Active Defense Technology Based on Natural Software Diversity in Java Web Services

Zhenwu Liu$^{1,2}$, Zheng Zhang$^{1,2*}$, Hao Liu$^{1,2}$, Gaofei Zhang$^{1,2}$, Yuan Chen$^3$

$^1$State key Laboratory of Mathematical Engineering and Advanced Computing, ZhengZhou, HeNan, 450001, China
$^2$PLA Strategic Support Force Information Engineering University, ZhengZhou, HeNan, 450001, China
$^3$Jiangnan Institute of Computing Technology, WuXi, JiangSu, 214083, China

e-mail: mjandlzww@qq.com
*e-mail: ponyzhang@126.com

Abstract. With the popular use of network application services and the growing number of users, web services have become the main target of hackers. Traditional single software has insufficient security protection against unknown threats and is prone to various vulnerabilities. In this paper, we adopt a new type of active defense strategy in java web services. At different application layers, we use existing components based on the natural software diversity to form different software stacks, and use dynamic scheduling mechanisms to change the attack surface at all times to obtain security protection. Our analysis shows that the flexibility of java web services using this defense strategy is enhanced, and the exploitability of its vulnerabilities is effectively reduced.

1. Introduction

With the advent of the "Internet +" era, the Internet has fundamentally changed people's life and production mode. At the same time, cyberspace security issues have become increasingly serious. For a long time, it has been considered in biology that monoculture is extremely susceptible to pest and disease disasters. For example, the Irish potato famine caused by phytophthora in the 19th century. The European potato is so dependent on a single crop that it has no resistance to pests and diseases. The phenomenon of crisis in a single biological ecosystem also occurs frequently in the current software ecosystem. Due to the rapid development of the big data, cloud computing industry, and certain software occupying a monopoly position in the market, the current scale and parallelism of software is amazing. Once a security vulnerability is discovered, the number of users affected is huge and the cost of the loss is high.

As companies and other organizations increasingly deploy services and data in the cloud, a system services catastrophe could be triggered by a small outage on the cloud server. On the early morning of March 3, 2019, aliyun, which had a 43 percent market share in China and ranked first, suffered a server failure. Many Internet company services were crippled by the outage. With the frequent disclosure of high-risk security vulnerabilities and the frequent occurrence of malignant security incidents, the relevant interests of individuals, enterprises and even the country suffer serious harm due to the abnormality and paralysis of system services and software functions. Such as in May 2017,
WannaCry, a worm-like ransomware software, was spread by criminals using the high-risk vulnerabilities “EternalBlue” of Windows system. Important documents in computers in key industries such as governments, enterprises, education, healthcare, energy, communications, transportation, and manufacturing in many countries have been encrypted, and many critical information infrastructures have suffered unprecedented damage.

![Vulnerability distribution map]

As shown in Figure 1, according to the data collected by the China National Vulnerability Database (CNVD)[1], web applications account for up to 20% of vulnerabilities. Attacking any web service script, application or system can cause millions of customers' sensitive data to leak. Therefore, it is very important to protect web services from hackers. The traditional single software is not enough to protect against unknown threats, mainly reflected in the following points:

1. Software developers usually focus on the performance and functionality of the application and less on security. Simplification of web services development and deployment often guarantees functionality and promotes ease of use at the expense of security.

2. Any vulnerability anywhere in the web services’ software stack may lead to unauthorized users gaining full control over the web service.

3. Encryption technology and other security measures can guarantee the security of online network traffic and messages during transmission, but they cannot avoid attacks against application vulnerabilities.

4. It is difficult for static defense strategies to prevent attackers from observing and discovering new software vulnerabilities or breaking detection mechanisms.

In this paper, we adopt an active defense strategy and uses software diversification technology supplemented by dynamic transformation technology to enhance the security of java web services. This paper is mainly based on natural diversity in the operating system layer and web service software layer, and creates randomness through the combination of existing mature web service components.

