International Journal of Security and Its Applications
Vol. 10, No. 11 (2016), pp.267-280
http://dx.doi.org/10.14257/ijsia.2016.10.11.22

Research on the Defense Method of Vtable Hijacking

Wang Zixiang, Shan Chun[*], Xue Jingfeng, Sun Shiyou and Hu Changzhen

Beijing Key Laboratory of Software Security Engineering Technique, School of Software, Beijing Institute of Technology, Beijing 100081, P.R.C
**sherryshan@bit.edu.cn*

Abstract

Memory corruption vulnerability is an oldest type of vulnerabilities in software vulnerabilities. Attackers typically use a technique called virtual function table hijacking to exploit memory corruption vulnerability. In this paper, we propose a defense method which extracting virtual function tables and virtual function call related location information from the binary program. Then instrumenting identifier on vtables or backuping the vtables' pointers to detect vtables' integrity. Finally, the defense method is verified by Firefox, Chrome, IE browsers. Experiments show that the method can fully and effectively defend the real-world virtual function table hijacking attack with the small performance overhead and good compatibility.

***Keywords**: Software security; Memory corruption vulnerability; Virtual function table hijacking attack*

Introduction

Memory corruption vulnerability is the oldest type of software security vulnerabilities, a recent study showed that 69% of browser vulnerabilities and 21% of operating system associated with such vulnerabilities [1]. Virtual function table hijacking attack is one of the main way to take advantage of the vulnerability of the memory [2], it through the pointer to tamper with the virtual function table or a virtual function table (pointer points to the virtual function table, the value of the pointer is the virtual function table's address), to achieve hijack program control flow type of attack.

SafeDispatch [3] is a defense solution against virtual function table hijack attack based on compiler. It costs around 7% on average by using SafeDispatch method and detection of the virtual function table costs about 30% or so. Cost is definitely too high in performance, and the latter cannot resist the virtual function table corruption attacks. VTGuard [6] is one of the defense solutions that Microsoft deployment in IE browser internal, to some extent, it can effectively alleviate the reuse of the virtual function table attacks, but it cannot resist the virtual function table injection attack and corruption to the virtual function table. DieHard [4, 5] provides a custom memory allocator, in this way can it defense a part of using vulnerability after being released and heap overflow vulnerabilities, so it can avoid virtual function table and virtual function table pointer in some cases are covered, it cost about 8% on average.

Hijacking virtual function table is one way to exploit memory corruption by modifying the virtual function pointer (which points to the start address of the virtual function tables) or virtual table thus seize control of the flow of the program. The traditional control flow hijacking attacks can be classified into the following three ways: Code corruption attacks, code injection attack, code reuse attack. Similar thereto, and attack associated virtual function table (called a virtual function table hijacking attacks) may also be classified as: reuse attack, corruption attack and injection attack of virtual function tables.

[*] Corresponding Author

1) The corruption attack

This type of attack will directly destroy the virtual function table that already exists in the content (virtual function pointer) to achieve their goals. Once the virtual function table is damaged, call any virtual functions associated with the virtual function table hijacking will occur. If the virtual function table attribute is writable, then this will be very prone to attack. Attackers virtual function table pointers can write directly to the virtual function table area coverage for their shellcode. Once associated virtual function is called, subsequent shellcode will be executed and achieve the aim of the assailant.

2) The injection attack

This type of attack will be false virtual function tables injected into the application of which, by overriding the virtual function table pointer (here referred to vfptr) to point to false virtual function tables. These false virtual function tables attacker injected area is located within a writable memory, so an attacker can easily control the fake table of contents. Due to the high reliability of injection attacks, it became a virtual function table hijacking attacks in the most widely used method. An attacker first build a fake virtual function table, filled with carefully constructed in the form of "virtual function pointer", and then covered with an object instance vfptr by using certain loopholes (such as the release after use loopholes) to point the fake watch. Therefore, any access to the vfptr pointer will be hijacked, and further selection of "virtual function pointer" will execute malicious code.

