Hotpatching and the Rise of Third-Party Patches

Alexander Sotirov

asotirov@determina.com

BlackHat USA 2006

Overview

In the next one hour, we will cover:

- Third-party security patches
  - recent developments (late 2005 and 2006)
  - why use third-party patches?
  - the issue of trust
- Implementing hotpatching
  - challenges in building a hotpatching system
  - Microsoft hotpatching in Windows 2003
  - dynamic binary translation
- Vulnerability analysis and hotpatch development
  - debugging techniques for vulnerability analysis
  - building a hotpatch from scratch in 15 minutes (demo)

What Is Hotpatching?

Hotpatching is a method for modifying the behavior of an application by modifying its binary code at runtime. It is a common technique with many uses:

- debugging (software breakpoints)
- runtime instrumentation
- hooking Windows API functions
- modifying the execution or adding new functionality to closed-source applications
- deploying software updates without rebooting
- fixing security vulnerabilities

Demo

Old-School DOS Viruses

Part I

Third-Party Security Patches

A Recent Development

Dec 2005

- WMF patch by Ilfak Guilfanov, author of IDA Pro

Mar 2006

- IE createTextRange patch by eEye
- IE createTextRange patch by Determina

WMF Vulnerability

The vulnerability was caused by a feature of the WMF file format. A special type of record (SET_ABORT_PROC) can contain executable code, which is registered as a callback function during WMF processing.

This allows an attacker to execute arbitrary code on a user system. Since it does not rely on memory corruption, the exploit is completely reliable on all versions of Windows.

The vulnerability was already actively exploited and used for malware distribution when it was first made public on Dec 27. It took Microsoft 10 days to release a patch.

Ilfak Guilfanov released an unofficial patch on Dec 31.

Ilfak's WMF Patch

- Injected into all processes with the AppInit_DLLs registry key
- Loads GDI32.DLL and patches it in memory
- Overwrites the function prologue of an exported function with a 5-byte JMP to a hook routine
- The hook routine makes the function return 0 if the second parameter is 9, indicating a SET_ABORT_PROC record
- Requires uninstallation to avoid conflicts with the Microsoft patch

IE createTextRange Vulnerability

The vulnerability was caused by the use of an uninitialized variable in MSHTML.DLL. It can be triggered by calling the JavaScript createTextRange() method on a radio button.

It was reported privately to Microsoft on Feb 13 and independently disclosed on Mar 22. Microsoft did not release patch until Apr 11, almost two months after the initial report.

Two third-party patches were released independently of each other on Mar 27 by eEye and Determina.

eEye's createTextRange Patch

- Creates a patched copy of JSCRIPT.DLL
  - _ loads the original DLL
  - _ uses pattern matching to find the code to patch
  - _ appends the hook routine to an image section with unused space at the end
  - _ the hook routine initializes the uninitialized variable
  - _ writes the patched file as JSCRIPT-EEYE-PATCH20.DLL
- Modifies the registry to force the use of the patched file
- Requires uninstallation to avoid conflicts with the Microsoft patch

Determina's createTextRange Patch

- Injected into all processes with the AppInit_DLLs registry key
- Patches the loader to intercept the loading of MSHTML.DLL
- Contains a list of patch locations for 98 different versions of MSHTML.DLL, generated using our DLL database
- Patch locations are verified with a CRC32 hash
- The patch changes the target of a conditional jump and makes it jump to code that initializes the uninitialized variable. On most versions of MSHTML.DLL this can be accomplished by changing a single byte in memory.

Why Use Third-Party Patches?

Advantages:

- Availability before the official patch
- Provide source code and a detailed explanation of the vulnerability
- Allow you to make your own decisions about risk
- Easy uninstallation

Disadvantages:

- Limited patch QA process
- Limited support for multiple OS versions and languages
- Some vulnerabilities require extensive changes or redesign of the affected application and cannot be hotpatched

The Issue of Trust

Most software vendors have a long record of shipping vulnerable software. If we trust them, there is no reason not to trust a third party patch from a well-known security expert or a security company.

Third-party patches have the following advantages compared to most commercial software:

- Third-party patches are small and self-contained
- Source code is available for review

Third-party patches are ideal for situations where the risk of a system compromise outweighs the risk of interoperability issues.

