov     [esi], ebx
```


Binary Diffing: A Simple Example (cont.)

```
push    dword ptr [esi] ; void *
call    ??3@YAXPAX@Z    ; operator delete(void *)
pop     ecx
mov     [esi], ebx
```

- The added instruction is used to clear out the value of [ESI] (ebx = 0)
- Translation:
  - Before the patch, a pointer was not properly cleared out (it was not properly set to 0)
  - The patch fixed this

Binary Diffing: A Simple Example (cont.)

- That patch was for CVE-2014-0301
- Description from cve.mitre.org:
  - “Double free vulnerability in qedit.dll in DirectShow in Microsoft Windows... allows remote attackers to execute arbitrary code via a crafted JPEG image”
- Translation: a user views my maliciously crafted picture and I can run code on their computer

cigital

Exploited!

cigital

Binary Diffing: General Process

- How to find the patched code of interest:
  - Get the old and new versions of the file
  - Diff the files
  - Sort through the noise to find the real issue
- There are practical ways to do this and there are less practical ways

Try to think of some ways to compare two binary files. What approaches come to mind?

cigital

Binary Diffing: Less Practical Approach #1

- Doing a byte by byte analysis of the code
- This will make about as much sense as trying to parse this picture without a computer and/or calculator

cigital

 |

Binary Diffing: Less Practical Approach #1 (cont.)

- Byte-by-byte comparison is not useful as a generic approach
- On the x86 architecture (and others), there are variable-length instructions
- X86: 1 instruction can be up to 14 bytes
- Swapping sequential, semantically separate instructions can cause up to 28 bytes of change

cigital

Binary Diffing: Less Practical Approach #1 (cont.)

- Consider the example from earlier
- The byte changes of the instructions are:
  - Original: `14684D70 FF 36 E8 11 D0 FD FF 59`
  - Patched: `14684D5F FF 36 E8 12 D0 FD FF 59 89 1E`
- Problem areas:
  - Shared instructions show variation
    - `11` becomes `12`
  - The addresses are completely different
    - Address ended in `70`, but now ends in `5F`