3) The reuse attack

This type of attack can also inject attack with virtual function table as to cover vfptr pointer. But this is not covered by the pointer points to virtual function tables attacker carefully constructed, but point to the virtual function table that already exists, or point to the memory of other existing data or code that has not been controlled by the attacker. Unlike virtual function table injection attack that launched the virtual function table reuse attack is relatively difficult, because only a small amount of data and code, and virtual function tables can be used as an object of attack. By reusing vfptr to the content of the short sequence of instructions to manipulate the stack, registers, and ultimately control the flow of the program, the implementation of the attacker preset target.

This paper proposes a defense method called VIP (the full name is the Virtual function table Integrity Protection), it can defense the Virtual function table hijacking attack by rewriting binary file written in c + + language and checking the Integrity of the Virtual function table. If the Integrity of the Virtual function table is destroyed, it proves that program has been subjected to the virtual function table hijacking attack, and the program will stop calling the virtual function immediately.

1. An Overview of the VIP

Overall description of VIP defense is shown in Figure 1, first put the binary applications written in c + + language as input, then automatically reverse analyzing it, the output analyzed is the virtual function related position information. Then taking binary file with the output of the automated analysis results as the next phase of the input, inserting identification number into the virtual function table located or back up the virtual function table pointer, and the virtual function pointer, then inserting check in the virtual function call, regarding it as instrumentation protection here. Last output the final protected applications.

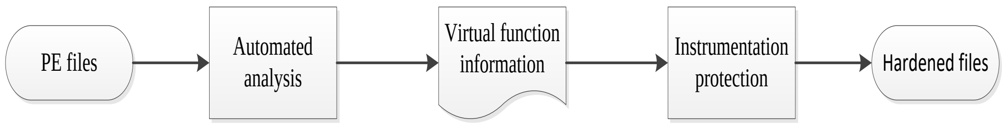

Figure 1. VIP Defense Method

From Figure 1, we can see that the VIP defense method proposed in this paper can be divided into two phases: automated analysis section and instrumentation protection stage.

Automated analysis of VIP method's main aim is to get enough information for the back of the binary instrumentation, these information includes the location of the object pointer, Virtual function table pointer offset in object space, The location of the virtual function instruction called, the offset of Virtual function pointer in the virtual function table, the register of accessing virtual function table pointer when called, *etc*. According to the information, we can take strategy to protect the target afterwards.

We use vExtractor, the unit which identifies virtual dispatches in binary code, It takes an executable as input whose vtable load instructions before virtual calls should be protected, disassembles it and generates a control flow graph (CFG). The disassembly is then transformed into an intermediate language (IL) to boil down the complex instruction set into a RISC-like syntax, while preserving the semantics and the CFG of the original code. Next, all addresses of indirect call instructions are extracted and defined as slicing criterions [7, 8]. vExtractor then performs backward program slicing on the IL to determine if an indirect call is a virtual call. It extracts all instructions which fulfill the low-level semantics of a virtual dispatch, thus, we retrieve virtual dispatch slices.

This paper focuses on instrumentation protection phase, the integrity of the VIP defense protection is the core idea in this paper.

A detailed description of the method of defense to protect the integrity of the stage. First, protecting the integrity of the defense strategy of the method are outlined as we see before, followed by a description of the detailed program integrity protection policies and procedures. Finally, to protect the integrity of this policy raised by the validity of the theoretical analysis.

At last, we proposed the process of defense method with the experiment and analysis to prove the validity and performance.

2. An Overview of the VIP Integrity Protection Strategy

VIP integrity protection strategy can be divided into two solutions.

2.1. ID Instrumentation Solution

Figure 2 shows the ID instrumentation solution and process as follows:

Create a read-only area and move all identifiable virtual function table into it

First, creating a read-only VIP area and copying all identified virtual function table to the area, if there is a legitimate virtual function table located in a zone can be written, move all recognition to the virtual function table to new read-only area. So, even if the original virtual function table is writable, new copy of the virtual function table will be set to read-only area.

Then, to move the virtual function table, it is necessary to know the size of the table. Yet for identifying the accurate size of the virtual function table in the binary file is a challenging work, by conservative estimates, should move around more than the size of a virtual function table several bytes in new area.