2. Background

2.1. Moving target defense
MTD (moving target defense)[2] increases the complexity of network attacks through frequent changes in system configuration and software diversification, reducing system homogeneity, staticity, and certainty, and making the attack surface more dynamic. MTD increases the uncertainty in the attack process, invalidates the prerequisites of the attack, and then causes the attacker to fail the operation during the scanning and detection, vulnerability exploitation and other stages, which greatly increases the difficulty and cost of the successful attack. For example, we can randomize the IP
address or the base address of the process memory segment. MTD cannot directly eliminate vulnerabilities, but it can limit the opportunities for vulnerabilities to be exposed, so it can reduce the possibility of being attacked and reduce the asymmetric advantages that attackers have. The effectiveness of the MTD method depends on the number of components to be moved (portions to be moved) and the frequency of movement (moments to be moved). Applying MTD to web services can divide related specific layers, then change the application runtime environment, change the application type and control version, and route through different hosts.

2.2. Software and hardware diversity of web services
Most web services are implemented using a fixed software stack that includes web server software, web applications, operating systems, and virtualization layers. Web services can also be implemented by software stacks composed of different reusable components. Defenders can combine different software diversification strategies and deploy multiple different components at the same time in multiple layers of web services to provide different combinations of protection. These programs provide the same functionality, but were developed by different technical teams. If one handler is hacked or crashed, other programs should still be able to handle client requests. To achieve software diversity, defenders need to find at least two pieces of software that provide the same service. Most software has alternatives, while those without replacements are mostly hardware-specific software (such as hardware drivers and firmware), operating system-specific software, specific domains software, and custom software for specific areas (such as software in aerospace, medical, biological, nuclear, etc.).

Table 1. Software and hardware diversity of web services.

| Stack level          | Diversity Example          |
|----------------------|----------------------------|
| Web Application Code | Perl, PHP, Ruby, Java, Python |
| Web Server Software  | Apache, Nginx, Lighttpd, Tomcat, Jetty, Resin |
| Operating System     | Windows, Centos, Ubuntu, Debian |
| Virtualization       | VMware Workstation, Virtualbox, KVM |
| Hardware             | X86, ARM                    |

There are a large number of functionally equivalent resources in the Internet web service system, such as the application layer, server software layer, operating system layer, and virtualization layer. As shown in Table 1, using high-level programming languages such as Java, PHP, or Python at the application layer to implement application business logic; receiving HTTP requests from clients at the server software layer, parsing and passing the requests to the corresponding server-side programs, such as Apache, Lighttpd, and Nginx, etc.; providing the basic operating environment for the web service layer and the application layer in the operating system layer, such as Windows, Linux, and Unix.

3. Related work
There have been many studies on active defense of web security. Taguinod et al. proposed to apply MTD in four layers of web application: logic layer, storage layer, presentation layer and browser.[3] Huang et al. proposed creating and rotating between a set of virtual servers to move the network attack surface. Each virtual server is configured with a unique software combination.[4] Thompson et al. adopted a strategy that combines the use of operating system diversity with dynamically changing footprints to improve defense capabilities. The same application can run on multiple virtual machines with different operating systems. MTD is implemented by rotating the operating system running the web application.[5] Thompson et al. then successfully achieved their goals of increasing uncertainty and resilience by redirecting incoming network traffic to non-standard web ports.[6] Parno et al. developed CLAMP, a runtime protection system that addresses access control vulnerabilities. CLAMP protects sensitive user data by virtualizing and isolating application components running on behalf of different users, preventing sensitive data from leaking to anonymous and unprivileged users. CLAMP
instantiates the web stack from the main web server and assigns a virtual web server instance to each user’s web session so that users can only access their own data.[7] The content security policy (CSP) proposed by Stamm et al. has been implemented in all major web browsers, such as FireFox, Chrome and Safari. In CSP, the developer or maintainer of a web application determines the strategy for each web page. CSP specifies which resource type in the page the browser must load.[8] As can be seen from the above, there is no good solution to the active defense issue for java web service security.

4. Defense model

4.1. DRS model

As shown in Figure 2, the Dissimilar Redundancy Structure (DRS) [9] model consists of an input proxy, a functionally equivalent heterogeneous redundant executor set, and a voter. Task executors A₁ to Aₙ are completely independent individuals, all of which have the ability to complete a task independently, and there are also some vulnerabilities. However, when these executors perform a task together, they can correct each other's errors and seek common ground while shelving differences, thereby ensuring that the task can be successfully completed. In the case of running multiple heterogeneous programs in parallel to maximize the heterogeneity and sending program inputs to the heterogeneous programs at the same time, it can tolerate operating errors and detect inconsistent behaviors to detect attacks, and achieving the effect of early warning and alarm. Although many functionally equivalent heterogeneous redundant executors are running at the same time, the DRS model still maintains certain staticness and certainty.

