Analysis of Zero-Day Vulnerabilities in Java

MARIUS POPA
Department of Economic Informatics and Cybernetics
Bucharest University of Economic Studies
Piata Romana 6, Bucharest
ROMANIA
marius.popa@ase.ro

***Abstract**:* The zero-day vulnerability is a security lack of the computer system that is unknown to software vendor. This kind of vulnerability permits building attack strategies for gaining the access to the resources and data of a computer system. The main issue of the topic is how a computer system can be protected by zero-day vulnerabilities using the actual security procedures and tools for identifying the potential attacks that exploit the vulnerabilities unknown to computer users and software providers. The paper highlights the main features of such kind of vulnerabilities, some exploitation methods and examples of them for Java zero-day vulnerabilities and how protection strategies can be built on intelligence extracted from attack anatomy analysis.

***Key-Words**:* Zero-Day Vulnerability, Security Exploit, Threat Intelligence

1. Features of Zero-Day Vulnerabilities

A Zero-day vulnerability refers to a security lack in a software that is unknown to the software vendor. The security lack is exploited by hackers to vulnerability window closing with security patches by the software vendors. Vulnerability window duration depends on the time when the software vendor becomes aware about the vulnerability and a security patch is developed to close the security window.
The vulnerability window exists between the time when the vulnerability is first exploited and the time when the software vendor publishes the security patch to close the security lack. Taking into account the above considerations, the following actions are stated as being part of the vulnerability window time line:

- The software vendor develops a software application containing an unknown vulnerability;
- The attacker find the vulnerability that is still unknown to the software vendor;
- The attacker develops malicious software that is used or distributed; the software vendor does not know the vulnerability or know it but cannot close it;
- The software vendors or users are aware about the vulnerability exploiting and security patch development starts;
- The software vendor releases the security update to close the security lack.

Sometimes, the security patch is a source of information for attack engineering planned by exploitation developers.
The main issue of a zero-day vulnerability is the software vendor awareness about the vulnerability presence in the released software. Before that, aware hackers about the security lack use malicious software for access to the users' information.
There are situations when the software vendor or technology provider is aware about the presence of the vulnerability. They postpone the security fixes till the vulnerability exploiting or public disclosure of that vulnerability. A very good example is the vulnerability found in Simple Network Management Protocol (SNMP) that was fixed more than 6 months after the time when it was discovered.
The users' information can be accessed by the following attacking techniques:

- Web browsers when the security lack is discovered in such kind of software or they permit software installing on target computer system; web browsers are very popular in accessing the Internet information and distributed software;

ISSN 2067 – 4074

- E-mails to place malicious software on the target computer systems by tricking users to open or install the malware attachments.

It is very difficult to anticipate a zero-day attack because of aware lack about the software vulnerabilities. An attempt to identify zero-day attacks automatically is stated in [2]. According to [2], the analysis method for identifying the zero-days attacks automatically has five steps:

1. Building the ground truth – using the Open Source Vulnerability Database (OSVDB) and other references to collect information about vulnerabilities; each vulnerability is identified on a Common Vulnerabilities and Exposures (CVE) number and carry information about discovery, disclosure, exploit and patch release dates; thus, the candidates for zero-day attacks are established on mapping the threats with their corresponding CVE identifiers;
2. Identifying exploits in executables – identifying the exploits that can be searched in binary data; threats are mapped to the file hashes extracted from antivirus software database;
3. Identifying executables dropped after exploitation – mapping the threats to malicious files as dropped binary files instead of exploit files; the result can be a false positive because the executable cannot be linked to a zero-day attack only then a dropped executable is found in binary data;
4. Analyzing the presence of exploits on the Internet – estimating when the candidate for zero-day attack identified with executable exploit in antivirus software database first appeared on the Internet; the start of the attack is reported when first executable is found;
5. Identifying zero-day attacks – comparing the start date with the disclosure date to find the malicious software used in vulnerability exploitation; a zero-day attack was performed when a hash of a file containing a threat was downloaded before the disclosure time.

