The Art of SIP fuzzing and Vulnerabilities Found in VoIP

Ejovi Nuwere – Mikko Varpiola

About the authors

Ejovi Nuwere

Ejovi Nuwere is the founder of SecurityLab Technologies. Nuwere gained media attention and international recognition for his highly publicized security audit of Japan’s National ID system—JukiNet. Nuwere is the Chief Technology Officer of SecurityLab Technologies where he heads the companies VoIP security auditing group. He currently lives in Boston and is working on his second book, Practical Penetration Testing (O’Reilly).

Mikko Varpiola

Mikko Varpiola is the head of test tool development at Codenomicon Ltd. His specific area of expertise is in anomaly design - e.g. what to feed into software to make it fail. Before Codenomicon he worked as a researcher in the acclaimed PROTOS project at Oulu University Secure Programming Group (OUSPG). He is the author of the ASN.1 encoding anomalies first deployed in the widely-publicized PROTOS LDAP and SNMP test suites.
Agenda

1. Proof of concept
2. Current state of the VoIP security
3. The art of SIP fuzzing

Proof of concept test setup

- Two user agents
- One infrastructure component
- Demonstrate the loss of availability
- Potential security implications of found bugs still under investigation
- Vendors have been notified
Current state...

- Open-source vs. proprietary
  - Large product companies are doing fairly well
  - Some telcos and hardware vendors lacking?
  - How to measure the differences between products?

- Military and private usage
  - Multilevel Precedence and Preemption (MLPP)
  - Small businesses at risk
  - Off shoring by large corporations

Current state continued...

- Progress since 2000?
  - Lack of basic coding flaws (sorry no easy overflows)
  - PROTOS (2001), discovered most basic bugs
  - Some companies begin to have very mature threat modeling

- Back to 1999…
  - Some VoIP vendors have no concept of vulnerabilities (usual call the lawyers, downplay,…)
  - Make it work mentality
  - Closed network assumptions
Testing Approach

- Defining fuzzing terminology
- Products evaluated
  - sipXphone (sip foundry stack)
  - PartySIP (GNU SIPo stack)
  - SIPset (vovida/vocal stack)
  - linPhone (GNU SIPo stack)
  - Commercial Brand X (unknown stack, proprietary?)
- Doesn’t look too promising!
  - At least two critical bugs per product

The problems with SIP

- Share and share alike
  - Many are using the same flawed code base
  - No one admits they are using the same code base
- No update mechanisms for most products
  - Hardware devices
  - Consumer products
- Writing parsers are inherently complex
  - Ethereal (150+ vulnerabilities since 1999)
Don’t forget the environment

- It is essential to understand the environment
- Some errors trigger in only certain environments and certain configurations
- In context of SIP – just think of UDP vs. TCP as a transport
  - Stream vs. Datagram
  - alternate physical limitations for maximum message size
- Beyond the parser lies the application

The Art of SIP fuzzing

- What’s all the fuzz about?
- Deciding what to fuzz
- Isolated bug fault model
- A systematic approach
- What ASCII (as in SIP) brings to the table
- Types of anomalies
What’s all the fuzz about

Bug symptoms usually located:
- Crashes
- Performance degradation
- Other unexpected behaviour

To guarantee:
- Safety
- Security
- Dependability

Positive tests to prove coverage/conformance
Negative tests

Infinity of possible tests

Deciding what to fuzz

- Decision need to relate to available protocols and surrounding environment
- Ideally test all open interfaces
- Environment
  - What are the open interfaces
  - History of identified protocols
  - Risk analysis
- Protocol
  - Only test the actual behaviour
  - Check common sources for known vulnerabilities → Improvise

WITH SIP TRY THESE:
- SIP REQUEST LINE
- SIP URIs in headers below
- Authorization headers
- Contact header
- CSeq
- From header
- Route header
- Record-Route header
- To header
- Via Header
**Isolated bug fault model**

YOU CAN'T TEST EVERYTHING AT SAME TIME – NEITHER YOU CAN DO EVERYTHING IN SAME MESSAGE/ELEMENT!

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. decode input</td>
<td>1. decode input, check syntax</td>
<td>3. update state, generate output</td>
</tr>
<tr>
<td>2. check state</td>
<td>2. check semantics</td>
<td>2. check state</td>
</tr>
</tbody>
</table>

Test Case
- invalid syntax
- invalid state
- both invalid and valid

**A systematic approach**

1. Identify sub structures (required and optional)
2. Identify data types of identified fields
3. Anomalise fields one at the time with proper anomalies for data type
4. Or apply structural mutations

To: aaaaaaaaaaaaaaaaaaaaaa...  
To: `sip: user[:password]@example.com[:5060]`  
To: `@192.168.1.1`  
From: `sip: user[:password]@example.com[:5060]`  
From: `@192.168.1.1`
Fuzzing SIP

- ASCII (as in SIP) allows various levels of freedom
  - Human readable protocols tend to be harder to parse
  - Binary vs. ASCII protocols
  - It is easier to create huge amount of (redundant?) test cases with ASCII based protocols
  - SDP and other content payloads a task of their own (may require special injection arrangements)

Anomalies for ASCII based protocols

- For each anomaly we present
  - Examples up close and personal
  - Them applied to SIP message
- We cover:
  - standard overflows (ascii, c-format strings, control/non-ascii utf8)
  - standard integers (negative, 'float', big)
  - addresses (IPv4, IPv6, ISDN (tel uris))
  - structural (repetitions (header, header element), underflows)
  - protocol specifics (by closely observing the SIP & related specs)
- Why different lengths / values for each data type?
  - All rules of boundary value testing apply to fuzzing as well
  - Different software, different limits
  - Different routines likely get exercised with different strings
**Standard overflows**

- ASCII (alpha vs. Alphanumeric) 16x \text{0x61} ("aaaaaaaaaaaaaaaaaa")
- C-format string 1024x \text{0x62}
- Control character 2048x \text{0x34}
- UTF8 128x \text{0x00}, 512x \text{0x07}, 1024x \text{0x7f}, ...

