Poster: Unider: Exploit Attack Emulator Armed with State-of-Art Exploit Techniques

Yu Ding
Peking University
Beijing, China
dingelish@gmail.com

Chao Zhang, Tao Wei
UC Berkeley
Berkeley, CA, USA
{gausszhch, lenx.wei}@gmail.com

***Abstract*—We have many security enforcement techniques deployed in commodity systems. But we lack of universal benchmark program to evaluate these techniques. This poster introduces working project named 'unider ', a universal exploit attack emulator. Unider works on recent Windows systems and supports state-of-art exploit techniques. By using unider , researchers can evaluate their defense techniques quickly and conveniently.**

I. Introduction

Exploit attack detection and prevention techniques (such as [1]–[4], [8], [11]) are well studied. Many of these techniques are now deployed in commodity systems. To test these tools, researchers use various penetration platforms such as Metasploit [7] to launch attacks. Also, researchers spend lots of time collecting vulnerable software and configuring testbeds. We need a universal benchmark program to evaluate exploit attack detection and prevention tools, instead of using piecemeal, ad hoc exploits.

To better evaluate defense techniques, we design unider , a universal exploit attack emulator. Unider generates malicious input to itself and hi-jacks the control flow to injected shellcode. Unider has the following characteristics:

- Unider supports current Windows operating systems, including both 32 bit and 64 bit version of Windows 7/Vista/XP. In future Unider will be ported to Linux and OS X.
- Unider emulates state-of-art exploit techniques, such as heap-spraying code injection, Kill-Bill exploit [5] and DEP/ASLR bypassing.
- Unider contains V8 and SpiderMonkey. These two script engines are used to emulate JIT related code injection techniques.
- Unider suits analysis tools better. First, Unider provides an option to select the type of input source (file, network, user input). This option helps taint analysis based tools better understanding the whole exploit attack. Also unider provides a built-in function to launch all supported attacks continuously.

II. Unider Design

The structure of unider is shown in Figure 1. Unider has two parts: exploit generator and vulnerable components. Vulnerable components leak essential information about themselves

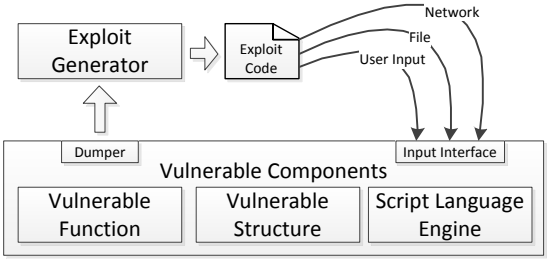

Fig. 1: **General Structure of unider**

to the exploit generator and the exploit generator generates exploit code. The exploit codes are sent to vulnerable components to trigger the vulnerability and execute payloads.

Unider describes an exploit attack in 7 dimensions: target/location/technique/attack code/function/code injection technique/input method. First five of these dimensions are similar to RIPE [10] and the last two dimensions are new. Code injection techniques contain state-of-art code injection skills. These code injection skills are manually collected from : (a) Metasploit repository, (b) hackers' forums and (c) some other online public exploit databases such as exploit-db [6]. Input method dimension indicates the routine of passing exploit code to vulnerable components. This is the special design for convenience of taint based analysis tools. Taint based analysis tools need to mark inputs as tainted precisely. However, most practical programs get input from multiple channels such as file/network/user inputs. The input method dimension is designed to help these taint based tools mark the input bytes precisely.

Unider provides JIT based code injection techniques. These code injection techniques can emulate JIT spraying perfectly. With the help of vulnerable components, essential information such as injected code pointers are leaked to the exploit generator. The exploit generator then generates exploit codes which hi-jack the control flow of vulnerable components to the injected JIT codes. This kind of attack breaks the control flow integrity of JIT dynamic codes and can evade from current CFI based detection/protection.

