Sharingan: A Ninja Art to Copy, Analyze and Counter Attack

[Automated Fuzzer Generation using Heuristic based Analysis of Network Protocols]

Mrityunjay Gautam (Author and Presenter)
B. Tharaka Ramu (Author)

Product Security Group,
Symantec, Pune, India.
Motivation

• Network protocols
  – Legacy
  – Little to no documentation
  – Vulnerable to new and sophisticated attacks

• Writing Fuzzers require knowledge of packet structure

• Traditional Approach
  – Manual Analysis of Protocols using Sniffers
  – Source Code Review [*If you have it...*]
  – Blind Fuzzing
Problem Statement

Automate Analysis of Protocols and Fuzzer Generation
Network Vulnerabilities

CVE-2002-4248
Summary: Buffer overflow in the RTSPProtocol::HandleGetParameterRequest function in client/core/rtspprotocol.cpp in RealNetworks RealPlayer 10, RealPlayer 10.9.5.0.12.1040 through 10.9.5.0.12.1741, RealPlayer 11 110.0 through 110.4, RealPlayer Enterprise, Mac RealPlayer 10 and 10.1, Linux RealPlayer 10, and Mozilla 1.0 allows remote attackers to cause a denial of service (application crash) or possibly execute arbitrary code via a crafted RTSP SET_PARAMETER request.
Published: 01/25/2010
CVSS Severity: 9.2 (HIGH)
CVE-2002-4247
Summary: Stack-based buffer overflow in protocol/rtsp/rtspint.cpp in RealNetworks RealPlayer 10; RealPlayer 10.9.5.0.12.1040 through 10.9.5.0.12.1741; RealPlayer 11 110.0.x; RealPlayer SP 1.0.0 and 1.0.1; RealPlayer Enterprise; Mac RealPlayer 10, 10.1, 110.0, and 110.1; Linux RealPlayer 10, 110.0, and 110.1; and Mozilla Player 10.x, 110.0, and 110.1 allows remote attackers to cause a denial of service (application crash) or possibly execute arbitrary code via an ASIM rulebook with a large number of rules, related to an "array overflow."
Published: 01/28/2010
CVSS Severity: 9.2 (HIGH)
CVE-2010-0138
Summary: Buffer overflow in CiscoWorks Internetwork Performance Monitor (IPM) 2.0 and earlier on Windows, as distributed in CiscoWorks LAN Management Solution (LMS), allows remote attackers to execute arbitrary code via a malformed getProcName CORBA General Inter-ORB Protocol (GIOP) request, related to a "third-party component," aka Bug ID CSCct62350.
Published: 01/21/2010
CVSS Severity: 10.0 (HIGH)
CVE-2002-4420
Summary: Buffer overflow in the bsd daemon in F5 Networks BIG-IP Application Security Manager (ASM) 9.4.4 through 9.4.7 and 10.0.0 through 10.0.1, and Protocol Security Manager (PSM) 9.4.4 through 9.4.7 and 10.0.0 through 10.0.1, allows remote attackers to cause a denial of service (crash) via unknown vectors. NOTE: some of these details are obtained from third party information.
Published: 12/24/2009
CVSS Severity: 7.8 (HIGH)
CVE-2008-3677
Summary: The Internet Authentication Service (IAS) in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP2, Vista Gold and SP1, and Server 2008 Gold does not properly verify the credentials in an MS-CHAP v2 Protected Extensible Authentication Protocol (PEAP) authentication request, which allows remote attackers to access network resources via a malformed request, aka "MS-CHAP Authentication Bypass Vulnerability."
Published: 12/05/2009
CVSS Severity: 10.0 (HIGH)
ART-2009-0568
Summary: The SSL protocol, and the SSL protocol 2.0 and possibly earlier, as used in Microsoft Internet Information Services (IIS) 7.0, mod_ssl in the Apache HTTP Server 2.2.14 and earlier, OpenSSL 0.9.8, 0.9.9, and earlier, Mozilla Network Security Services (NSS) 3.12.4 and earlier, multiple Cisco products, and other products, does not properly associate renegotiation handshakes with an existing connection, which allows man-in-the-middle attackers to insert data into HTTPS sessions, and possibly other types of sessions protected by TLS or SSL, by sending an unauthenticated request that is processed.

Sharingan

Symantec
Time Complexity Analysis

• Assuming average experience in manual protocol analysis.
• Per packet – 20 mins to 30 mins.
• A simple proprietary protocol has 7 to 10 packet types
• 8 to 10 samples of each type => ~ 100 packets to analyze
• Continuous work of 20 to 25 hours.
• “Human” time frame – 4 working days
• Writing fuzzer – 1 working day
• Total 5 working days = 1 week.
Fuzzers and Fuzzing Constructs

• Target Fuzzing Frameworks – SPIKE & Peach
• See Reference for more details.
• Typical Fuzzing Constructs
  – Blocks
  – Length Relations
  – Human readable Strings
  – Delimiters
Automated Protocol Analyzers – Existing Tools

• Generic – GAPA, binpac
• Hardcoded RFC – Wireshark
• Bio-Informatic Algorithms – Protocol Informatics (PI), ScriptGen
• Clustering Algorithms – Prospex, Discoverer
• Monitoring Server Program – Prospex
Our Solution -- Sharingan

Courtesy: http://files.myopera.com/mxforce/blog/Mangekyou%20Sharingan.jpg
Proposed Approach

• Automated Analysis of Network Protocols
• No prior knowledge base – Heuristic + AI
Wireshark to XML

- Use of Jpcap library
- Use Conversion.jar to convert to XML
2-XOR Heuristic

- Pattern matching heuristic to identify common prefix.
- Input Bit Patterns – $P_1 \ P_2 \ P_3 \ldots \ P_n$
- $R_i = P_1 \text{xor} \ P_{i+1}$
- $R = R_1 \text{xor} \ R_2 \text{xor} \ldots \ R_{i-1}$
- $\text{Min}\{\text{Length of ‘0’ prefix in } R, \text{Length of ‘0’ prefix in } R_1\}$
Rearrange Module

- Remove repetitions by structural match with previous packets in the Base Dataset.