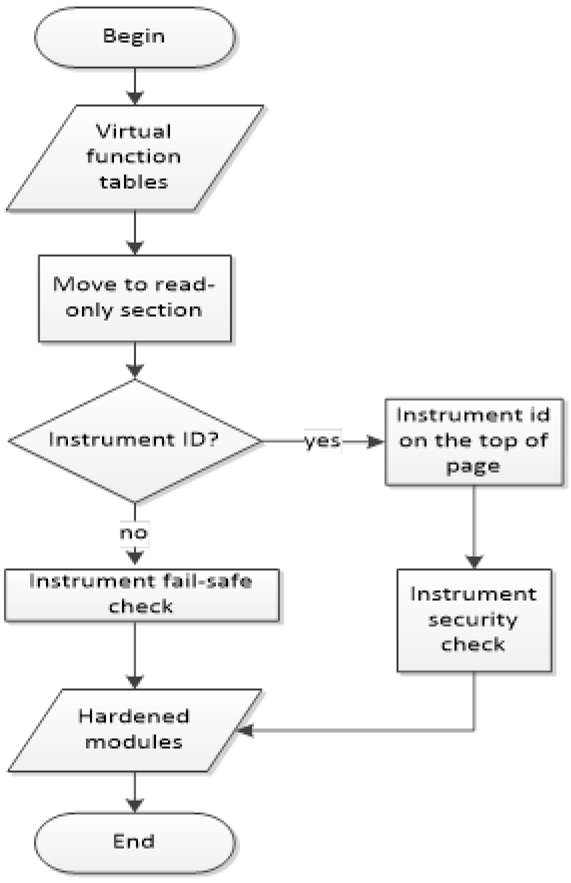
Begin
Virtual function tables
Move to read-only section
Instrument ID?
yes
no
Instrument id on the top of page
Instrument fail-safe check
Instrument security check
Hardened modules
End

Figure 2. Virtual Function Table ID Instrumentation

Add a unique ID number in the new area of a virtual function table header position

Use a special ID number to insert virtual function table to distinguish read-only area data and code. ID numbers are selected at randomly and different from any character existed in the pages of the starting position of the of read-only area, only need to insert a few pages of instrumentation ID number and do not need to change the layout of the virtual function table. Even if the attacker know the ID number, he can't use this ID in read-only area to counterfeit pages, he can only reuse the page in the new virtual function table area.

After the completion of (1) and (2), all references of the virtual function table will be modified for the virtual function table new address.

Figure 3 shows the data structure of ID instrumentation. Moved function tables are located in created read-only memory area, (vt-readonly), the read-only area is divided into multiple pages, each rectangle consisted of dotted line represents read-only memory area of each page, each page of the first part is the ID number of instrumentation, and these protected virtual function table exists in every page.

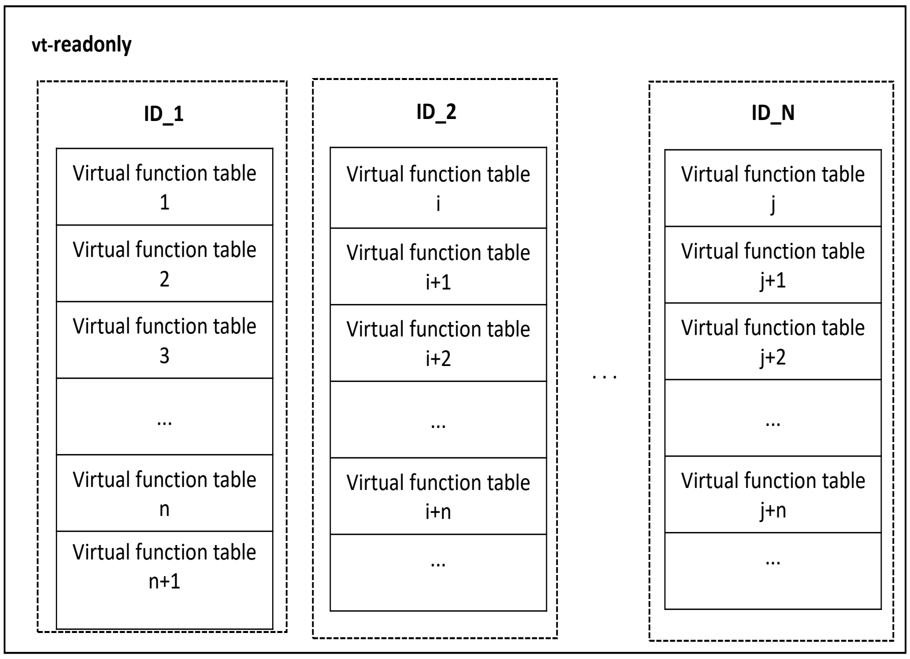

Figure 3. Id Instrumentation Data Structure

2.2. Id Instrumentation Sheck

When a virtual function is called, it will be implicit access to the corresponding virtual function table to get the target virtual functions, and we test the integrity of the virtual function table at this time.