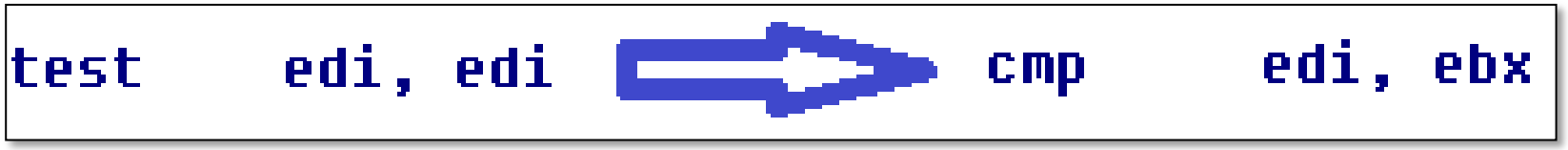
cigital

 |

Binary Diffing: Less Practical Approach #2

- Disassemble all the code from the before/after files and do a source code comparison of the disassembled code
- Possible, but not always practical
- Generates lots of noise, due to structural changes
- This becomes a search for a needle in a haystack

Binary Diffing: Noise



- Noise is usually introduced in a patch
  - Compiler option change (e.g. optimization flags)
  - Feature updates
  - Modification of the compiler itself
  - Refactoring
  - Code cleanup

Binary Diffing: Practical Approach #1

- Use tools
  - DarunGrim (free)
  - BinDiff ($$$)
- Both require IDA Pro ($$$)
  - Interactive DisAssembler
- This all costs $$$, but if you work in the industry, your employer will pay for it

DarunGrim

zynamics BinDiff

cigital

Binary Diffing: Practical Approach #2

- Write your own disassembler by using freely available disassembler libraries
  - BeaEngine (LGPL 3) 
  - diStorm (GPL 3) 
- I started to do this and then realized why commercial products like IDA exist
- Automatic file disassembly is **really hard**
  - Lots of difficulties surrounding file structure
    - PE files / ELF / Mach-O

cigital

Binary Diffing: Academic World vs Industry

- Academic world
  - There are more opportunities to spend time inventing a better wheel
- Security industry
  - Results are needed sooner rather than later
  - It is more cost effective to use existing tools that might not be perfect, but get the job done faster in the short term
- Note: there are always exceptions

cigital

Binary Diffing: Tools

- Highlight changes, additions, and removals of instructions
- Can usually be used to jump to the relevant code within IDA
- Are sometimes extensible
  - Allows for writing plugins
  - Can also create external standalone applications that use the core of the tools

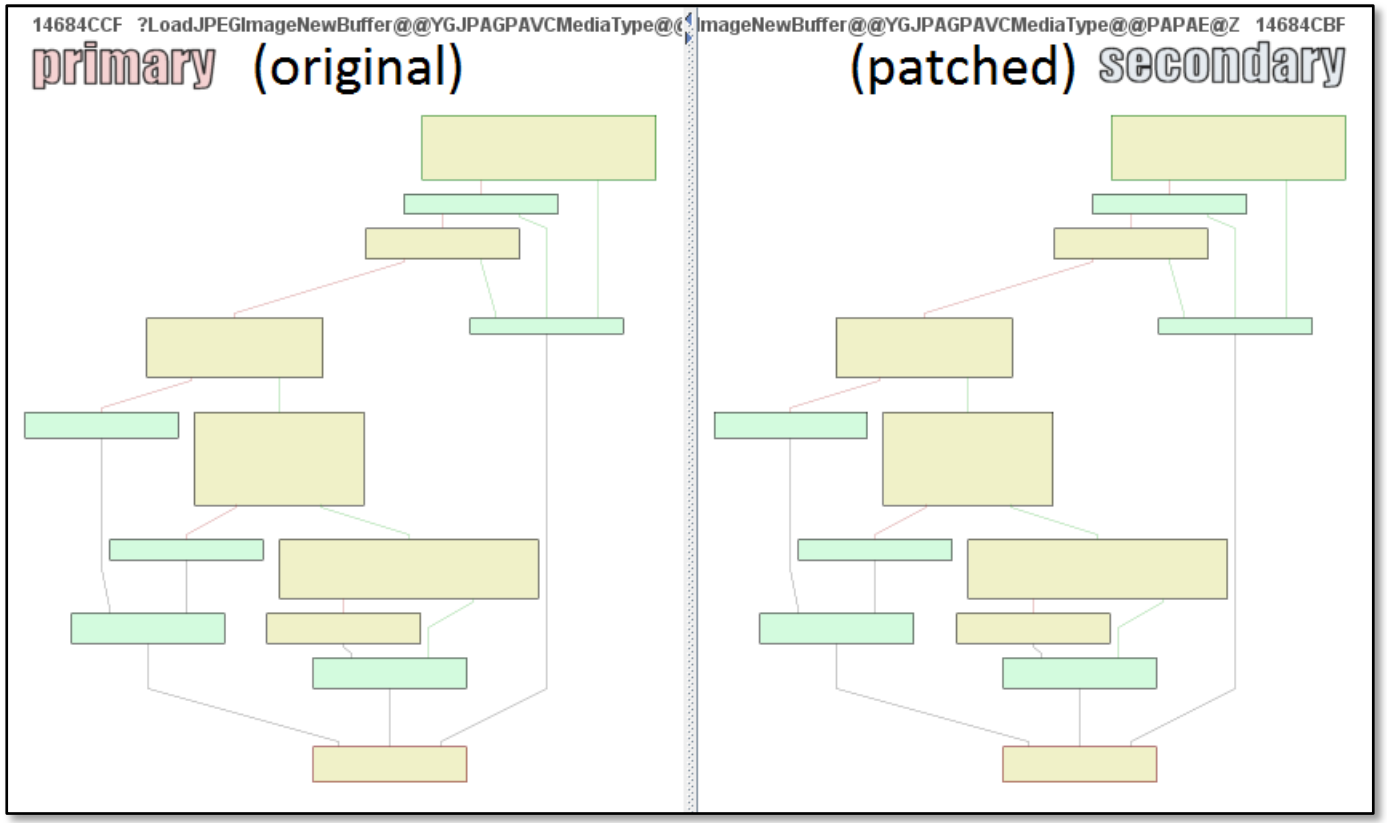
Binary Diffing: Tool-Specific Advantages

- DarunGrim advantages
  - Provides a “security implication score”
    - Helps separate security vs regular patches
  - Open source (free!)