Part II

Implementing Hotpatching

Designing a Hotpatching System

- Injecting into the process
  - AppInit_DLLs registry key
  - kernel injection
- Intercepting module loading
  - hooking the loader
  - loading the DLL into every process
- Module matching
  - name
  - checksum
  - DLL version
- Locating the patch points
  - hardcoded patch points
  - exported functions
  - pattern matching
- Code modification
  - function table hooking
  - in-place code modification
  - 5-byte JMP overwrite
- Remediation
  - silent
  - detector/protector

Locating the Patch Points

- hardcoded list of patch points
  - _ requires a database of all versions of the DLL
  - _ cannot patch new DLL versions
  - _ most reliable and easy to QA
- exported functions and RPC interfaces
  - _ relies on a well-defined function lookup method
  - _ limited to patching high-level functions
  - _ the APIs rarely change, so this method is reliable
  - _ cannot patch the middle of a function
- pattern matching
  - _ can patch anything
  - _ as long as you can find a good static pattern for it
  - _ not very reliable, you could patch the wrong place

Function Table Hooking

Replacing a function pointer in a table is one of the simplest ways to modify the behavior of a program. This method was commonly used by DOS viruses and memory resident programs to hook system functions by replacing an interrupt handler.

The current targets for hooking on Windows are the IAT table in userspace and the system call table in the kernel.

- Function hooks can be chained
- Hooking on the API level is usually version independent
- Since we're not modifying any code, it is safer than the other approaches

- We can hook only exported functions or system calls
- Modifying the code inside a function is impossible

In-Place Code Modification

Modifying the instructions of the program by overwriting allows us to remove and change the instructions of a program:

- removing code

  overwrite the instructions with NOPs

- changing code

  overwrite an instruction with another instruction of the same or smaller size

```
call check_license                     call check_license
jnz valid                    jmp valid
```

- adding code

  not possible

5-Byte JMP Overwrite

The most common approach to hooking functions on Windows is to overwrite the function prologue with a 5-byte JMP instruction that transfers the execution to our code.

Original code:

```
55                  push    ebp
8B EC               mov     ebp, esp
53                  push    ebx
56                  push    esi
8B 75 08            mov     esi, [ebp+arg_0]
```

Patched code:

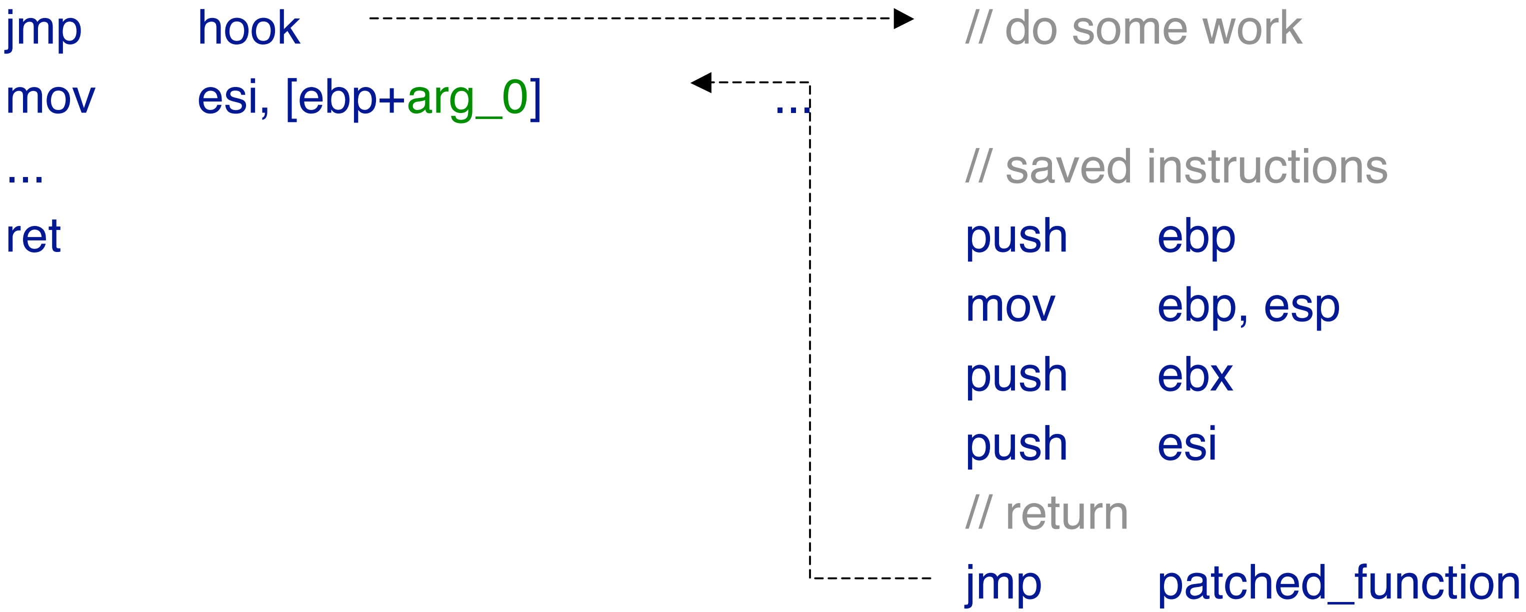
```
E9 6F 02 00 00    jmp hook
8B 75 08            mov     esi, [ebp+arg_0]
```

5-Byte JMP Overwrite

Before overwriting the function prologue, we need to save the overwritten instructions. The hook routine should execute the saved instructions before returning to the patched function.