1. Security check

Verifying the integrity of the virtual function table process is as follows:

(1) Matching the target ID of a virtual function table

(2) Detecting whether the virtual function table is read-only

If any test fails, the control process will be transferred to the previously defined error handler, to prevent further execution procedures. Checking process is shown in Figure 4.

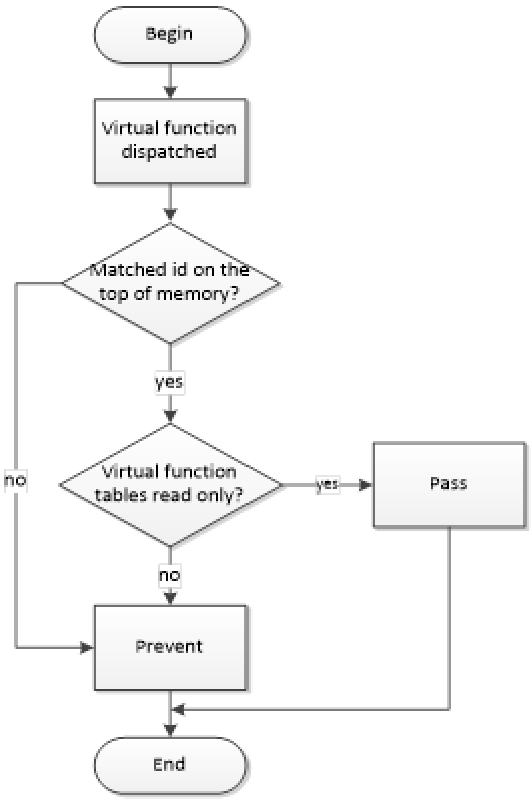
Begin
Virtual function dispatched
Matched id on the top of memory?
yes
no
Virtual function tables read only?
yes
Pass
no
Prevent
End

Figure 4. Security Check

2. The fail-safe check

Safety monitoring design idea is in a relatively ideal case, but in the actual cases, it cannot protect all modules at the same time. For example, some modules belong to the operating system and cannot be modified by user applications. As for the module without reinforcement, we can't give the virtual function table instrumentation ID, when accessing to the virtual function table, the security check work in front of the virtual function call bottom will fail and cause false alarms. In order to solve this problem, the VIP also uses the fail-safe check effectively to remove these false positives.

Perform fail-safe check will only check part of the virtual function table properties, specific process is as follows:

(1) To check whether the current module was reinforced by the ID instrumentation solution

By security check in the process "header whether there is a matching ID number" to continue testing, when there does not exist matching ID number that has been detected, the exception handler that ID does not match will test whether the current module was reinforced by the identity of the instrumentation project in this paper. In order to identify weather a module was reinforced by the identity of the instrumentation project in this paper, it will traverse the executable file at runtime PEB analytic information about current module, if we found memory read-only area called vt-readonly, and the area of the first four bytes are used to assign to ID number of a virtual function table, and we can draw the conclusion: it has been reinforced by identity of the instrumentation solution in this paper. If so, it will stop the flow of control report attacks happening. Otherwise, it will not stop the control flow

(2) To check the property of virtual function table

If it has not been hardened by ID instrumentation solution, it returns to the original code further and then check whether virtual function table is read-only. If so, it allows the dispatch. The process is shown in Figure 5.