The overview of the process for automatic zero-day attacks anticipation is presented in [2], respectively figure 1.

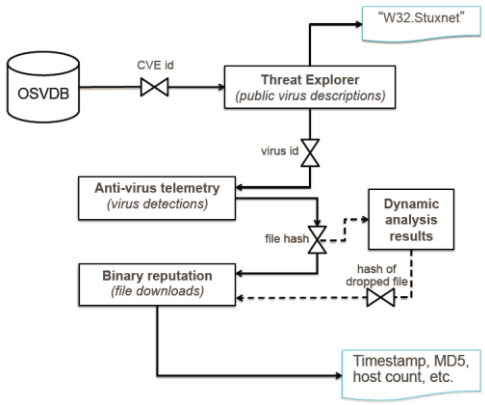

Figure 1 Overview of the process for automatic zero-day attacks identifying, as it is depicted in [2]

OSVDB is a public database containing information about vulnerabilities disclosed since 1998.

CVE is a dictionary about security vulnerabilities and exposures. It is free and public, enabling data exchange between security products.

The protection against zero-day exploitations aims the following techniques, [24]:

- Limiting the effectiveness of zero-day memory corruption vulnerabilities – implemented in operating system software;
- Implementing multiple layers services – aiming the users' access to the software functionalities;
- Using port knocking or single packet authorization daemons – ensuring protection in network services;
- Providing information and support by the software vendors to vulnerable systems;
- Updating the software with the latest security patches and checking the latest updates regularly;
- Paying attention to the e-mail attachments;
- Securing Internet connections.

Avoiding the serious security risks is made by the users making regular checks for new software versions or updates. Also, the released software can have manual or automatic update features. However, the users are exposed to

security attacks to the update launching or checking.
Other ways to prevent exploiting of zero-days vulnerabilities include keeping up-to-date the antivirus software and avoid opening the attachments of the e-mail received from unknown senders.
A very good measure to secure the computer systems is to filter and monitor the network traffic for protecting against the unknown threats. This is made by hardware or software (or both) firewall for blocking the unnecessary traffic at hardware level and access to computer system resources and services at software level.

2. Exploitation Methods of Zero-Day Vulnerabilities in Java

CVE offers standardized solutions for addressing the security vulnerability. Information covers the potential security gaps in security and interoperability. According to [18], CVE means:

- Standard solution in naming the vulnerabilities and exposures;
- Standard solution in description of the vulnerabilities and exposures;
- A dictionary of the vulnerabilities and exposures;
- Common language across disparate databases on the same topic of security;
- Interoperability tool for a better security coverage;
- Starting point among security evaluation tools and databases;
- Free and public security tool;
- Valuable information source recognized by CVE Community.

Java programming language safely executes the untrusted code. The Application Programming Interface (API) methods used by software developers incorporate security functionalities in their programs. The Java's security API contains the *SecurityManager* class that is the most critical because monitors potentially sensitive actions and causes a *SecurityException* when the program tries to perform actions which has no rights to make them for. There is a single static instance of *SecurityManager* class declared in *System* class. When *System.SecurityManager* is set to null, the program is authorized to perform all actions. The typical sensitive actions monitored by *SecurityManager* instance aim, [16]:

- Execution of other applications;
- Opening local files;
- Opening the network sockets;
- Loading additional Java code.

Java applets have a restrictive security policy regarding opening local files and network sockets with the exception when the applets open the home web address. Thereby, the contact of the Java applet with untrusted web sites and working with local files are avoided.
The security policy is managed by Java environment and the latest evolutions demonstrate that the applets can break that policy. The exploits can be executed as Java applets and they use the security manager to disable it and run arbitrary code.
In Java, reflection is supported by Beans API and Reflection API. Beans API code provides information about itself and it is used in applications like Integrated Developer Environments (IDE) or Graphical User Interfaces (GUI). Reflection API provides information about the internal structure of a Java program.
Java reflection APIs can be used to find the classes and their content available in a package. Also, classes used by Java Virtual Machine (JVM) that runs the program can be discovered. The access to the content of the classes is controlled by:

- Security manager having the security policy;
- Access specifier implemented by Java programing language.

