Dissecting Wireshark
A Case Study on Network Anti-Forensics

Ben Schmidt (Presenter) // @_supernothing
Lord Commander of Security Research @NarfIndustries

Paul Makowski // @myhndl
Director of World Domination @NarfIndustries
this talk

• background & why you should care
• life of a packet
• dissector overview
• packet reconstruction example
• some lame DoS & how to find your own
• RCE affecting default heuristic dissector
• mitigations & recommendations
background // why you should care

• Wireshark is **the** packet analysis tool
• used everywhere
• breaking it lets you:
  ○ hide traffic
  ○ hinder analysis
  ○ hack analysts
  ○ hurdle airgaps
• what alternatives are there?
background // Samurai F.U.D. Team

- Fear, Uncertainty and Doubt
- started at DEFCON 20
- led by Paul Makowski & Ben Schmidt
- task: protect legit ‘sploits & implants
- solution: pop Wireshark & related tools
spreading F.U.D. to workstations
spreading F.U.D. to workstations
spreading F.U.D. to passive sniffers
1. user asks [libp|WinP]cap to listen and tells it what to filter out
   a. Linux: a BPF is JITted in kernelspace
   b. Windows: filter installed in kernelspace (npf.sys)
2. packets pass monitor network interface
3. kernel passes packet to [libp|WinP]cap
4. [libp|WinP]cap ignores or duplicates packet to listening user process
5. Wireshark reconstructs traffic from lowest to highest layer
   a. at each layer, Wireshark determines which dissector is responsible for handling the data
   b. higher-level candidates dependant on lower-level decisions
   c. dissectors tell Wireshark what data they care about... and can be pretty promiscuous
dissectors

(P)IDL

ASN.1

The C Programming Language
dissectors

“Note: **This is currently broken** (see commits 49066, 49138).

http://wiki.wireshark.org/Python

“...make it harder to enable Python: change it from "--with-python" to "--with-broken-python" just to prevent people from enabling unless they really mean it (are going to work on fixing it).”

https://anonsvn.wireshark.org/viewvc?revision=49138&view=revision
“Although it's possible to write dissectors in Lua, Wireshark dissectors are written in C, as C is several times faster than Lua. Lua is ok for prototyping dissectors, during Reverse Engineering you can use your time for finding out how things work instead of compiling and debugging your C dissector.”

http://wiki.wireshark.org/Lua
dissectors

THE C PROGRAMMING LANGUAGE

http://www.phrack.org/
example traffic reconstruction
0000 00 0c 29 98 30 31 00 50 56 ea 1f 28 08 00 45 00 ..).01.PV..(E.
0010 01 20 ff fd 00 00 80 06 17 31 de ad be ef de ad ........1.my..P.....e&$Y....P.
0020 be ef 00 50 c0 1b ef 65 26 24 59 8d f2 f6 50 18 ..P...e&$Y...P.
0030 fa f0 e2 17 00 00 48 54 54 50 2f 31 2e 31 20 32 HTTP/1.1 2
0040 30 34 20 4e 6f 20 43 6f 6e 74 65 6e 74 0d 0a 30 04 No Content..Content-Length: 0
0050 0d 0a 44 61 74 65 3a 20 57 65 64 2c 20 30 32 20 ..Date: Wed, 02
0060 4a 75 6l 20 32 30 31 34 20 32 32 3a 33 32 3a 31 Jul 2014 22:32:1
0070 33 20 47 4d 54 0d 0a 43 6f 6e 6e 65 63 74 69 6f 3 GMT..Connectio
0080 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 50 n: keep-alive..P
0090 72 61 67 6d 61 3a 20 6e 6f 2d 63 61 63 68 65 0d ragma: no-cache.
00a0 0a 45 78 70 69 72 65 73 3a 20 4d 6f 6e 2c 20 30 Expires: Mon, 0
00b0 31 20 4a 61 6e 20 31 39 39 30 20 30 30 3a 30 30 1 Jan 1990 00:00
00c0 3a 30 20 47 4d 54 0d 0a 43 68 65 2d 43 6f 6e 72 00 GMT..Cache-C
00d0 6f 6c 3a 20 70 72 6f 78 79 2d 72 65 76 61 6c 69 0 control: private,
00e0 64 61 74 65 0d 0a 0d 0a no-cache, no-ca..revalidate....
/epan/dissectors/packet-eth.c:

```c
void proto_register_eth(void)
{
    ...
    register_heur_dissector_list("eth", &heur_subdissector_list);
    ...
    register_dissector("eth_withoutfcs", dissect_eth_withoutfcs, proto_eth);
}
```

