YES, NOW
YOU CAN PATCH
THAT VULNERABILITY TOO!

Mitja Kolšek, CEO, ACROS Security & 0patch co-founder
THE STATE OF AFFAIRS

– 15 years of finding and reporting vulnerabilities
– The same types of bugs again and again
– New exploit mitigations, new bypasses
– Few vendors proactively look for vulnerabilities
– Critical security fixes are not being applied
– Feels like being a problem instead of a solution
– Nobody is happy (except the attackers)
VULNERABILITIES

VULNERABILITIES EVERYWHERE
ONE DOES NOT SIMPLY APPLY A VENDOR UPDATE
WHAT IF I TOLD YOU THAT I CAN ALWAYS BYPASS ANTI-MALWARE?
I CAN BREAK INTO ANY CORP NETWORK (BUT I SHOULDN‘T BE ABLE TO)

1. Pick any browser/reader/player vulnerability with a public PoC younger than 2 months
2. Prepare an exploit
3. Mutate the exploit until VirusTotal doesn‘t detect it any more
4. Phish until you‘re in
FOUND A NASTY VULNERABILITY
NOT SURE WHAT TO DO WITH IT
RESEARCHER’S DILEMMA

1. Privately report to vendor
   (and risk anger, silence or lawsuit)

2. Publish
   (and risk anger or lawsuit)

3. Sell it
   (and risk prosecution)

4. Shelve it
   (what’s the point of your work then?)
PATCHING
IS A HARD PROBLEM
A HARD PROBLEM

SOFTWARE VENDORS
- monopoly on patching
- direct and opportunity costs
- deploying fixes is costly
- have better things to do

USERS
- hate downtime
- updating = risk breakage
- not updating = risk ownage

SECURITY RESEARCHERS
- constant conflict with vendors
- considered part of the problem
REINVENTING
SOFTWARE PATCHING
REINVENTING SOFTWARE PATCHING

Take less than a minute to install a small piece of software that will apply tiny security patches in the same way for all applications.

Then apply and remove patches instantly without disturbing users or admins.
DEMO
NO REBOOT, NO RELAUNCH
FUNCTION HOOKING ON STEROIDS

Relocatable instructions (anywhere in the code, not just at the beginning of a function)

We want to inject patch code after this call
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e8</td>
<td>9c</td>
<td>4f</td>
<td>e5</td>
<td>ff</td>
<td>call</td>
</tr>
<tr>
<td>8b</td>
<td>f0</td>
<td>mov</td>
<td>esi,</td>
<td>eax</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>inc</td>
<td>esi</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8d</td>
<td>85</td>
<td>ac</td>
<td>fd</td>
<td>ff</td>
<td>lea</td>
</tr>
<tr>
<td>33</td>
<td>c9</td>
<td>xor</td>
<td>ecx,</td>
<td>ecx</td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>04</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>mov</td>
</tr>
<tr>
<td>e8</td>
<td>63</td>
<td>aa</td>
<td>e4</td>
<td>ff</td>
<td>call</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>push</td>
<td>ebx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8d</td>
<td>85</td>
<td>ac</td>
<td>fd</td>
<td>ff</td>
<td>lea</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>push</td>
<td>eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
mov     esi,eax
inc     esi
lea     eax,[ebp-254h]

CONTINUE:
xor     ecx,ecx
mov     edx,104h
call    00405278h
push    ebx
lea     eax,[ebp-254h]
push    eax

PATCH:
mov     esi,eax
inc     esi
lea     eax,[ebp-254h]
jmp     CONTINUE
DEFINING A PATCH

1. Module hash
2. Offset of the patch inside the module
3. Patch code
PATCH SOURCE CODE

MODULE_PATH "C:\vulnerable_app\app.exe"
PATCH_ID 87235
VULN_ID 993

patchlet_start

  PATCHLET_ID 1
  PATCHLET_OFFSET 0x0000b979
  N_ORIGINALBYTES 5

  code_start
    xor eax, eax
  code_end

patchlet_end
WHAT CAN BE PATCHED

- Unchecked buffers
- Numeric over/underflows
- Use after free
- Double free
- Uninitialized variables
- Format strings
- Binary planting / DLL injection
- Data patching
- (many others)
WHAT CAN’T BE PATCHED
(or not that easily)