**Standard integer anomalies**

- Cover the data range with presentative values
- Examine specification for enumerations
  - STANDARD: -1, 0, 1, 2, 4, 5, 6, 7, 15, 16, 32, 63, 64, 127, 128, 255, 256, 1023, 1024, 4095, 4096, ...
  - FLOATS: 0.1, 0.9, -0.1, 0.0, -0.0, ....
  - UNEXPECTED NUMERIC SYSTEMS: 0b000, 0x01, 042364
### Addresses

<table>
<thead>
<tr>
<th>Network Range</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0/8</td>
<td><code>This Network</code></td>
<td>RFC1700, page 4</td>
</tr>
<tr>
<td>10.0.0.0/8</td>
<td>Private-Use Networks</td>
<td>RFC1918</td>
</tr>
<tr>
<td>14.0.0.0/8</td>
<td>Cable Television Networks</td>
<td>RFC1918</td>
</tr>
<tr>
<td>19.0.0.0/8</td>
<td>Reserved but subject to allocation</td>
<td>RFC1918</td>
</tr>
<tr>
<td>128.0.0.0/16</td>
<td>Loopback</td>
<td>RFC1918</td>
</tr>
<tr>
<td>192.0.0.0/24</td>
<td>Reserved but subject to allocation</td>
<td>RFC1918</td>
</tr>
<tr>
<td>192.168.0.0/16</td>
<td>Private-Use Networks</td>
<td>RFC1918</td>
</tr>
<tr>
<td>240.0.0.0/8</td>
<td>Reserved for Future Use</td>
<td>RFC1700, page 4</td>
</tr>
<tr>
<td>255.255.255.255</td>
<td>Broadcast</td>
<td>RFC1700, page 5</td>
</tr>
</tbody>
</table>

### Structural anomalies

- **Repetitions**
- **Header**
- **Sub elements**
- **Underflows**
- **Unexpected data**

---

INVITE sip:userinfo@example.com SIP/2.0  
To: <sip:userinfo@example.com>  
Via: SIP/2.0/UDP from.example.com;branch=v900041580  
Content-Type: application/sdp

```plaintext
v=0  
O=example.com  
S=example.com  
U=tel:+1234567890  
T=0  
X=FOO:BAR  
C=IN  
SCTP/65535  
M   
```

---

INVITE sip:userinfo@example.com SIP/2.0  
To: <sip:userinfo@example.com>  
Via: SIP/2.0/UDP from.example.com;branch=v900041580  
Content-Length: 177  
Content-Type: application/sdp

```plaintext
v=0  
O=example.com  
S=example.com  
U=tel:+1234567890  
T=0  
X=FOO:BAR  
C=IN  
SCTP/65535  
M   
```

Protocol specific anomalies

- SIP Tokens as in RFC3261
- SIP line continuations as in RFC3261
- URI escapes as in RFC2616/RFC1945
- Embedded BASE64 encoding of RFC2617 headers
- UTF8 (see http://www.cl.cam.ac.uk/~mgk25/unicode.html)
- Other SIP specific escapings
- MIME multipart bodies
- You name it!

Protocol specifics continued

INVITE http://user@to.example.com SIP/2.0
To: <sip:user@to.example.com>
From: <sip:user@example.com;tag=00008154>
Via: SIP/2.0/UDP from.example.com;branch=z9hG4bK1109441111191235310
Call-ID: 00008154@to.example.com
Contact: <sip:user@to.example.com;transport=udp>
Content-Length: 77
Content-Type: application/sdp
CSeq: 155 INVITE
Max-Forwards: 70

INVITE sip:user@to.example.com SIP/2.0
To: <sip:user@to.example.com>
From: 'user' <sip:user@example.com>
Via: SIP/2.0/UDP from.example.com;branch=y9hG4bK1109441111191235310
Call-ID: u0c00000651011110904351510@exmple.com
Contact: 'user' <sip:user@example.com>
Content-Length: 177
Content-Type: application/sdp

INVITE sip:example.com:SIP/2.0
From: sip:example.com
To: sip:example.com
Via: SIP/2.0/UDP sip:example.com;branch=y9hG4bK1109441111191235310
Call-ID: u0c00000651011110904351510@exmple.com
Contact: sip:example.com
Content-Length: 177
Content-Type: application/sdp

Authorization:Basics "Ana Maria" "z9hG4bK1109441111191235310" @sip:example.com
Call-ID: 0000081361681011110904666562@from.example.com
Is that all about anomalies?

“Thrill to the excitement of the chase! Stalk bugs with care, methodology, and reason. Build traps for them..... [Beizer]”

“Testers! Break that software (as you must) and drive it to the ultimate - but don’t enjoy the programmer’s pain. [Beizer]”

“The tester in you must be suspicious, uncompromising, hostile, and compulsively obsessed with destroying, utterly destroying, the programmer’s software. The tester in you is your Mister Hyde... [Beizer]”

Conclusions

- VoIP is going prime time – lets fix it before its too late!!!
- Find out what stacks your vendors are using and how they are testing them!
- Its not only the signaling - there is voice and management among others to be worried about as well
- Beyond presented fundamental problems there are other cans of worms to be opened:
  - Tapping, session hijacking, etc....
Questions?

Ejovi Nuwere
ejovi@securitylab.net

Mikko Varpiola
mvarpio@codenomicon.com