- If new packet type,
  - Identification of base packet (lowest # of groups)
  - Structural Match using **2XOR Heuristic** with ALL packets in ALL sessions
  - Tabulate the percentage match
  - Statistical **Mode** of frequency picked up for structural analysis
  - Find the first match of above structural type in ALL sessions
  - Store in Adjacent session files

- Discard all packets which do not correspond to ANY packet in Base Dataset.
Magic String Identification

- Use 2-XOR Heuristic
- Mark the Magic String as CONST component
- If \((\text{Length(Magic String)} > N)\), analyze for patterns which can be fuzzed.
- ‘N’ is a configurable parameter.
Identifying Human Understandable Strings

- Character Classes: [A-Za-z@._]+ [0-9.]+ [^]+
- Tokenization Algorithm
  - Start from current unclassified character
  - Continue until the next character is in different character class
  - If (Token Length > K && Token Class != [^]+) mark Token as [P] else mark Token as [NP]
  - Loop till end of byte stream.
Merger Heuristics

• **Pass 1:** If [NP] follows [P] combine them as one [P] block iff the [NP] sub-block contains ASCII printable characters only.

• **Pass 2:** If a [P] sub-block follows another [P] sub-block, join them as one [P] sub-block.

• **Pass 3:** If a [NP] sub-block follows another [NP] sub-block, join them as one [NP] sub-block.

**Example:**

```
root@node1.doom.com -> [P][root@node] [NP][1.] [P][doom.com] ->
[P][root@node1.] [P][doom.com] -> [P][root@node1.doom.com]

AX9..zd*/+ -> [NP][AX] [NP][9..] [NP][zd] [NP][*/+] [NP][AX9..zd*/+]
```
CONST and VAR Components

• Compare corresponding [P] or [NP] groups from all data sets using 2-XOR

• Check if the entire block matches. If yes, this block is constant over all data sets. Mark it as CONST, else mark as VAR.

• Components marked with VAR are the ones which would be picked for fuzzing data.
Length Heuristic

• Assume the bytes preceding [P] block are 0xMN and 0xKL.
  If (length([P]) – 2 < Base_{10}(MNKL) < length([P]) + 2) {
    Mark Preceding Bytes as Length Field.
  }

• If Length([P]) block < 256, pick only ONE byte else TWO.
Delimiter Heuristic

• Largest subsequence of bytes in the packet which has more than $D$ occurrences in the packet.

• $D$ is a configurable component.
Intermediate Results - Sample

```xml
<group id="18"><dstport>21</dstport>
  <packet id="1">
    <comp type="const">
      <actual-data><![CDATA[USER]]></actual-data>
      <binary><![CDATA[55 53 45 52 20]]></binary>
    </comp>
    <comp type="var">
      <actual-data><![CDATA[coverity]]></actual-data>
      <binary><![CDATA[63 6f 76 65 72 69 74 79]]></binary>
    </comp>
    <comp type="const">
      <actual-data><![CDATA[0d 0a]]></actual-data>
      <binary><![CDATA[0d 0a]]></binary>
    </comp>
  </packet>
</group>

<group id="66"><dstport>21</dstport>
  <packet id="1">
    <comp type="const">
      <actual-data><![CDATA[RETR]]></actual-data>
      <binary><![CDATA[52 45 54 52 20]]></binary>
    </comp>
    <comp type="var">
      <actual-data><![CDATA[license.dat 0d 0a]]></actual-data>
      <binary><![CDATA[6c 69 63 65 73 6e 73 6e]]></binary>
    </comp>
  </packet>
</group>
```
Fuzzer Code Generation

- Target for presentation – SPIKE.
- C code generated from packet analysis result XML file.
- All Binary Components marked as CONST
- All VAR components are fuzzed with SPIKE
Sample SPIKE Code Generated

...  
spike_clear();  
s_binary("55 53 45 52 20"); //"USER "  
s_string_variable("coverity");  
s_binary("0d 0a");  
if (spike_send() <= 0) goto xx;

while(s_read_packet() <= 0) // Ensuring State Change of Server  
    usleep(delay*1000);

spike_clear();  
s_binary("50 41 53 53 20"); //"PASS "  
s_string_variable("coverity");  
s_binary("0d 0a");

while(s_read_packet() <= 0) // Ensuring State Change of Server  
    usleep(delay*1000);

spike_clear();  
s_binary("52 45 54 52 20"); //"RETR "  
s_string_variable("license.dat");  
s_binary("0d 0a");
Results on Other Protocols
What we have, What we want?

• Speed in analysis (1 week -> 10 mins)
• Generation of Peach Fuzzers [Work in Progress]
• Application in other areas – File Fuzzing ?? [Work in Progress]
References

References


Thank you!

mrityunjay_gautam@symantec.com