The security manager must be aware about the indirect access to the classes and their content via reflection.
Bypassing security manager in Java means introducing security vulnerability into the programming environment. This security vulnerability can be exploited by attackers to execute arbitrary code remotely. An example of such vulnerability is indexed as CVE-2013-0422 in Common Vulnerabilities and Exposures.
The vulnerability impact regarding the confidentiality, integrity and availability of

ISSN 2067 – 4074

the system resources is evaluated and has the following conclusions, [26]:

- Confidentiality – it is compromised because of revealing all system files; the impact is complete;
- Integrity – the system protection is lost and its integrity cannot be assured; the impact is complete;
- Availability – the affected resources are unavailable when the attacker performs exploitation actions on this vulnerability; the impact is complete.

The exploitation can be performed by attackers with little knowledge or skill in computer system security. Therefore, the access to the system resources affected by this vulnerability has a low complexity. Also, the potential attacker does not need credential to pass over authentication barriers implemented in the computer system.

According to [19], CVE-2013-0422 vulnerability is specific to Oracle Java 7 and the exploit has the following operations made by potential attacker:

- *getMBeanInstantiator* method defined in *JmxMBeanServer* class is used to get a private reference to a *MBeanInstantiator*; *findClass* method defined in *MBeanInstantiator* class gets arbitrary *Class* references; according to Java programming specifications, the *Class* instances are classes and interfaces in a running Java application; Java Virtual Machine builds *Class* objects automatically when classes are loaded and by calls to the *defineClass* method;
- Reflection API is used to bypass the security check by *checkSecurityManager* method because the *getCallerClass* method is not able to skip frames related to the new reflection API.

The *findClass* method allows retrieving the *Class* references of any package. The method has a very simple implementation, calling another private method as the below piece of code states:

```
public Class findClass(String className,
ClassLoader loader)
   throws ReflectionException {
   return loadClass(className,loader);
}
```

The method parameters have the following meanings:

- *className* – the specified class name;
- *loader* – the object responsible for className class.

The code implementation of loadClass method in *MBeanInstantiator* class has the below content in Java Platform Standard Edition 7, [23]:

```
static Class loadClass(String className,
ClassLoader loader)
 throws ReflectionException {
 Class theClass = null;
 if (className == null) {
  throw new RuntimeOperationsException(new
IllegalArgumentException("The class name
cannot be null"),
"Exception occurred during object
instantiation");
 }
 try {
  if (loader == null)
  loader =
MBeanInstantiator.class.getClassLoader();
  if (loader != null) {
   theClass = Class.forName(className,
false, loader);
  } else {
   theClass = Class.forName(className);
  }
 } catch (ClassNotFoundException e) {
   throw new ReflectionException(e,
"The MBean class could not be loaded by the
context classloader");
 }
 return theClass;
}
```

The *forName* method called in above *loadClass* method is a static method defined in *Class*. The *forName* method returns the *Class* object associated to the class or interface given in *className* parameter, using the given class loader. Any class in any package is obtained by *forName* method calling.

The access to *MBeanInstantiator* object is necessary to get the class or interface references. The *MBeanInstantiator* object is accessible from a *JmxMBeanServer* object which has an attribute called *instantiator* used for *MBeanInstantiator* object storing.