4.2. DHRA model

As shown in Figure 3, the Defense Hybrid Redundancy Architecture (DHRA) [10] model consists of an input strategy distribution, a functionally equivalent heterogeneous executor set, a strategic scheduling, a strategy voting, a voter, and an output. The heterogeneous executor set consists of E₁, E₂, E₃, ..., Eₘ, and the component pool is used to store the heterogeneous executor set. The strategic scheduling is responsible for scheduling the tasks to the heterogeneous executors, and the strategy voting is used to vote on the results of the heterogeneous executors to ensure the correctness of the results. The voter is used to make a decision based on the voting results to ensure the reliability of the results. The output is used to output the final results.
As shown in Figure 3, based on the DRS model, the Dynamic Heterogeneous Redundant Architecture (DHRA)[9] model introduces a policy scheduling mechanism to achieve a dynamic defense effect.[10] Each executor in the set is functionally equivalent and structurally different. Each service function is performed by multiple heterogeneous executors. The input will be forwarded to each executor \((A_1-A_n)\) in the set through the input agent, and the output of each executor will be compared by the voter to obtain the system output result. After detecting inconsistent behaviors or a certain period of time, the policy scheduling mechanism takes the currently running executors offline and cleans them, and then selects the same amount of heterogeneous executors from the hot standby heterogeneous executor set to go online to ensure service stable and uninterrupted, thus presenting a dynamic attack surface.

5. Implementation

5.1. Structure design

Figure 4. Structure design of java web service system.

Figure 4 presents the logical structure design of the system at multiple layers, such as the hardware platform layer, operating system layer, data storage layer, virtualization layer, server software layer, and application code layer. With special scheduling strategies, vulnerabilities or backdoor attacks are included in these layers to achieve the effect of blocking vulnerabilities or backdoor attack chains. In response to the security threats faced by web services, multiple designs have been adopted at multiple layers of the web service software stack, using a large amount of existing heterogeneous software. In the structural design, the entrance adds a request distribution equalization module as the input proxy of the web service, and the exit adds a response redundancy voter as the output proxy of the web service. The abnormality and failure of web services were discovered through the voting mechanism, and the dynamic executor was cleaned and recovered through the scheduling mechanism in time to ensure the security of web services. The following features are achieved through dynamic composition at multiple layers:

1) Heterogeneity: Deploy different types of heterogeneous software and hardware at different layers. For example, the software layer consists of optional sets of web server software such as Resin, Tomcat, Jetty, and the system layer consists of an optional set of operating systems such as Windows, Linux, and Unix.

2) Redundancy: For the same request, multiple different software and hardware are used to execute the request simultaneously and vote on the results to achieve redundant execution.

3) Dynamic: Proactively replace heterogeneous executors in the current service set based on scheduling policies or feedback from threat perception, increasing system uncertainty.
5.2. Architecture design

As shown in Figure 5, we use the complete processing mechanism of distribution-processing-voting-logging-scheduling. Designing heterogeneous redundant executors at multiple layers of the web service. The distributor and the voter are the actual entrance and actual exit of the user's access respectively. The executor set is the real bearer and provider of web services, and is composed of heterogeneous and redundant web service executors. The distributor copies the web service request into multiple copies and distributes it to multiple online executors in the set. The voter receives the responses from multiple online executors, votes on the responses from different executors according to the large number voting algorithm, and outputs the large number consistent results. If there is an inconsistency in the voting results, the scheduler will simultaneously clean and recover multiple online executors that generate responses based on the exception-driven scheduling strategy. In addition, the scheduler can also periodically select multiple executors online for cleaning and recovery based on the time-driven scheduling strategy according to the cycle time set by the system. The ruling log can further improve the feedback scheduling strategy to facilitate business function consistency testing and correction.

5.3. Effect

Figure 6. Defensive effect.
As shown in Figure 6, starting with the software structure of web services, using mechanisms such as heterogeneity, redundancy, randomization, and dynamics, the influence range is set in the response message of the web service communication. By ruling the returned response message, the attack difficulty of the web service system is increased, and the success probability of the vulnerability exploitation in the web service is reduced, which can achieve the following effects:

1. By taking advantage of the heterogeneity of the operating environment between web applications, it can resist attacks against web applications such as file uploads, file inclusion, and physical path leaks.