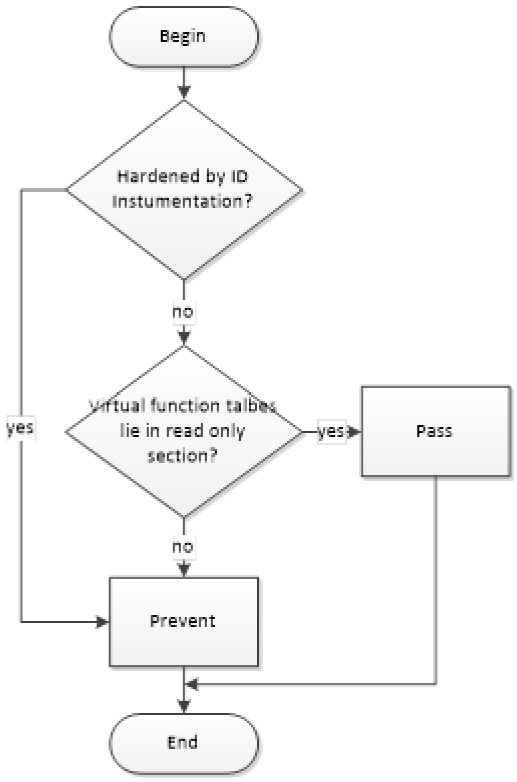
Begin
Hardened by ID Instumentation?
no
yes
Virtual function talbes lie in read only section?
yes
Pass
no
Prevent
End

Figure 5. Fail-Safe Check

2.3. Backup Instrumentation Solution

We have mentioned module that has not been improved in section 2.2, most of these modules belong to operating system. In a few cases, these module has not been improved of virtual function table cannot be moved to the specified read-only memory area, they are always in a writable state. In Figure 5 when the process judges whether a virtual function table located in read-only memory area, even if the virtual function table did not exist in read-only zone, we cannot decided immediately that it was hijacked, so it is not reasonable that we stop this part of the apparently normal virtual function calling. So we need to test the integrity of the virtual function table before and after the call by backing up virtual function table and virtual function table pointer.

Figure 6 shows such a module instrumentation project that was backed up, the process is as follows:

Back up data of virtual functions

Finally back up data of the virtual function, the content that is backed up is virtual function table and virtual function table pointer and the virtual function table pointer and the corresponding relationship between virtual function table and virtual function table pointer.

Put the backup content into read-only area

In the process of data backup, the key issues need to consider is backing up the data in the new virtual function table read-only (vt-readonly) in the area. The backup data is write operation, when backup operation finished, it will change the attribute of backup area into read-only.

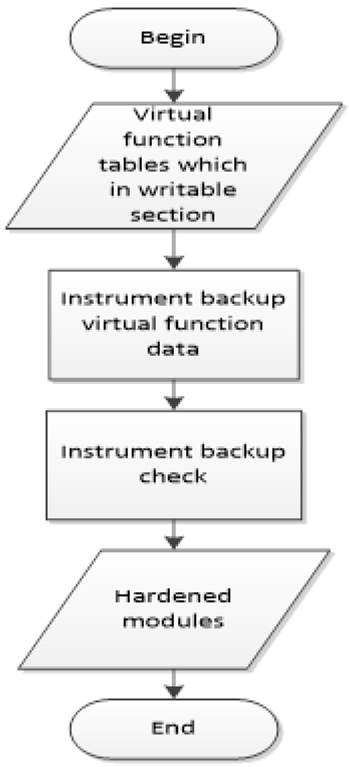
Begin
Virtual function tables which in writable section
Instrument backup virtual function data
Instrument backup check
Hardened modules
End

Figure 6. Virtual Function Table Backup Instrumentation

Figure 7 shows the data structure diagram of backup instrumentation, all backup data is stored in read-only memory page. All of the virtual function table pointer that are backing up are called "backup virtual function table pointer set"; Backing up all virtual function pointer is to back up the virtual function table, so backup all the virtual function pointer is called "backup virtual function table set". We should note that the arrow in this picture only represents corresponding relation between the virtual function pointer and the virtual function table, backup also includes corresponding relation.