- BinDiff advantages
  - Synchronizes original/patched views
  - Allows filtering by change type:
    - Structural changes
    - Instructions changed

cigital

Binary Diffing: BinDiff Changes At-A-Glance

- View changes to functions at a glance



Binary Diffing: DarunGrim Advantages

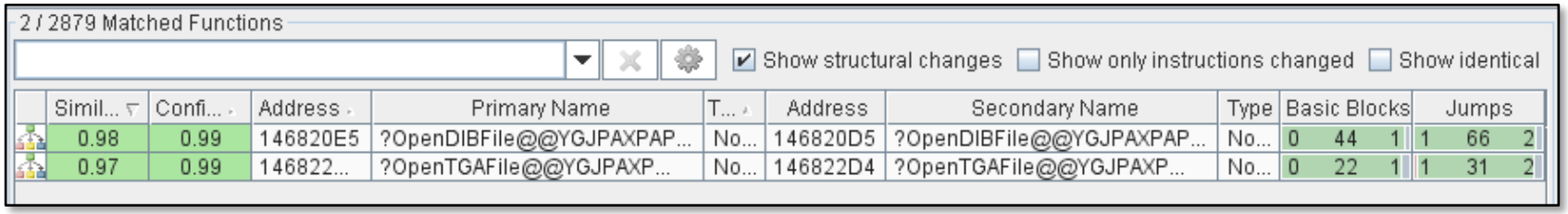
- Compare original and the patched files
- Look at the security implication score
- Start with the highest score and move down the list

Patched	Security Implication Score
?LoadJPEGImageNewBuffer@@YGJPAGPAVCMediaType@@PAPAE@Z	6
?OpenTGAFile@@YGJPAXPAPAEPAVCMediaType@@PAE@Z	2
?NonDelegatingQueryInterface@CBasePin@@UAGJABU_GUID@@PAPAX@Z	1

cigital

Binary Diffing: BinDiff Advantages

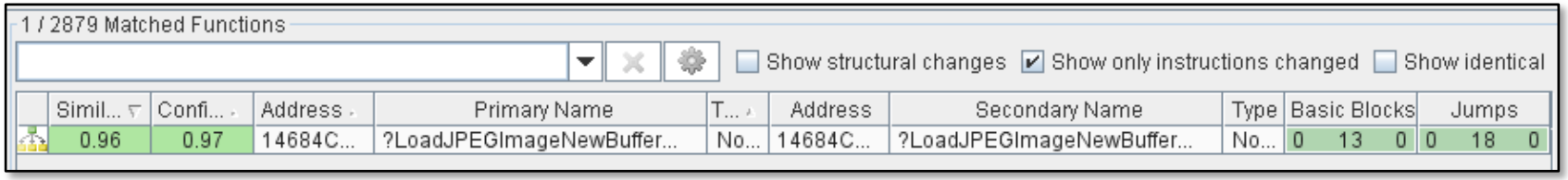
- Filter results based on change properties
- View structural changes


2 / 2879 Matched Functions

Show structural changes | Show only instructions changed | Show identical

	Simil...	Confi...	Address	Primary Name	T...	Address	Secondary Name	Type	Basic Blocks	Jumps
	0.98	0.99	146820E5	?OpenDIBFile@@YGJPAXPAP...	No...	146820D5	?OpenDIBFile@@YGJPAXPAP...	No...	0 44 1	1 66 2
	0.97	0.99	146822...	?OpenTGAFile@@YGJPAXP...	No...	146822D4	?OpenTGAFile@@YGJPAXP...	No...	0 22 1	1 31 2

- View changes to instructions


1 / 2879 Matched Functions