Patched function:

Hook routine:



```
jmp     hook
mov     esi, [ebp+arg_0]
...
ret
```

```
// do some work
...

// saved instructions
push    ebp
mov     ebp, esp
push    ebx
push    esi
// return
jmp     patched_function
```

5-Byte JMP Overwrite

Using the 5-Byte JMP overwrite technique, we can patch arbitrary locations with the following restrictions:

- No reads or writes to the 5 overwritten bytes
- No jumps or calls targeting the overwritten bytes
- No CALL instructions in the overwritten area, except at the end. If we insert our hook while the code is in the callee, it will return in the middle of the overwritten area.
- No INT instructions in the overwritten area, for the same reason

In most cases, static analysis with IDA is sufficient to determine if a location is suitable for hotpatching.

Remediation

For more sophisticated remediation, we can use a pair of detector/protector functions:

- The detector function checks for a vulnerable condition
  - string longer than X bytes
  - string containing ..\ for a directory traversal attack
  - number of array elements > 0x3FFFFFFF
- The protector takes some action to prevent the exploitation of the vulnerability
  - truncate the long string
  - replace ..\ with underscores
  - force the patched function to abort and return an error

If we disable the protector and run in detect-only mode, we get a very precise IDS system.

Microsoft Hotpatching

The Microsoft hotpatching implementation is described in US patent application 20040107416. It is currently supported only on Windows 2003 SP1, but we'll probably see more of it in Vista.

The hotpatches are generated by an automated tool that compares the original and patched binaries. The functions that have changed are included in a file with a .hp.dll extension. When the hotpatch DLL is loaded in a running process, the first instruction of the vulnerable function is replaced with a jump to the hotpatch.

The /hotpatch compiler option ensures that the first instruction of every function is a `mov edi, edi` instruction that can be safely overwritten by the hotpatch. Older versions of Windows are not compiled with this option and cannot be hotpatched.

Dynamic Binary Translation

A common problem for all of the code modification approaches described earlier is the inability to patch completely arbitrary locations. This can be solved by using a dynamic binary translation engine.

One such engine is the DynamoRIO project from MIT. Its basic model of operation is given below:

- disassemble the current basic block
- copy the instructions into a code cache
- add instrumentation and hotpatches
- execute the basic block from the code cache

This allows us to add, remove and change arbitrary code in the program.

Part III

Vulnerability Analysis and Patch Development

Overview

Developing a hotpatch for a security vulnerability is a four step process:

- Run the exploit
- Use a debugger to find the vulnerable code
- Reverse engineer the vulnerable code
- Write the hotpatch

These steps are very similar to the process of exploit development.

Stack Overflows

- Modify the exploit and overwrite the return address with an invalid address, causing an exception after the jump to the shellcode. Another option is to turn on DEP.
- Overwrite the stack with the minimum amount of data to cause a crash and avoid corrupting the stack frame of the previous function.
- Put a breakpoint in the parent function and trace it until we find the function where the overflow happens.

Heap Overflows

- Run the application in WinDbg with the debugging heap enabled to spot heap corruption early on.
- If the exploit overwrites a fixed location, put a hardware breakpoint on it and see which function writes to it.
- Use conditional breakpoints in WinDbg to display all heap allocations and frees in a suspicious area:

```
bu ntdll!RtlFreeHeap ".printf \"\\nfree(%x, %x)\\n\",
poi(esp+4), poi(esp+c); k 6; g"

bu ntdll!RtlAllocateHeap+113 ".printf \"\\nalloc(%x, %d) =
%x\\n\", poi(esp+4), poi(esp+c), eax; k 6; g"
```

Uninitialized Variables

- If the uninitialized variable is on the stack, find out which stack frame it is in.
- Disassemble the function that created it and look for code that reads or writes to that variable.
- Send a non-exploit request and find the code that initializes the variable. Find out why it was not executed during the exploit request.
- If the variable is on the heap, you have to find who allocated that memory block and what it's used for.

Non-Memory Corruption Vulnerabilities

- Very hard to debug
- Runtime tracing is probably the best option
    - Process Stalker
    - BinNavi
- Wait for the official patch :-)

Demo

Building a Hotpatch From Scratch in 15 Minutes

determina™

Questions?

asotirov@determina.com