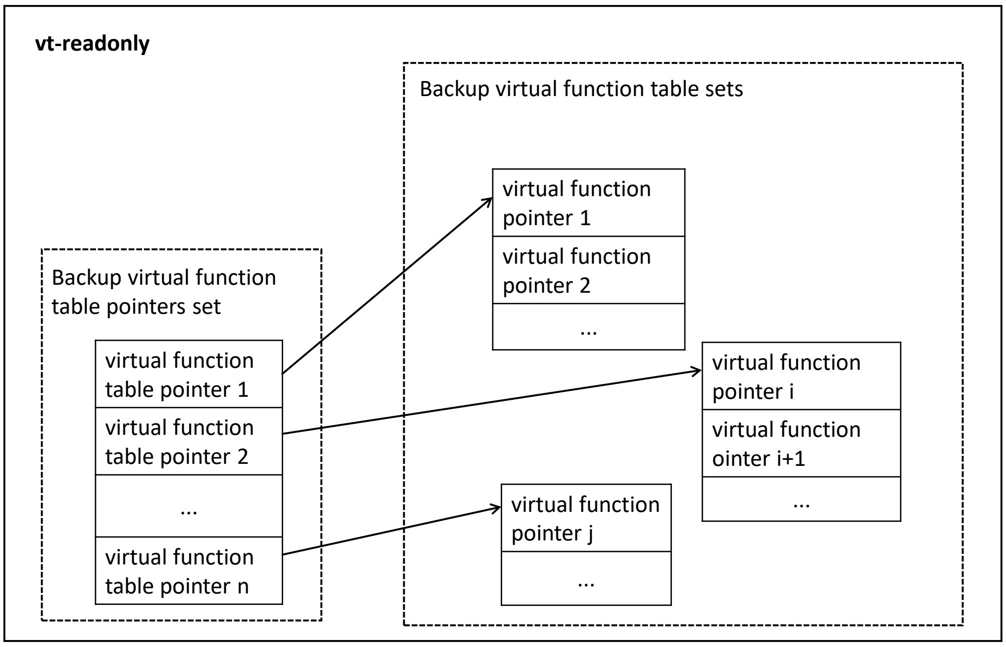
vt-readonly
Backup virtual function table sets
Backup virtual function table pointers set
virtual function table pointer 1
virtual function table pointer 2
...
virtual function table pointer n
virtual function pointer 1
virtual function pointer 2
...
virtual function pointer i
virtual function ointer i+1
...
virtual function pointer j
...

Figure 7. Backup Instrumentation Data Structure

2.4. Backup Instrumentation Check

Before calling a virtual function, we need to check data integrity of virtual functions. The steps are as follows:

(1) Test matching virtual function table pointer

If they can match, execute (2).

(2) Test matching virtual function pointer

If any test cannot past, the program is considered hijacking attack by virtual function table, terminate immediately the current call, and return error information. The process is shown in Figure 8.

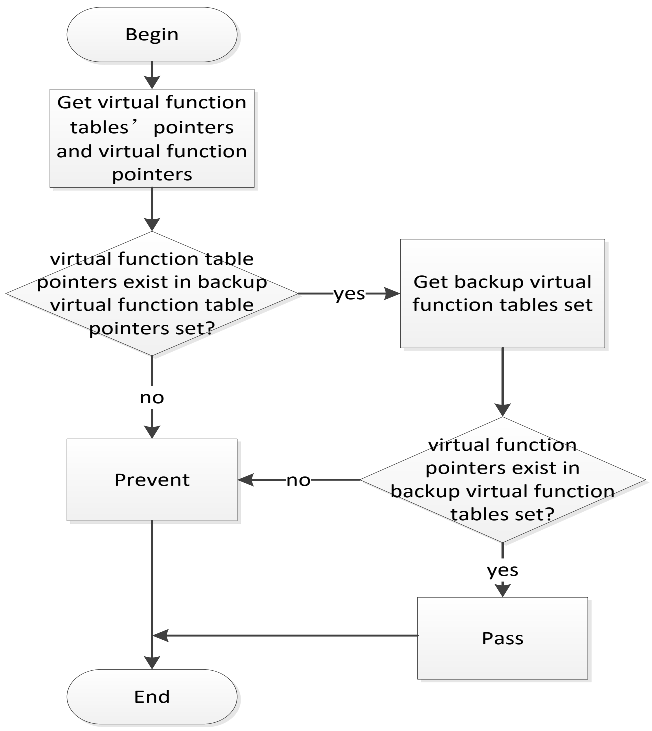
Begin
Get virtual function tables' pointers and virtual function pointers
virtual function table pointers exist in backup virtual function table pointers set?
yes
Get backup virtual function tables set
no
Prevent
no
virtual function pointers exist in backup virtual function tables set?
yes
Pass
End