The link layer is responsible for the basic networking functions such as address resolution, error checking, and data flow over the physical medium. The code snippet shows the registration of dissectors for the Ethernet protocol.
link layer

/app/.../dissectors/packet-eth.c:
(dissect_eth_withoutfcs()) passes through to dissect_eth_common()}

static proto_tree *
dissect_eth_common(tvbuff_t *tvb, packet_info *pinfo, proto_tree *parent_tree,
int fcs_len)
{
    ...
    if (dissector_try_heuristic(heur_subdissector_list, tvb, pinfo,
        parent_tree, &hdtbl_entry, NULL))
        ...
        call_dissector_with_data(ethertype_handle, tvb, pinfo, parent_tree,
            &ethertype_data);
    ...
}
link layer

/epan/dissectors/packet-ethertype.c:

void proto_register_ethertype(void)
{
    ...
    new_register_dissector("ethertype", dissect_ethertype, proto_ethertype);
    ...
    ethertype_dissector_table = register_dissector_table("ethertype",
        "Ethertype", FT_UINT16, BASE_HEX);
    ...
}
/epan/dissectors/packet-ethertype.c:

```c
static int
dissect_ethertype(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree, void *
data)
{
    ...
    next_tvb = tvb_new_subset(tvb, ethertype_data->offset_after_ethertype,
        captured_length, reported_length);
    ...
    dissector_found = dissector_try_uint(ethertype_dissector_table,
        ethertype_data->etype, next_tvb, pinfo, tree);
    ...
}
```

0000 00 0c 29 98 30 31 00 50 56 ea 1f 28 08 00 45 00 ..).01.PV..(..E.
0010 01 20 ff fd 00 00 80 06 17 31 de ad be ef de ad .........1..my..
0020 be ef 00 50 c0 1b ef 65 26 24 59 8d f2 f6 50 18 ...P...e&$Y...P.
0030 fa f0 e2 17 00 00 48 54 54 50 2f 31 2e 31 20 32 ....HTTP/1.1 2
network layer

/app
transport
network
link

/net/pan/dissectors/packet-ip.c:

void
proto_register_ip(void)
{
    ...
    ip_dissector_table = register_dissector_table("ip.proto", "IP protocol",
                                    FT_UINT8, BASE_DEC);
    ...
    register_dissector("ip", dissect_ip, proto_ip);
    ...
}
network layer

/epan/dissectors/packet-ip.c:

void proto_reg_handoff_ip(void) {
    ...
    ip_handle = find_dissector("ip");
    ...
    dissector_add_uint("ethertype", ETHertype_IP, ip_handle);
    ...
}
network layer

/epan/dissectors/packet-ip.c:

static void
dissect_ip(tvbuff_t *tvb, packet_info *pinfo, proto_tree *parent_tree)
{
    ...
    dissector_try_uint_new(ip_dissector_table, nxt, next tvb, pinfo,
                           parent_tree, TRUE, iph)
    ...
}
The transport layer handles the communication between applications and the network layer. The transport layer provides a reliable communication service to the applications. In the network packet shown, the transport layer protocol is indicated by the "tcp" protocol identifier. The code snippet from `packet-tcp.c` demonstrates the `proto_register_tcp` function, which registers the transport layer protocol dissector. This function calls `register_dissector` to register the transport layer protocol dissector with the protocol identifier "tcp".
/epan/dissectors/packet-tcp.c:

```c
void
proto_reg_handoff_tcp(void)
{
    ...
    tcp_handle = find_dissector("tcp");
    dissector_add_uint("ip.proto", IP_PROT_TCP, tcp_handle);
    ...
}
```

transport layer
/epan/dissectors/packet-tcp.c:

callchain:
\textbf{dissect\_tcp()} calls into:
\hline
\textbf{dissect\_tcp\_payload()} calls into:
\hline
\textbf{process\_tcp\_payload()} calls into:
\hline
\textbf{decode\_tcp\_ports()} calls into:
\hline
\textbf{dissector\_try\_heuristic()}\hline

dissector\_try\_heuristic() iterates over the callbacks registered against the “tcp” heuristic dissector list.

HTTP registers one such callback...

transport layer
how many of these are you legitimately running on your network?
HTTP/1.1 204 No Content.