2. By utilizing the natural diversity of web server software, constructing heterogeneity at the server software layer, it is able to reduce the probability of a single server software vulnerability exploitation event, and resist attacks against a specific web server software vulnerability.

3. By making use of the natural diversity of the operating system and constructing heterogeneity at the operating system layer, the conditions for virus outbreaks, and Trojan connections are more stringent, making it more difficult for web service system resources to be used.

As shown in Figure 7, the software stack consisting of web server software, operating system, and virtualization software, constitutes the attack surface of web services. Any vulnerability in any component of the web services software stack is a potential threat to the entire web service. Software diversification can change the attack surface of software and make the vulnerability sets different for different software attack surfaces. But software diversification does not eliminate the loopholes in the software, and may not necessarily resolve logical design flaws or protect other web service components from attacks. Assume that the attack surfaces of all web service software combinations are completely different, that is, the vulnerability set of the web service software combination does not intersect, and there are k vulnerabilities in n different attack surfaces/software stacks. Then the
The probability that all k exploits reach the target attack surface/software stack from n different attack surfaces/software stacks can be calculated as:

\[ p = \frac{1}{\binom{n}{k}} \]

### 6. Evaluation

#### 6.1. Security

| Items                                      | Physical Host Operating System | Virtual Machine Operating System | Web Server Software |
|--------------------------------------------|--------------------------------|----------------------------------|---------------------|
| Software configuration with safety protection | Target                         | Windows                          | Tomcat              |
| Proxy                                      | Windows Server                 | Windows                          | Jetty               |
| Pool A                                     | Centos                         | -----                            | Nginx               |
| Pool B                                     | Windows Server                 | Windows                          | Tomcat              |
|                                            | Debian                         | Ubuntu                           | Jetty               |
|                                            | Debian                         | Ubuntu                           | Resin               |
| Pool C                                     | Windows Server                 | Ubuntu                           | Tomcat              |
|                                            | Centos                         |Centos                            | Jetty               |
|                                            | Centos                         |Centos                            | Resin               |

The experimental environment software configuration is shown in Table 2. The equipment mainly includes 4 computing and storage nodes, of which 1 node deploys the centos system environment, which is responsible for data synchronization transmission, traffic replication distribution, and response output ruling; 3 nodes deploy the windows server system environment as a carrier of the heterogeneous executors’ pool. Multiple functionally equivalent heterogeneous executors are distributed on 3 nodes. Heterogeneous executors are java web service systems that are responsible for providing real web services. In addition, there is also an unprotected web executor for comparison test verification of defensive effects. The topology of the security assessment is shown in the figure 8.

![Figure 8. Topology for security assessment.](image-url)
To analyze the effectiveness of software diversification strategies in response to web service attacks, we conduct security analysis and verification tests for different types of vulnerability exploitation, including application layer jsp Trojan, software layer tomcat and jetty vulnerabilities, system layer vulnerabilities and Trojan. The test set is shown in Table 3:

| Test items | Type              | Description                                                                 |
|------------|-------------------|-----------------------------------------------------------------------------|
| Jsp Trojan | Web Application   | Constructing a jsp Trojan using metasploit's java/jsp_shell_reverse_tcp module |
| Tomcat     | Web Server Software | CVE-2017-12615 By constructing a special suffix name to bypass tomcat detection and upload malicious files |
| Jetty      | Web Server Software | CVE-2015-2080 Trigger an exception by submitting a malicious request and shift it into a shared buffer to obtain user sensitive data |
| EternalBlue| Operating System | Use metasploit's exploit/windows/smb/ms17_010_eternalblue module for EternalBlue exploits |
| System Trojan | Operating System | Construct system trojan using metasploit's linux/x86/meterpreter/reverse_tcp or windows/meterpreter/reverse_tcp module |

6.1.1. **Web application layer - jsp Trojan.** Code injection attacks are one of the most common sources of attacks in web services. In a code injection attack, an attacker attempts to inject code that is executed by a corresponding interpreter on the server. The root cause of web code injection attacks is that interpreted languages (such as php, jsp) cannot distinguish between web application code and code injected by the attacker, so the interpreter will execute any code passed by the web server. Successful execution of the injected code can enable the attacker to achieve remote control. We use metasploit's java/jsp_shell_reverse_tcp module to construct a jsp Trojan. Assume that the attacker pre-uploaded the jsp Trojan to a common web server and the system in some way. By connecting the Trojan file on the ordinary web server through the Trojan control software, the attacker can obtain system permissions and can perform file upload and modification operations on the server. While in our system environment, the attacker cannot connect due to the inconsistent information returned by the Trojan, which prevents it from gaining system administrator permissions.