The *JmxMBeanServer* constructor has the below Java code implementation, [17]:

```
JmxMBeanServer(String domain, MBeanServer
outer,
MBeanServerDelegate delegate,
MBeanInstantiator instantiator,
boolean interceptors,
boolean fairLock) {
 if (instantiator == null) {
   final ModifiableClassLoaderRepository
```

```
   clr                 =                 new
ClassLoaderRepositorySupport();
   instantiator            =            new
MBeanInstantiator(clr);
 }
 this.secureClr = new
SecureClassLoaderRepository(instantiator.get
ClassLoaderRepository());
 if (delegate == null)
   delegate = new MBeanServerDelegateImpl();
 if (outer == null)
   outer = this;
 this.instantiator = instantiator;
 this.mBeanServerDelegateObject = delegate;
 this.outerShell   = outer;
 final   Repository   repository   =   new
Repository(domain);
 this.mbsInterceptor                    =new
DefaultMBeanServerInterceptor(outer,
delegate, instantiator,
repository);
 this.interceptorsEnabled = interceptors;
 initialize();
}
```

The *JmxMBeanServer* class contains the *getMBeanInstantiator* method getting the *MBeanInstantiator* reference:

```
public                      MBeanInstantiator
getMBeanInstantiator() {
 if (interceptorsEnabled)
   return instantiator;
 else                throw                new
UnsupportedOperationException(
"MBeanServerInterceptors are disabled.");
}
```

In such way, having the *MBeanInstantiator* reference the *findClass* method can be called to get any class. *MBeanInstantiator* and *JmxMBeanServer* classes are defined in *com.sun.jmx.mbeanserver* package. *MBeanInstantiator* implements the *MBeanInstantiator* interface. *JmxMBeanServer* class is the core component of the JMX infrastructure aiming management actions with *MBeans*.
The above explained vulnerability affects the recursive reflection security in Oracle Java 7.
In computer programming, reflection is an infrastructure allowing to a computer program to see and manage itself. Self-referring is implemented by metadata that is information about oneself.
The recursive reflection can be exploited as the [25] states:

1. Access to two classes from a restrictive package using the vulnerability described above;
2. Using reflection, a *Lookup* object leads to *findConstructor* method from *MethodHandle* class;
3. Calling *findConstructor* method to get a *MethodHandle* reference;
4. Calling constructor.

The lookup object interacts with security manager for performing additional checks. The calls of the security manager methods are made up to 4 times following the below rules as they are stated in [23]:

- Calling *checkMemberAccess* method of the security manager object in all cases;
- Calling *checkPackageAccess* method to see if the packages are restrictive; it checks the packages when the class loader of the lookup class is not the same or ancestor of the class loader of the class in which the member is searched for;
- Calling *checkMemberAccess* method when the retrieved member is not public;
- Calling *checkPackageAccess* method when the following requirement are met:
  - The retrieved member is not public;
  - The class in which the member is defined and the class in which the member is found are in different class loaders;
  - The lookup class and the class in which the member is defined, have different class loaders or the class loader of lookup class is not an ancestor of the class loader of the other class.

The *checkSecurityManager* method is defined in *Lookup* class and implements the access checks made by the security manager. The security manager is obtained with the *getSecurityManager* method defined in *System* class returning the security manager object reference or null if a security manager was not established for the current application.
The important security bypass appears in step 2 when the check of the class loaders of the current restrictive class and the caller of the reflection API is passed. The piece of code used in implementing step 2 is [25]:

```
SecurityManager            smgr            =
System.getSecurityManager();
...
```

ISSN 2067 – 4074

```
Class<?> callerClass = ((allowedModes &
PRIVATE) != 0
? lookupClass :
 getCallerClassAtEntryPoint(true));
if
(!VerifyAccess.classLoaderIsAncestor(lookupC
lass, refc) ||
(callerClass != lookupClass &&
!VerifyAccess.classLoaderIsAncestor(callerCl
ass, refc)))
smgr.checkPackageAccess(VerifyAccess.getPack
ageName(refc));
```

The explanation of condition bypassing consists in null class loaders and all the classes involved in the check are part of the Java Development Kit (JDK).

The real issue in recursive reflection vulnerability is the *getCallerClass* method call. The *getCallerClass* method is called in *getCallerClassAtEntryPoint* method and work with the frames to be skipped in reflection. The method ignores skipping frames to the new Reflection API, taking into account only the old Reflection API.