Date: Wed, 02 Jul 2014 22:32:10 GMT

Connection: keep-alive

Expires: Mon, 01 Jan 1990 00:00:00 GMT

Cache-Control: private, no-cache, no-store, proxy-revalidate
void proto_reg_handoff_message_http(void)
{
    ...
    heur_dissector_add("tcp", dissect_http_heur_tcp, proto_http);
    ...
}
application layer

/epan/dissectors/packet-http.c:

static gboolean
dissect_http_heur_tcp(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree, void *data)
{
...  
  if((tvb_strncaseeql(tvb, linelen-8, "HTTP/1.1", 8) == 0)|| (tvb_strncaseeql(tvb, 0, "HTTP/1.1", 8) == 0)){
  ...
    dissect_http(tvb, pinfo, tree, data);
    return TRUE;
  }
...  
}
lame DoS 0days #1-12
lame DoS 0days #1–12

- AKA revenge of the decompression bomb
- `grep -r "tvb_child_uncompress" .`
- at least 12 dissectors affected
  - some of these are heuristic (!)
- limited to 65535 bytes in some cases
  - solutions:
    - attack protocol doing TCP session reassembly
    - send more packets
lame DoS 0day // demo

- packet-http.c
- packet-gadugadu.c
- you get the idea...
less-lame DoS 0days #13–???

- find loops controlled by user length
  - tvb_get_ntohl/letohl is usually a good sign
- make sure offset isn’t incremented
  - or can be controlled (offset += user_data)
- ensure there aren’t any hidden checks
- most Defcon 20 bugs were this type
- slightly less lame than prior bugs
less-lame DoS // MongoDB

● infinite loop in packet-mongo.c, <= v1.8.1
● found with:
  
grep -r -B8 -e 'for.
  
  
  
  

● packet-mongo.c registers interest in TCP port 27017
● if you can send TCP traffic on 27017 past a listener: profit
less-lame DoS // MongoDB

/epan/dissectors/packet-mongo.c:
(multiple functions)
while(offset < tvb_reported_length(tvb)) {
    offset += dissect_bson_document(tvb, pinfo, offset, tree, hf_mongo_document, 1);
}

...what if dissect_bson_document() return 0?

/epan/dissectors/packet-mongo.c:
static int
dissect_bson_document(tvbuff_t *tvb, packet_info *pinfo, guint offset, proto_tree *tree, int hf_mongo_doc, int nest_level)
{
    ...
    document_length = tvb_get_letohl(tvb, offset);
    ...
    return document_length;
}
bug #??? // want more?

- [https://www.wireshark.org/download/automated/captures/](https://www.wireshark.org/download/automated/captures/)
  - check back early and often
  - usually at least one current
- [https://bugs.wireshark.org/bugzilla/](https://bugs.wireshark.org/bugzilla/)
  - search for “crash” and/or “hang”
  - mirror all the capture files that have been uploaded
- get more pcaps and use Wireshark’s fuzzing framework
/epan/dissectors/packet-rtps.c @ rtps_util_add_typecode():

guint32 size[MAX_ARRAY_DIMENSION]; /* Max dimensions */
dim_max = NEXT_guint32(tvb, offset, little_endian);
offset += 4;
for (i = 0; i < MAX_ARRAY_DIMENSION; ++i)
    size[i] = 0;
for (i = 0; i < dim_max; ++i) {
    size[i] = NEXT_guint32(tvb, offset, little_endian);
    offset += 4;
}
// :D :D :D :D :D :D :D :D :D :D
exploit mitigations // Windows

DEP is on:(

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Path</th>
<th>Company Name</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>svhost.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TrustedInstaller.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsass.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsm.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csrss.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>winlogon.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explorer.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmtoolsd.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SnippingTool.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wireshark.exe</td>
<td>Wireshark</td>
<td>C:\Program Files\Wireshark\Wireshark.exe</td>
<td>The Wireshark developer...</td>
<td>DEP (permanent)</td>
</tr>
<tr>
<td>procexp.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet.dll</td>
<td>packet.dll (Vista) Dynamic Link Library</td>
<td>C:\Windows\System32\Packet.dll</td>
<td>Riverbed Technology, Inc.</td>
<td></td>
</tr>
<tr>
<td>wpcap.dll</td>
<td>wpcap.dll Dynamic Link Library - b...</td>
<td>C:\Windows\System32\wpcap.dll</td>
<td>Riverbed Technology, Inc.</td>
<td></td>
</tr>
<tr>
<td>Wireshark.exe</td>
<td>Wireshark</td>
<td>C:\Program Files\Wireshark\Wireshark.exe</td>
<td>The Wireshark developer...</td>
<td>ASLR</td>
</tr>
<tr>
<td>SensApi.dll</td>
<td>SENS Connectivity API DLL</td>
<td>C:\Windows\System32\SensApi.dll</td>
<td>Microsoft Corporation</td>
<td>ASLR</td>
</tr>
<tr>
<td>wimaxmacphy.dll</td>
<td>wimaxmacphy dissector</td>
<td>C:\Program Files\Wireshark\plugins\1.10.8\wimaxmacphy.dll</td>
<td>The Wireshark developer...</td>
<td>ASLR</td>
</tr>
<tr>
<td>wimaxasncp.dll</td>
<td>wimaxasncp dissector</td>
<td>C:\Program Files\Wireshark\plugins\1.10.8\wimaxasncp.dll</td>
<td>The Wireshark developer...</td>
<td>ASLR</td>
</tr>
<tr>
<td>wimax.dll</td>
<td>m2m dissector</td>
<td>C:\Program Files\Wireshark\plugins\1.10.8\wimax.dll</td>
<td>Intel Corporation</td>
<td>ASLR</td>
</tr>
</tbody>
</table>
exploit mitigations // Windows