6.1.2. **Web server software layer - tomcat.** The Tomcat remote code execution vulnerability (CVE-2017-12615) affects Apache Tomcat v7.0.0-7.0.79. When the HTTP PUT request method is enabled for tomcat, an attacker can bypass tomcat detection by constructing a special suffix name, thereby uploading a malicious file through a malicious request. We first deployed tomcat with the same vulnerabilities in the common web server and our system. In ordinary web servers, attackers can successfully upload malicious files by exploiting the tomcat vulnerability. In our system, the attacker submitted a malicious request by using the same vulnerability. However, he could not successfully exploit the vulnerability because the vulnerability will not take effect simultaneously in heterogeneous software layers.

6.1.3. **Web server software layer - jetty.** The Jetty remote disclosure vulnerability (CVE-2015-2080) affected the Jetty web server v9.2.3-9.2.8. When malicious characters are inserted in the header, the attacker can trigger the exception handling code by submitting a malicious request and offset it into the shared buffer. As a result, sensitive data such as previous requests submitted by users stored in the shared cache area can be obtained remotely through this vulnerability. We first deployed jetty with the same vulnerability in the common web server and our system. In ordinary web servers, the attacker can successfully obtain user sensitive data by exploiting jetty’s vulnerability. In our system, the attacker submits a malicious request by using the same vulnerability. But he could not successfully exploit the vulnerability for the same reasons as above.
6.1.4. Operating system layer - EternalBlue. After the attacker transformed and implanted the "EternalBlue", the WannaCry ransomware became a powerful killer. Eternalblue exploited a remote code execution vulnerability MS17-010, which targets port 445 of the Windows Server Message Block (SMB). Older operating system versions (mainly Windows XP and Windows 7) are relatively more vulnerable to this vulnerability. We deploy the Windows 7 system on both the ordinary web server and our system, and then use the exploit/windows/smb/ms17_010_eternalblue exploit module of metasploit to attack. In a normal web server, an attacker can successfully get a meterpreter session, and then use shell commands to effectively access and control remote system. In our system, the same operation is performed. Since the vulnerability cannot be effective at the same time at the heterogeneous system layer, the connection establishment session is rejected and subsequent attacks cannot be performed.

6.1.5. Operating system layer – system Trojan. An attacker usually implants a Trojan into the target system, and then remotely connects to the Trojan to execute related commands, obtaining critical information of the controlled host or destroying it. We use metasploit's linux/x86/meterpreter/reverse_tcp or windows/meterpreter/reverse_tcp module to construct a system Trojan. Assume that the attacker pre-uploaded the system Trojan to a common web server in some way, and monitor using metasploit. When the system administrator is successfully induced to run the system Trojan, an attacker can use the system Trojan to obtain server permissions. In our system, the attacker performs the same operation. But he cannot launch the attack because the Trojan cannot run on all heterogeneous execution environments at the same time, which causes the metasploit fail to listen for connections.

6.2. Performance

We performed a system performance test as shown in Figure 9. Figure 9 (a) is a virtual host hosting web services, and Figure 9 (b) is a physical host hosting web services. The test uses the Sprient Avalanche tester to simulate many HTTP requests sent to the server under test to test the number of transactions per second, the maximum number of TCP concurrent connections, and the throughput. The tested virtual software is vmware. The virtual machine uses dual core and dual threads, 2G memory, and the web server software is nginx. When testing throughput, the page size is 32kB, and the rest is 1KB. The test results are shown in the table 4.
Table 4. Testing records of performance.

| Test items                  | Physical machine (Centos7) | Virtual machine1 (Centos7) | Virtual machine2 (Debian8) |
|-----------------------------|---------------------------|----------------------------|---------------------------|
| transactions per second     | 61245                     | 5114                       | 4362                      |
| TCP concurrent connections  | 3271386                   | 186652                     | 179230                    |
| throughput                  | 9688517 kbps              | 472144 kbps                | 465716 kbps               |

According to Table 4, using virtualization technology will inevitably lead to system performance degradation. As the price of multi-core CPUs and memory gradually decreases, and ongoing research shows a significant increase in operating efficiency [11-13], virtualization technology will become increasingly attractive. The introduction of a variety of diversified software will increase the complexity and costs of installation and management. Developing web services on different platforms and supporting various servers will require technicians with a multi-platform environment background, but its cost is still less than NVP (N-version programming) [14] where different teams build complete solutions.