The above vulnerability bypasses the Security Manager and affects only the Java applications running in web browsers. It is exploited over a computer network with no username and password. The web browsers having unpatched releases that run Java applets are exposed to the vulnerability.

After CVE-2013-0422 vulnerability, new zero-day Java vulnerability was discovered. Its identifier is CVE-2013-1493 according to Common Vulnerabilities and Exposures dictionary.

Exploitation of CVE-2013-1493 vulnerability aims the Color Management functionality in the 2D component. A possible remote attacker can execute arbitrary code or causes a denial of service of service via an image with crafted raster parameters [20]. The crash is caused by the following triggers, [20]:

- Out-of-bounds read;
- Memory corruption in the JVM.

CVE-2013-1493 vulnerability affects the Java applications running in web browsers. Like CVE-2013-0422 vulnerability, the server, standalone and embedded Java applications are not affected by this zero-day vulnerability.

Other attack vectors are Java Development Toolkit plug-in, Java Web Start and applications that use the browser web-content-rendering components.

The vulnerability impact on system security is assessed on different levels, [27]:

- Confidentiality – all system files are disclosed, therefore the data confidentiality is totally affected by this vulnerability; the impact is complete;
- Integrity – the entire system is affected because it losses the whole protection of data; the impact is complete;
- Availability – the resource becomes unavailable when the attacker takes the control of that resource by vulnerability exploitation; the impact is complete.

Exploitation of CVE-2013-1493 vulnerability does not need complex knowledge or very good skills in computer system security. The authentication is not necessary to gain the control of the resource. Also, a computer network is necessary to perform the remote attack.

Information regarding the JVM architecture and how the bytecode is interpreted by a virtual machine are presented in [10], [11] and [13].

CVE-2013-1493 vulnerability permits to attacker arbitrary memory read and write in JVM process. The vulnerability targets the memory that stores the JVM internal data structure. Once the target memory is identified, it is overwritten by a memory chunk containing bytes with null values. Thereby, it is possible to disable the security manager making null the reference, following by exploitation method escalating the untrusted privileges of the code.

The successful exploitation of this vulnerability downloads and executes a Mc Remote Access Tool (McRAT) backdoor as an executable from the same server hosting the Java Archive (JAR) file. Thus, the targeted computer system is infected by virus software dropped by other malware application or downloaded from the Internet.

The main issue of this vulnerability is the binary code of *initImageLayouts* functionality from Color Management which permits buffer overflow operations [28].

According to Symantec, the virus software is detected as *Trojan.Naid* which connects the computer system to a command-and-control server. *Trojan.Naid* is signed by the compromised *Bit9* certificate and a communication channel is established to IP address 110.173.55.187, [15].
The exploitation method of CVE-2013-1493 vulnerability is depicted at [15] and the stages of the attack are presented in figure 2.

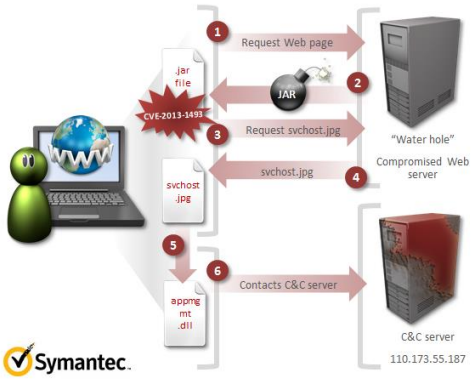

Figure 2. Exploitation method of CVE-2013-1493 vulnerability, as it is depicted in [15]

The exploitation steps are, [15]:

1. Web page request – the target system visits a compromised web site containing a malicious JAR file (Java applet); Symantec identifies the malicious JAR file as *Trojan.Maljava.B*;
2. Exploit file download – the JAR file is downloaded and executed on target system in the web browser; it contains the exploit of CVE-2013-1493 vulnerability; the attacker can place the malicious JAR file compromising a legitimate web site or convincing the user to visit the web site by social engineering techniques;
3. Malware request – the exploit download a file called *svchost.jpg* that is an executable file; *svchost.jpg* is identified by Symantec as *Trojan.Dropper*;
4. Malware download – the *svchost.jpg* file is download on target system;
5. Infection and persistence – the *svchost.jpg* file load the *appmgmt.dll* file that is persistent on the target system; *appmgmt.dll* file is identified as *Trojan.Naid* by Symantec;
6. System compromising – *appmgmt.dll* file opens a communication and control channel to a command-and-control server for target system crash.

The persistence of McRAT in the computer system is assured by copying itself as a DLL to **C:\Documents and Settings\admin\AppMgmt.dll** together with the following registry modifications, [6]:

```
\REGISTRY\MACHINE\SYSTEM\ControlSet
001\Services\AppMgmt\Parameters\"Se
rviceDll"    =    C:\Documents    and
Settings\admin\AppMgmt.dll

\REGISTRY\MACHINE\SYSTEM\ControlSet
001\Services\AppMgmt\Parameters\"Se
rviceDll"                          =
%SystemRoot%\System32\appmgmts.dll
```

Disinfection techniques for exploited CVE-2013-1493 vulnerability are manual or automatic using anti-virus software. The manual techniques aim the following steps:

1. Deleting malicious processes – is made from Task Manager by deleting the executable file from Processes tab having a random character filename;
2. Deleting associated files – is made from the file system of the computer system; the target folders are Desktop and Programs;
3. Cleaning Windows registry – is made from the registry editor by deleting entries in registries identified by random characters filename.

As prevention measures to avoid the exploitation of the CVE-2013-1493 vulnerability, it includes:

- System patching with the latest security updates;
- Strong administrator credentials preventing the brute force attacks on administrator passwords;
- External storage device scanning when they are inserted into the system;
- Authenticated sharing of files and folders;
- Legitimate web site visiting for multimedia resources and computer games.

The zero-day vulnerabilities presented in this chapter are both remote

ISSN 2067 – 4074

vulnerabilities requiring no authentication in to computer network to exploit them. However, the vulnerability type is different for analyzed zero-day vulnerabilities. CVE-2013-0422 is a security manager bypass, and CVE-2013-1493 is JVM memory corruption vulnerability.

3. Extracting Intelligence from Zero-Day Attacks

Intelligence means acquiring and applying knowledge and skills. In the context of computer security, especially the zero-day vulnerabilities, intelligence is the ability to learn, understand or to deal with new challenge situations generated in computer systems. The acquired knowledge is used to manage the computer security in information systems or to think abstractly for development of new improved security architectures or protection methods and techniques.

To understand, learn and apply improved knowledge in prevention of zero-day attacks, it is necessary to analyze the attack anatomy. SANS Institute describes the attack progression in the following sequential steps, [4]:

1. Reconnaissance – provides an attacker information regarding the attack target; it includes actions like web browsing, online documentation, learning the internal structure of the target, port scans, system enumeration and so forth; the goal is acquiring information about the attack target; it is very difficult to identify these actions from normal activity to detect the possible attacker's behavior;
2. Weaponization – aims the way in which the malicious payload is placed into a delivery unit; it is a technique to hide the malicious payload to be not detected by the victim; understanding of how the weaponization is made by the attacker can be done by reverse engineering of delivered payloads; features and methods about reverse engineering are presented in [10], [11] and [13];
3. Delivery – is the critical step where the payload is passed to the target; the payload can have different formats as malicious payload hidden in a transportation unit, social engineering vector and so forth; the target must manage the received payloads and visited web sites;
4. Exploitation – is made on software, hardware or human vulnerability; the malicious payload is executed and compromises the victim's computer system exploiting the identified vulnerability; the victim can be aware or not about the exploitation; when the exploitation is known, the victim can apply specific procedures as incident response; if the exploitation has an unknown behavior then an intelligence-driven response has to be carried out and it is called compromise response;
5. C2 – means the command-and-control step; it is the period of time in which the attacker exploits the target system; the action patterns observed during this step can be used to implement response actions for the next attacks in order to prevent loos of information and resources;
6. Exfiltration – is the step when data is taken from the target system; data is used to access sensitive information or to access other data useful for the attacker's purpose.