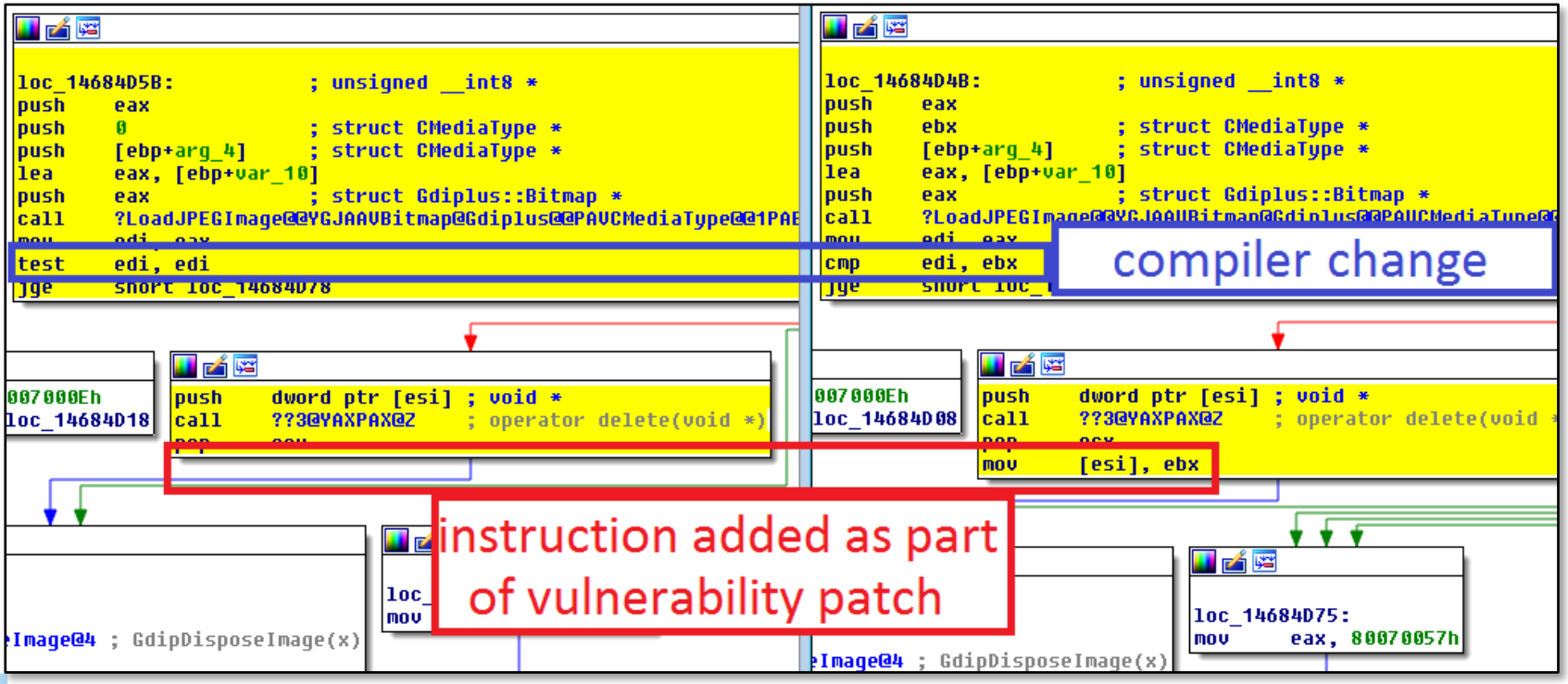
Show structural changes | Show only instructions changed | Show identical

	Simil...	Confi...	Address	Primary Name	T...	Address	Secondary Name	Type	Basic Blocks	Jumps
	0.96	0.97	14684C...	?LoadJPEGImageNewBuffer...	No...	14684C...	?LoadJPEGImageNewBuffer...	No...	0 13 0	0 18 0

cigital

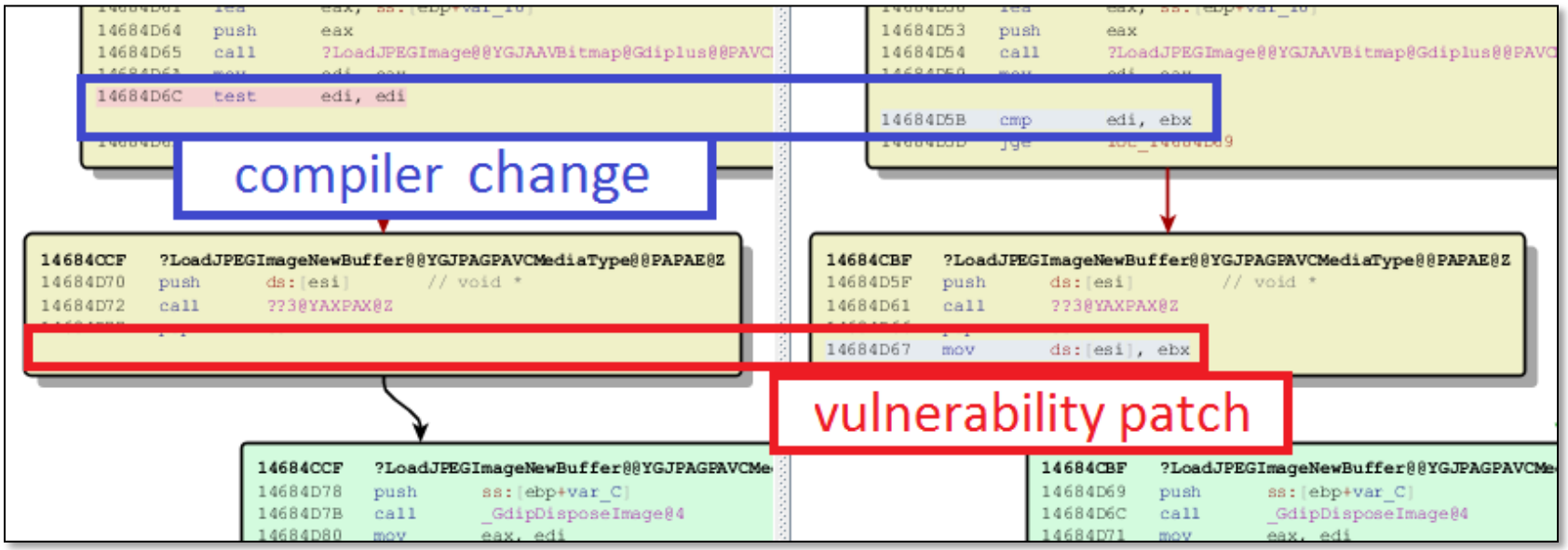
Binary Diffing: DarunGrim Code Comparison

- Instruction-group changes shown in IDA
  - Identify compiler changes vs security patch



Binary Diffing: BinDiff Code Comparison

- BinDiff can show changes without viewing them in IDA Pro
- Shows instruction-level differences



cigital

Binary Diffing: Found the Vulnerability

```
push    dword ptr [esi] ; void *
call    ??3@YAXPAX@Z    ; operator delete(void *)
pop     ecx
```

```
push    dword ptr [esi] ; void *
call    ??3@YAXPAX@Z    ; operator delete(void *)
pop     ecx
mov     [esi], ebx
```

```
14684CCF    ?LoadJPEGImageNewBuffer@@YGJPAGPAVCMediaType@@PAPAE@Z
14684D70    push    ds:[esi]        // void *
14684D72    call    ??3@YAXPAX@Z
14684D77    pop     ecx
```

```
14684CBF    ?LoadJPEGImageNewBuffer@@YGJPAGPAVCMediaType@@PAPAE@Z
14684D5F    push    ds:[esi]        // void *
14684D61    call    ??3@YAXPAX@Z
14684D66    pop     ecx
14684D67    mov     ds:[esi], ebx
```