7. Conclusion
In this paper, we designed and implemented a java web service active defense system based on natural software diversity. By combining different components of the virtual web service software stack, different attack surfaces were generated. At the same time, we added dynamic scheduling software stack technology to introduce diversity and uncertainty into the java web service execution environment which makes java web services more resilient to attacks and effectively improves the security of web services. In further work, we will study the realization of automatic software diversification technology to obtain richer software diversity [15], such as using the mature java obfuscation tool yGuard [16] and ProGuard [17], and using runtime application self-protection technology’s [18] open source tools-openrasp [19].

Acknowledgments
This research is supported by National Key Research and Development Program of China (2018YFB0804003)

References
[1] China National Vulnerability Database, https://www.cnvd.org.cn/flaw/statistic
[2] Jajodia, S., Ghosh, A. K., Swarup, V., Wang, C., & Wang, X. S. (Eds.). (2011) Moving target defense: creating asymmetric uncertainty for cyber threats (Vol. 54). Springer Science & Business Media.
[3] Taguinod, M., Doupé, A., Zhao, Z., & Ahn, G. J. (2015) Toward a moving target defense for web applications. In: 2015 IEEE International Conference on Information Reuse and Integration pp. 510-517.
[4] Huang, Y., & Ghosh, A. K. (2011) Introducing diversity and uncertainty to create moving attack surfaces for web services. In Moving target defense. Springer, New York, NY. pp. 131-151.
[5] Thompson, M., Evans, N., & Kisekka, V. (2014) Multiple os rotational environment an implemented moving target defense. In: 2014 7th International Symposium on Resilient Control Systems (ISRCS) pp. 1-6.
[6] Thompson, M., Mendolla, M., Muggler, M., & Ike, M. (2016) Dynamic application rotation environment for moving target defense. In: 2016 Resilience week (RWS) pp. 17-26.
[7] Parno, B., McCune, J. M., Wendlandt, D., Andersen, D. G., & Perrig, A. (2009) CLAMP: Practical prevention of large-scale data leaks. In: 2009 30th IEEE Symposium on Security and Privacy pp. 154-169.
[8] Stamm, S., Sterne, B., & Markham, G. (2010) Reining in the web with content security policy. In: Proceedings of the 19th international conference on World wide web pp. 921-930.
[9] Wu J.X. (2016) Research on Cyber Mimic Defense, Journal of Cyber Security, vol. 1, no. 4, pp.1-10.
[10] Lala, J. H. (2003) Organically assured & survivable information systems (OASIS): foundations of intrusion tolerant systems.
[11] Menon, A., Cox, A. L., & Zwaenepoel, W. (2006) Optimizing network virtualization in Xen. In: USENIX Annual Technical Conference (No. LABOS-CONF-2006-003).
[12] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., ... & Vahdat, A. (2010) Difference engine: Harnessing memory redundancy in virtual machines. Communications of the ACM, 53(10), pp. 85-93.
[13] Lagar-Cavilla, H. A., Whitney, J. A., Scannell, A. M., Patchin, P., Rumble, S. M., De Lara, E., ... & Satyanarayanan, M. (2009) SnowFlock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European conference on Computer systems pp. 1-12.
[14] Avizienis, A. (1985) The N-version approach to fault-tolerant software. IEEE Transactions on software engineering, (12), 1491-1501.
[15] Larsen, P., Homescu, A., Brunthaler, S., & Franz, M. (2014) SoK: Automated software diversity. In: 2014 IEEE Symposium on Security and Privacy pp. 276-291.
[16] yGuard. https://www.yworks.com/products/yguard
[17] ProGuard. https://stuff.mit.edu/afs/sipb/project/android/sdk/android-sdk-linux/tools/proguard/docs/index.html
[18] Čisar, P., & Čisar, S. M. (2016) The framework of runtime application self-protection technology. In 2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI) pp. 81-86.
[19] Openrasp. https://rasp.baidu.com/doc/