An attack can be identified in two ways, [5]:

- Persistence in the computer network of the organization;
- Repeatability of some actions trying to access resources in the target system.

Persistence and repeatability of the attacks are established on indicators used by security analysts for attack identifying. Such indicators can be an untrusted web site address visited repeatedly, a piece of metadata from the binary malicious code, a suspicious HTTP request and so forth.

The incident response is carried out for compromised systems. The attack detection happens after system exploitation. The security analysts must reconstruct every prior stage to obtain knowledge about the attack anatomy. Therefore, the organization increases its capability for an early response to a

similar attack that is detected prior to system compromising. The security analysts also analyses the prior steps and synthesizes the steps after attack detection to see what the changes must be applied in the incident response methodology, [5].

Gathering intelligence from zero-day attacks is possible beginning with the awareness of victim or software vendor about the computer system vulnerability. Sometimes, the attack is quietly carried out and the victim does not know that its data and resources are accessed and used by somebody else.

Regarding the CVE-2013-0422 vulnerability, the attack analysis has revealed lacks in the management of permissions, privileges, and other security features that are used to perform access control from Java reflection APIs. It is a vulnerability introduced during implementation of the APIs.

The risk mitigation aims separation of privileges when the access rights in the computer system are assigned to the entities.

The analysis of the attacks exploiting the CVE-2013-1493 vulnerability has highlighted improper operations on memory buffer permitting reads and writes outside the intended boundary of the memory buffer. It is a lack of the JVM introduced during software development lifecycle of the programming environment.

The risk mitigations include the following actions:

- Using appropriate language providing memory overflow protection;
- Using appropriate libraries or frameworks providing memory overflow protection;
- Compiling and running the software applications using features or extensions that mitigate or eliminate memory corruption;
- Considering the best practices in memory allocation and management;
- Using Address Space Layout Randomization (ASLR) to arrange randomly the memory key areas of the application;
- Using a Central Processing Unit (CPU) or operating system that provides Data Execution Protection (NX) to prevent execution of binary code from non-executable memory areas;
- Using appropriate memory copy functions considering the length of the copied memory areas.

Extracted intelligence helps the improvement of the protection strategies by understanding the attack anatomy as well as the trend in tools, tactics and procedures used by attackers.

4. Conclusion

There is an increasing trend of the zero-day vulnerability exploitations. In the first three months of 2013, it observed 11 zero-days vulnerability exploitations affecting Oracle Java, Adobe Flash, Adobe Reader and Microsoft Internet Explorer.

Also, another trend is the attackers who become more sophisticated, having high skills and knowledge about the target system. They dig deep into the protection systems to find methods for restriction bypassing for a complete exploitation.

Finding the ways and tools to prevent or mitigate the system compromises has become critical. An improvement way to do that is analyze of the zero-day attacks to establish their features and build some kind of patterns for future considerations of the potential attacks to the computer systems.

References

[1] Felician Alecu, Paul Pocatilu, George Stoica, Cristian Ciurea, Sergiu Capisizu, OpenID, a Single Sign-On Solution for E-learning Applications, Journal of Mobile, Embedded and Distributed Systems, vol. 3, no. 3, 2011, pp. 136 – 141

[2] Leyla Bilge, Tudor Dumitras, Before We Knew It: An Empirical Study of Zero-Day Attacks in The Real World, Proceedings of the 2012 ACM conference on Computer and communications security, Raleigh, NC, USA, Oct. 16-18, 2012, pp. 833 – 844

[3] Cătălin Boja, Security Survey of Internet Browsers Data Managers, Journal of Mobile, Embedded and Distributed Systems, vol. 3, no. 3, 2011, pp. 109 – 119