- Scripted (to-be-compiled) code
- Design flaws
- Windows kernel (PatchGuard)
- Apps that actively refuse to be patched
MICROSCOPIC CURES FOR BIG SECURITY HOLES
PATCHING DEMO
INTEGER OVERFLOW
array_extra(JSContext *cx, ArrayExtraMode mode, uintN argc, jsval *vp)
{
    JSObject *obj;
    jsuint length, newlen;
    jsval *argv, *elemroot, *invokevp, *sp;
    JSBool ok, cond, hole;
    JSObject *callable, *thisp, *newarr;
    jsint start, end, step, i;
    void *mark;

    obj = JS_THIS_OBJECT(cx, vp);
    if (!obj || !js_GetLengthProperty(cx, obj, &length))
        return JS_FALSE;

    switch (mode) {
    case REDUCE_RIGHT:
        start = length - 1, end = -1, step = -1;
        /* FALL THROUGH */
    }
CVE-2011-2371
Firefox 3.6.16 ReduceRight() Integer Overflow

```html
<html>
  <body>
    <script>
      foo = new Array;
      foo.length = 0x80100000
      foo.reduceRight(function(){}, 1)
    </script>
  </body>
</html>
```
array_extra(JSContext *cx, ArrayExtraMode mode, uintN argc, jsval *vp)
{

    JSObject *obj;
    jsuint length, newlen;
    jsval *argv, *elemroot, *invokevp, *sp;
    JSBool ok, cond, hole;
    JSObject *callable, *thisp, *newarr;
    jsint start, end, step, i;
    void *mark;

    obj = JS_THIS_OBJECT(cx, vp);
    if (!obj || !js_GetLengthProperty(cx, obj, &length))
        return JS_FALSE;

    switch (mode) {
        case REDUCE_RIGHT:
            start = length - 1, end = -1, step = -1;
/* FALL THROUGH */
    }
array_extra (JSContext *cx, ArrayExtraMode mode, uintN argc, jsval *vp) {

    JSObject *obj = JS_THIS_OBJECT(cx, vp);
    if (!obj || !js_GetLengthProperty(cx, obj, &length))
        return JS_FALSE;

    switch (mode) {
    case REDUCE_RIGHT:
        start = length - 1, end = -1, step = -1;
        /* FALL THROUGH */
    }

    void *mark;

    obj = JS_THIS_OBJECT(cx, vp);
    if (!obj || js_GetLengthProperty(cx, obj, &length))
        return JS_FALSE;

    switch (mode) {
    case REDUCE_RIGHT:
        start = length - 1, end = -1, step = -1;
        /* FALL THROUGH */
### CVE-2011-2371
Firefox 3.6.16 ReduceRight() Integer Overflow

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>6b6ab96b</td>
<td>56</td>
</tr>
<tr>
<td>6b6ab96c</td>
<td>8d7c241c</td>
</tr>
<tr>
<td>6b6ab970</td>
<td>894c242c</td>
</tr>
<tr>
<td>6b6ab974</td>
<td>e807240000</td>
</tr>
<tr>
<td>6b6ab979</td>
<td>83c404</td>
</tr>
<tr>
<td>6b6ab97c</td>
<td>85c0</td>
</tr>
<tr>
<td>6b6ab97e</td>
<td>0f84b1ce0a00</td>
</tr>
</tbody>
</table>

- `push esi` 6b6ab96b
- `lea edi,[esp+1Ch]` 6b6ab96c
- `mov dword ptr [esp+2Ch],ecx` 6b6ab970
- `call js_GetLengthProperty` 6b6ab974
- `add esp,4` 6b6ab979
- `test eax,eax` 6b6ab97c
- `je "return JS_FALSE"` 6b6ab97e