Figure 8. Backup Instrumentation Check

2.5. The Analysis of Effectiveness Theory

Restrictions virtual function table is read-only guarantees cannot be implemented destroy attacking virtual function tables, to backup and verify the virtual function table to ensure the effective detection of damage to attack the virtual function table; the restrictions virtual function table is read-only to ensure unable to implement the virtual function table injection attacks, while the virtual function table pointer backup and verification to ensure that the virtual function tables for injection attack detection; through stub ID number to distinguish the virtual function tables and other data, code, to ensure the virtual function table reused effectively detect attacks. Although the attacker can reuse the virtual function table read-only memory area that already exist to bypass the VIP, but in an application usually does not have so many virtual function table for virtual function table that already exists launch further attacks is very difficult, the attack surface is relatively small. When an attack is detected appears, it will terminate the current function call.

VIP simultaneously by addressing not enhanced in the read-only area module detection methods and writable area within the virtual function table detection methods for solving different types of module compatibility testing and complete protection to all modules. Data backup and virtual function and virtual function tables are identified stub in a read-only memory pages, which makes the whole memory even if the attacker to read data discovered the existence of protection strategies, could not be modified.

3. Experiments and Discussion

Experiment is operated in Windows 7 of 32-bit operating system environment. Using implementation tools Pintool that is based on Pin platform completes plugging instrumentation of binary files.

1) Runtime overhead in browser

We test the performance of the Chrome and Firefox browser, this article uses the Kraken and Sunspider two commonly used browser benchmark to evaluate the performance of these browsers. Things shown in Figure 9 and 10 is the performance overhead after using VIP, the sum of all the browser's spending is about 3.3%, Firefox and Chrome, the average costs are 3.4% and 3.4% respectively.

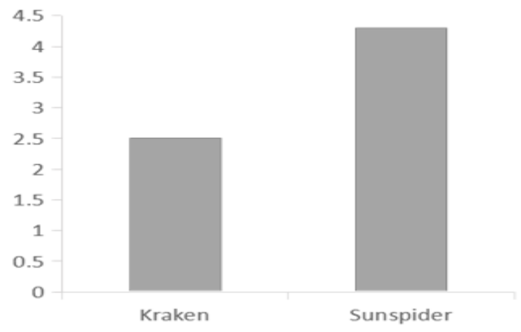

Figure 9. Performance Overhead On Firefox

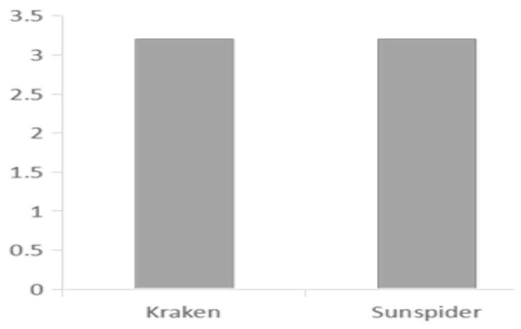

Figure 10. Performance Overhead On Chrome

2) The performance analysis when running

After using the VIP, performance overhead is relatively smaller than the traditional defense methods, such as memory allocation solution DieHard [4, 5] will introduce 8% overhead, and SafeDispatch [3] using virtual function table will introduce 30% overhead.

3) The vulnerability and defense in the real world

In order to assess the effectiveness of the VIP, we select 6 virtual function table hijacking attack vulnerability, as shown in Table 1. These vulnerabilities aim at browser applications, such as Internet explorer, Firefox browser. They pour forge virtual function table into memory and eventually start a virtual function table injection attacks. Chrome has a few public available vulnerabilities, so we did not test it.

Table 1. Defense Real World Vulnerabilities

CVE-Number	APP	Type	effectiveness
CVE-2013-3897	IE8	Use-after-free	effective
CVE-2012-1876	IE8	Heap-overflow	effective
CVE-2013-3205	IE8	Use-after-free	effective
CVE-2013-2551	IE10	Use-after-free	effective
CVE-2012-0469	FireFox6	Use-after-free	effective
CVE-2013-0753	FireFox17	Use-after-free	effective