but no ASLR on WinPcap modules :) :)
exploit mitigations // Windows

/GS enabled :

```
6a06ff4d 8b4df0    mov    ecx.dword ptr [ebp-10h]
6a06ff50 33cd    xor    ecx,ebp
6a06ff52 e8af79400    call    libwireshark!__security_check_cookie (6a9bf700)
```

…but variable reordering wasn’t triggered :) :) :)
super 1337 RCE // exploitation

Overwrite dim_size and i with the right values...
super 1337 RCE // wat
super 1337 RCE // exploitation

- let's recap: we've got
  - ASLR defeat
  - EIP & ESP
  - what else could we possibly need?
- attack different versions
  - vuln goes back to v1.03(!), released in Sept 2008
- stack space...later versions aren’t nice.
- solution: we have to go deeper
super 1337 RCE // exploitation

we can recurse before we crash!

/epan/dissectors/packet-rtps.c @ rtps_util_add_typecode():
case RTI_CDR_TK_ARRAY: {

    ...  
    /* Recursive decode seq typecode */
    /*offset += */ rtps_util_add_typecode(tree, tvb, offset, 
    little_endian, indent_level, is_pointer, 
    bitfield, is_key, offset_begin, name, -1, size, 
    ndds_40_hack);

    ...  
}
super 1337 RCE // exploitation

- benefits of recursing are many
  - fix up stack frames
  - perform differently with different stack layouts
  - gain more stack space to work with
  - support multiple versions in single packet
- downside: annoying to write and test
- let’s start with frame differences...
super 1337 RCE // exploitation

- v1.10 changed stack layout from v1.8
- i (and other vars) at different offsets
- RA & SP at different offsets
  - v1.8 has slightly more stack space
- strategy:
  - do v1.10 overflow first, then recurse
  - do v1.8 overflow second, and return
  - if no code exec yet, we’ll return again
super 1337 RCE // exploitation

- works good in theory
- tvb pointer stored after stored RA
  - greatly reduced stack space :(
- recurse three times just for v1.10
  - once to copy shellcode to stack
  - once to set up an add ESP, 8 pivot
  - once to copy ROP chain above pivot
- chaining together with v1.8 is hard.
## super 1337 RCE // exploitation

<table>
<thead>
<tr>
<th>v1.10.1</th>
<th>v1.8.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shared Preamble</strong></td>
<td><strong>First vulnerable frame - desync occurs</strong></td>
</tr>
<tr>
<td>Copy in shellcode</td>
<td>Skip frame by long copy</td>
</tr>
<tr>
<td>Set up stack pivot</td>
<td>Skip frame by long copy</td>
</tr>
<tr>
<td>Copy ROP</td>
<td>Copy in shellcode</td>
</tr>
<tr>
<td>Encounters i=0 and returns</td>
<td>Set up stack pivot</td>
</tr>
<tr>
<td>N/A</td>
<td>Copy ROP and return</td>
</tr>
</tbody>
</table>
super 1337 RCE // demo

- v1.8
- v1.10
mitigations

• for users
  ○ disable dissectors you don’t need

• for devs
  ○ seccomp() / seccomp_bpf()
  ○ actual privilege separation (separate processes)
  ○ better killing of long-running dissectors
  ○ more hardened memory-management functions
  ○ more official/automated code review
  ○ don’t handle jumping to 0x41414141?
thanks

- Samurai
- Wireshark devs
- thesaurus.com
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>243</td>
<td>0.049237</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>244</td>
<td>0.049423</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>245</td>
<td>0.049604</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>246</td>
<td>0.049841</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>247</td>
<td>0.050037</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>248</td>
<td>0.050232</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>249</td>
<td>0.050428</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>250</td>
<td>0.050626</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>251</td>
<td>0.050826</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>252</td>
<td>0.051038</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>253</td>
<td>0.051224</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
<tr>
<td>254</td>
<td>0.051402</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>40</td>
<td>[TCP Ret]</td>
</tr>
</tbody>
</table>

Any questions?