**After the call, array length is in dword ptr [edi]**

**Suitable bytes for overwriting**
### CVE-2011-2371
Firefox 3.6.16 ReduceRight() Integer Overflow

<table>
<thead>
<tr>
<th>Address</th>
<th>Mismatched Instructions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6b6ab96b</td>
<td>push esi</td>
<td>push <code>esi</code> to the stack</td>
</tr>
<tr>
<td>6b6ab96c</td>
<td>lea edi,[esp+1Ch]</td>
<td>load <code>edi</code> with <code>esp+1Ch</code></td>
</tr>
<tr>
<td>6b6ab970</td>
<td>mov dword ptr [esp+2Ch],ecx</td>
<td>move <code>ecx</code> to the memory location [esp+2Ch]</td>
</tr>
<tr>
<td>6b6ab974</td>
<td>call js_GetLengthProperty</td>
<td>call function <code>js_GetLengthProperty</code></td>
</tr>
<tr>
<td>6b6ab979</td>
<td>and dword ptr [edi],7FFFFFFFh</td>
<td>AND <code>edi</code> with 7FFFFFFFh</td>
</tr>
<tr>
<td>6b6ab979</td>
<td>add esp,4</td>
<td>add <code>esp</code> by 4 bytes</td>
</tr>
<tr>
<td>6b6ab97c</td>
<td>test eax,eax</td>
<td>TEST <code>eax</code> with itself</td>
</tr>
<tr>
<td>6b6ab97e</td>
<td>je &quot;return JS_FALSE&quot;</td>
<td>Jump if <code>eax</code> equals 0 to &quot;return JS_FALSE&quot;</td>
</tr>
</tbody>
</table>

We reset the top bit and keep the length below MAX_INT

Relative offset from start of module js3250.dll = B979h
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6b6ab96b</td>
<td>push esi</td>
</tr>
<tr>
<td>6b6ab96c</td>
<td>lea edi,[esp+1Ch]</td>
</tr>
<tr>
<td>6b6ab970</td>
<td>mov dword ptr [esp+2Ch],ecx</td>
</tr>
<tr>
<td>6b6ab974</td>
<td>call js_GetLengthProperty</td>
</tr>
<tr>
<td>6b6ab979</td>
<td>cmp dword ptr [edi],7FFFFFFFh</td>
</tr>
<tr>
<td></td>
<td>jbe DONE</td>
</tr>
<tr>
<td></td>
<td>and dword ptr [edi],7FFFFFFFh</td>
</tr>
<tr>
<td></td>
<td>call PIT_ExploitBlocked</td>
</tr>
<tr>
<td>DONE:</td>
<td></td>
</tr>
<tr>
<td>6b6ab979</td>
<td>add esp,4</td>
</tr>
<tr>
<td>6b6ab97c</td>
<td>test eax,eax</td>
</tr>
<tr>
<td>6b6ab97e</td>
<td>je &quot;return JS_FALSE&quot;</td>
</tr>
</tbody>
</table>

We want to display a warning to the user
PATCHING DEMO
BUFFER OVERFLOW
CVE-2013-7409
AllPlayer 5.8 Buffer Overflow In .M3U File
# CVE-2013-7409