We take IE10 as an example, by extracting the core module of the objective application, we use VIP adding instrumentation protection to the module of IE browser, finally the original module have been substituted for these protected module, which means completing the defensive deployment. After deploying these protected modules, makes the objective application access the malicious URL links of above vulnerabilities. Then we analyze the attack codes of vulnerability CVE - 2013-2551, as shown in Figure 11.

```
this.write32( fake_securitymanager, fake_securitymanager_vtable );
this.write32(script_engine_addr + 0x21c, fake_securitymanager);
……
this.write32( fake_securitymanager_vtable + 0x14,
this.searchBytes( [0x8b, 0xe5, 0x5d, 0xc2, 0x08], jscript9_code_start, jscript9_code_end ) );
/* mov esp, ebp; pop ebp; ret 8; */
this.write32( fake_securitymanager_vtable + 0x10,
this.searchBytes( [0x8b, 0xe5, 0x5d, 0xc2, 0x04], jscript9_code_start, jscript9_code_end ) );
/* mov esp, ebp; pop ebp; ret 4; */
```

Figure 11. Virtual Function Table Injection Attack Codes

By the code, we can realize that the way of attack it uses is the virtual function table injection attacks. Here we test unprotected IE Web browser and protected Internet explorer browser respectively.

Run attack codes in the unprotected browser, click rejection in the following pop-up safety tips, then Windows' calculator program is opened, which is the effect after the successful attack; Running attack code in the protected browser, in the following pop-up safety tips we click refection, the calculator program of Windows isn't open, the process of IE ends, which means preventing further attacks.

Through modifying the code of virtual function table injection attack, we can get the code of virtual function table attack. In this way, we can implement virtual function table attack in browser. We do experiment again to achieve the effect which is same with the virtual function table injection attacks in browser, we don't repeat it here.

4) The compatibility analysis

Compatibility analysis is binary compatibility and modular support VIP's. Binary Compatibility is when we revised the module, and whether they cannot modify the modules to work together, no problem; and modular support refers to any a single module, if it cannot rely on other modules only information collected by the module itself can be implemented to protect the VIP.

The results of two experiments show that the inspection of VIP will prevent the exploit from happening, in other words, the VIP can protect real browser from the virtual function table hijacking attacks. It is important to note that when the Internet explorer is protected by VIP, web page can still display properly, and safety tips to make to the pop up to allow users/refused to choose Internet explorer will still be able to normal processing.

4. Conclusion

VIP defense method successfully solves the contradiction between the performance and effectiveness of defense. Comparing with the traditional defense methods, VIP can protect every possible attack module at a relatively low performance overhead. The experiment proved that VIP can effectively prevent the virtual function table hijack attacks from the real world, and reduce the false negative. VIP implement their own protection solution on different kinds of modules at the same time, according to different types of modules in different state of the virtual function table, VIP can detect correctly, it won't produce false positives. Detection work of VIP is compatible with module protected by different solutions.

Acknowledgements

This work was supported by the National Key Research and Development Program of China (Grant No. 2016YFB0800700) and the Scientific Research Project of Beijing Institute of Technology (Grant No. 2016CX02027).

References

[1] Caballero J, Grieco G, Marron M, et al. Undangle: early detection of dangling pointers in use-after-free and double-free vulnerabilities[C]// Proceedings of the 2012 International Symposium on Software Testing and Analysis. ACM, （**2012**）:133-143.
[2] Szekeres L, Payer M, Wei T, et al. SoK: Eternal War in Memory [J]. Security & Privacy IEEE, （**2013**）, 12(3):48-62.
[3] Jang D, Tatlock Z, Lerner S. SAFEDISPATCH: Securing C++ Virtual Calls from Memory Corruption Attacks[C]// Network and Distributed System Security Symposium. （**2014**）
[4] Berger E D, Zorn B G. DieHard: probabilistic memory safety for unsafe languages [J]. Acm Sigplan Notices, （**2006**） 41(6):158-168.
[5] Novark G, Berger E D. DieHarder: securing the heap. [C]// ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, Usa, October. （**2010**）:573-584.
[6] Miller M R, Johnson K D, Burrell T W. Using virtual table protections to prevent the exploitation of object corruption vulnerabilities: U.S. Patent 8,683,583[P]. （**2014**）
[7] Binkley, David, and M. Harman. "A Survey of Empirical Results on Program Slicing." Advances in Computers 62.11-12**(2004)**:105-178.
[8] J. Silva. A vocabulary of program slicing-based techniques. ACM Computing Surveys (CSUR), 44(3):12, **(2012)**