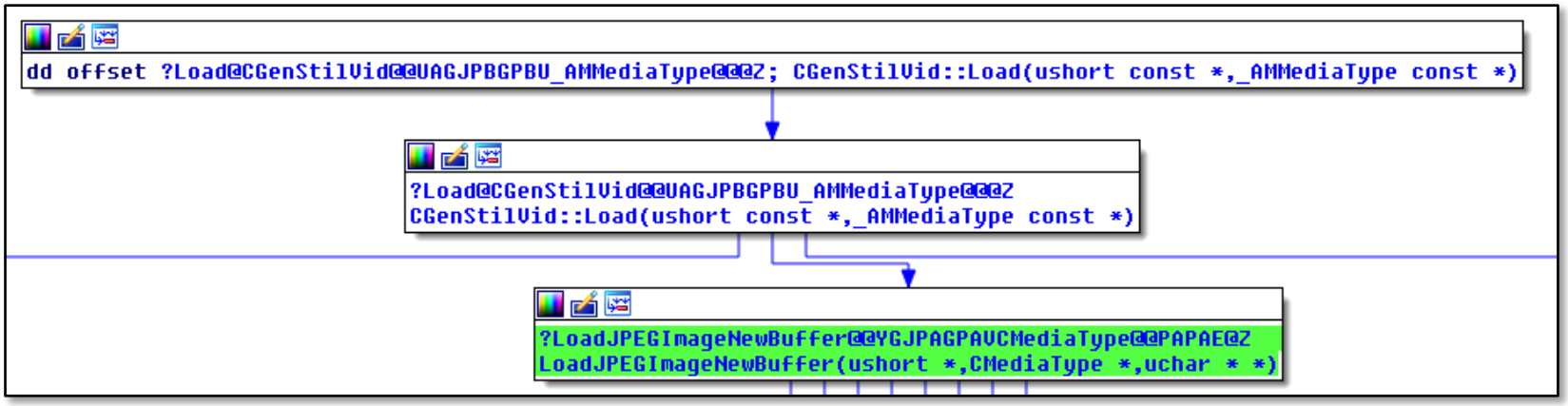
- The patch has been identified
- Now what?

Proof of concept creation: How do you go about making a proof of concept that will trigger the vulnerability?

cigital

Binary Diffing: Triggering the Vulnerability

- Find out how to call the vulnerable function
- Use IDA Pro to find a callable API



- Look at (or near) the root function

cigital

Binary Diffing: Triggering the Vulnerability (cont.)

- If a callable API is found:
  - Craft the API call's arguments carefully in order to trigger the vulnerable condition
- If no callable API is found:
  - Look up the chain of calls, looking for hints as to how to reach the vulnerable condition
- Make a proof of concept file, network packet, API call, etc.

cigital

Binary Diffing: Triggering the Vulnerability (cont.)

- Find lots of samples online
- Set breakpoints in and around the target function
- Run through all the samples to see which ones (if any) come near or actually hit the vulnerable function
- Modify the successful samples to focus in on and trigger the vulnerable code

Binary Diffing: Create a Proof-of-Concept

- Know how to trigger the vulnerability
- Create a simplified PoC
  - Can skip this step if no sample was used to trigger the vulnerability
- Make it **very** easy to use:
  - “Open this file”
  - “Run this executable”
  - “Do this 1 thing”

cigital

Exploit creation: What are some properties of an exploit that works really well? (this has a lot of overlap with well-written software)

cigital

Binary Diffing: Create an Exploit

- Weaponize the proof-of-concept
  - Glossing over lots of details here
  - “Left as an exercise for the reader”
- Make it work on multiple systems under as many configurations as possible
- Make it incredibly easy to use

cigital

Binary Diffing: Create an Exploit (Bonus Points)

- Learn Ruby, if you don't already know it
- Write a Metasploit module for your exploit
  - Metasploit is an exploit framework used by security professionals in penetration tests to exploit vulnerabilities and gain footholds on systems and networks
- Come to Cigital and use the exploit in pentests and red team assessments ;)

References

- Recommended papers/tools:
  - http://www.scribd.com/doc/79441056/EUSecWest-2010-DarunGrim-A-Tool-for-Binary-Diffing-and-Automatic-Vulnerabilities-Pattern-Matching
  - http://www.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-DiffingBinaries-PAPER.pdf
  - http://www.darungrim.org
  - http://www.zynamics.com/bindiff.html

Questions?

cigital

cigital

Thank You

CARTER JONES
SECURITY CONSULTANT
CJONES@CIGITAL.COM

 |