[4] Mike Cloppert, Security Intelligence: Attacking the Cyber Kill Chain, SANS Computer Forensics, October 14, 2009,

ISSN 2067 – 4074

http://computer-forensics.sans.org/blog/2009/10/14/security-intelligence-attacking-the-kill-chain/
[5] Mike Cloppert, Security Intelligence: Defining APT Campaigns, SANS Computer Forensics, June 21, 2010, http://computer-forensics.sans.org/blog/2010/06/21/security-intelligence-knowing-enemy/
[6] Jan Henrik, New 0-day vulnerability in Java JRE 1.7.0 Update 15 and 1.6.0 Update 41 is being exploited in the wild, March 1, 2013, http://www.mnemonic.no/en/Andre-sprak/English/Blog/java-vuln-1march-2013/
[7] Ion Ivan, Dragoş Palaghiţă, Sorin Vînturiş, Mihai Doinea, Vulnerability Analysis in Web Distributed Applications, Journal of Mobile, Embedded and Distributed Systems, vol. 3, no. 1, 2011, pp. 1 – 9
[8] Darien Kindlund, Yichong Lin, YAJ0: Yet Another Java Zero-Day, February 28, 2013 http://www.fireeye.com/blog/technical/cyber-exploits/2013/02/yaj0-yet-another-java-zero-day-2.html
[9] Paul Pocatilu, Android Applications Security, Informatica Economică, vol. 15, no. 3, 2011, pp. 163 – 171
[10] Marius Popa, Binary Code Disassembly for Reverse Engineering, Journal of Mobile, Embedded and Distributed Systems, vol. 4, no. 4, 2012, pp. 233 – 248
[11] Marius Popa, Characteristics of Program Code Obfuscation for Reverse Engineering of Software, Proceedings of the 4th International Conference on Security for Information Technology and Communications, 2011, ASE Publishing House, Bucharest, pp. 103 – 112
[12] Marius Popa, Techniques of Program Code Obfuscation for Secure Software, Journal of Mobile, Embedded and Distributed Systems, vol. 3, no. 4, 2011, pp. 205 – 219
[13] Marius Popa, Using the Disassembly of Binary Executables in Reverse Engineering of Software, Proceedings of the 5th International Conference on Security for Information Technology and Communications, 2012, ASE Publishing House, Bucharest, pp. 7 – 17
[14] Cristian Toma, Security Issues for 2D Barcodes Ticketing Systems, Journal of Mobile, Embedded and Distributed Systems, vol. 3, no. 1, 2011, pp. 34 – 53
[15] Latest Java Zero-Day Shares Connections with Bit9 Security Incident, Symantec, March 1, 2013, http://www.symantec.com/connect/blogs/latest-java-zero-day-shares-connections-bit9-security-incident
[16]http://www.cert.org/blogs/certcc/2013/01/anatomy_of_java_exploits.html
[17]http://www.cs.rit.edu/usr/local/pub/swm/jdoc7/com/sun/jmx/mbeanserver/package-summary.html
[18] http://cve.mitre.org/about/index.html
[19]http://cve.mitre.org/cgi-bin/cvename.cgi?name=2013-0422
[20]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1493
[21]http://cwe.mitre.org/data/definitions/119.html
[22]http://cwe.mitre.org/data/definitions/264.html
[23]http://docs.oracle.com/javase/7/docs/api
[24]http://en.wikipedia.org/wiki/Zero-day_attack
[25]https://partners.immunityinc.com/idocs/Java%20MBeanInstantiator.findClass%200day%20Analysis.pdf
[26]http://www.cvedetails.com/cve-details.php?cve_id=CVE-2013-0422
[27]http://www.cvedetails.com/cve/CVE-2013-1493/
[28] http://www.osvdb.org/90737
[29]http://www.pctools.com/security-news/zero-day-vulnerability/?goback=%2Egna_38412