## AllPlayer 5.8 Buffer Overflow In .M3U File

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>MIPS</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>005ba7fa</td>
<td>05 03</td>
<td>push ebx</td>
<td>ebx points to source buffer (line)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; ebx points to source buffer (line)</td>
<td></td>
</tr>
<tr>
<td>005ba7fb</td>
<td>05 89 0c 4f</td>
<td>call kernel32!lstrlenW</td>
<td>; eax is the length of the line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; eax is the length of the line</td>
<td></td>
</tr>
<tr>
<td>005ba800</td>
<td>05 8b 0f</td>
<td>mov esi,eax</td>
<td>; esi is the length of the line + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; esi is the length of the line + 1</td>
<td></td>
</tr>
<tr>
<td>005ba802</td>
<td>05 8d 0a c5</td>
<td>lea eax,[ebp-254h]</td>
<td>; eax points to destination buffer</td>
</tr>
<tr>
<td>005ba803</td>
<td>05 8d 0a c5</td>
<td>lea eax,[ebp-254h]</td>
<td>; eax points to destination buffer</td>
</tr>
<tr>
<td>005ba809</td>
<td>05 33 c9</td>
<td>xor ecx,ecx</td>
<td>; which only has 104h bytes on stack</td>
</tr>
<tr>
<td>005ba80b</td>
<td>05 ba 04 01</td>
<td>mov edx,104h</td>
<td>; which only has 104h bytes on stack</td>
</tr>
<tr>
<td>005ba810</td>
<td>05 e8 6a 0f</td>
<td>call zero-ize_destination_buffer</td>
<td></td>
</tr>
<tr>
<td>005ba815</td>
<td>05 05 03</td>
<td>push ebx</td>
<td>; ebx points to source buffer (line)</td>
</tr>
<tr>
<td>005ba816</td>
<td>05 8d 0a c5</td>
<td>lea eax,[ebp-254h]</td>
<td>; eax points to destination buffer</td>
</tr>
<tr>
<td>005ba81c</td>
<td>05 05 03</td>
<td>push eax</td>
<td>; which only has 104h bytes on stack</td>
</tr>
<tr>
<td>005ba81d</td>
<td>05 e8 62 0f</td>
<td>call kernel32!lstrcpyW</td>
<td></td>
</tr>
</tbody>
</table>

**Suitable bytes for overwriting**

**We want to shorten source buffer before this call**
CVE-2013-7409
AllPlayer 5.8 Buffer Overflow In .M3U File

005ba815 53    push   ebx
                ; ebx points to source buffer (line)
005ba816 8d85acfdffff    lea    eax,[ebp-254h]
                ; eax points to destination buffer
                ; which only has 104h bytes on stack
005ba81c 50    push   eax
005ba81d e8624fe5ff    call   kernel32!lstrcpyW
CVE-2013-7409
AllPlayer 5.8 Buffer Overflow In .M3U File

005ba815 53  push ebx
; ebx points to source buffer (line)

005ba816 8d85acfdffff  lea eax,[ebp-254h]
; eax points to destination buffer
; which only has 104h bytes on stack

DONE:
005ba816 cmp esi,104h  ; esi is line length + 1
jbe DONE
mov word ptr [ebx+208h],0
call PIT_ExploitBlocked

We cut the source buffer short by terminating it with a 0

Relative offset from start of module AllPlayer.exe = 1ba816h
GUIDELINES
HOW TO FIX WITHOUT BREAKING
PATCHING GUIDELINES

1. Find a good place for patching
2. Don’t break anything
3. Change as little code as possible
4. Execute as rarely as possible
5. Test security and functionality
PATCHING GUIDELINES

Find a good place for patching

1. Cover all vulnerable execution paths (but ideally nothing else)
2. Relocated original code must not be a target of calls or jumps
3. Relocated original code must be easily relocatable (nothing that uses relative offsets)
PATCHING GUIDELINES

Don’t break anything

2. Make no assumptions about how your patched code can be reached (instead, make sure using disassemblers, code analysis tools)
   - Preserve functionality (don’t cut off vulnerable code unless that’s the only possible solution)
   - Make sure there are no side-effects (changed registries, changed flags)
PATCHING GUIDELINES
Change as little code as possible

- Less code = fewer errors
- Less code = easier reviewing and testing
- Less code = less execution overhead
- Less code = less chance of race condition
PATCHING GUIDELINES

Execute as rarely as possible

- Avoid patching inside loops
- Sanitize user input at the beginning of an execution tree to cover all branches
PATCHING GUIDELINES

Test security and functionality

5

- PoCs and exploits should be blocked
- Legitimate use cases of the patched functionality should have exactly the same behavior as before
FOUND A NASTY VULNERABILITY
HECK, I'LL OPATCH IT
VULNERABILITIES

VULNERABILITIES EVERYWHERE
OPATCHES
OPATCHES EVERYWHERE
BETA ACCOUNTS
SEND YOUR EMAIL AND WE’LL CREATE AN ACCOUNT FOR YOU

support@0patch.com
PATCH

0patch.com
@0patch

THANK YOU!

Mitja Kolšek, CEO, ACROS Security & 0patch co-founder
mitja.kolsek@acrossecurity.com