Here we give an example about how Unider emulates an attack by using JIT engine. Latest SpiderMonkey engine supports JIT dynamic compiling and `asm.js` at the same time.

Control flow transfer between `asm.js` dynamic code/other JIT dynamic code/native code are implemented in variety ways. Under some certain circumstances, the control flow transfers between dynamic codes and static codes requires a pointer table locates at a *f*ixed offset in the global data section. This pointer table is writable and exposed to the attacker in runtime. By tampering this pointer table, the attacker can easily trigger injected codes. This kind of attack hi-jacks the control flow transfer between static codes and dynamic codes. Unider provides a information leakage vulnerability, which leaks the offset and base address to the exploit generator. In this way, the exploit hi-jacks the control flow by tampering the pointer table. This kind of attack is not well-studied in both academic area and hackers community.

III. Related Works

Only a limited number of this kind of benchmark tools are released to public. Zhivich et al. collected 55 vulnerable programs [12] and evaluated some defense techniques on these programs. However, these testbeds are not released to public and the experiments are not repeatable. RIPE [10] and its ancestor [9] provide similar solutions. RIPE describes an exploit attack in five dimensions (target, location, technique, attack code and function) and can launch 850 different attacks. However, RIPE can only be built in Ubuntu 6.06, which is a 7-year-old operating system with few security enhancements. Besides, RIPE only includes conventional exploit techniques. Modern exploit techniques such as JIT-spraying, Kill-Bill, DEP/ASLR bypassing techniques needs to be included in such benchmarks.

IV. Conclusion

In this poster, we show unider , an exploit attack emulator armed with state-of-the-art exploit techniques. Unider can help security researchers evaluate their defense techniques. Also Unider can act as a benchmark program for defense/mitigation techniques. Unider is a working project and we will soon release the code to community.

References

[1] Akritidis, P., Cadar, C., Raiciu, C., Costa, M., and Castro, M. Preventing Memory Error Exploits with WIT. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (SP'08)* (Oakland, CA, May 2008), IEEE, pp. 263–277.

[2] Anagnostakis, K. G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., and Keromytis, A. D. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14th USENIX Security Symposium (USENIX'05)* (Baltimore, MD, 2005), pp. 129–144.

[3] Cui, W., Peinado, M., and Wang, H. J. Shieldgen: Automatic data patch generation for unknown vulnerabilities with informed probing. In *Proceeding of the 28th IEEE Symposium on Security and Privacy (SP'07)* (Oakland, CA, 2007), In In Proceedings of 2007 IEEE Symposium on Security and Privacy.

[4] Diebold, P., Hess, A., and Schäfer, G. A Honeypot Architecture for Detecting and Analyzing Unknown Network Attacks. *Kommunikation in Verteilten Systemen (KiVS)* (2005), 245—-255.

[5] Eclipse, S. kill-bill : Microsoft asn.1 remote exploit for can-2003-0818 (ms04-007).

[6] Exploits Database by Offensive Security. http://www.exploit-db.com/.

[7] Rapid 7. Metasploit penetration testing platform, http://www.metasploit.com/.

[8] Wang, X., Jhi, Y.-C., Zhu, S., and Liu, P. STILL: Exploit Code Detection via Static Taint and Initialization Analyses. In *Proceedings of Annual Computer Security Applications Conference (ACSAC'08)* (Anaheim, CA, Dec. 2008), IEEE, pp. 289–298.

[9] Wilander, J., and Kamkar, M. A Comparison of Publicly Available Tools for Dynamic Buffer Overflow. *10th Annual Network and Distributed System Security Symposium (NDSS'03)*.

[10] Wilander, J., Nikiforakis, N., Younan, Y., Kamkar, M., and Joosen, W. RIPE: Runtime Intrusion Prevention Evaluator. In *Proceedings of the 27th Annual Computer Security Applications Conference on - ACSAC '11* (New York, New York, USA, Dec. 2011), ACM Press, p. 41.

[11] Zhang, C., Wei, T., Chen, Z., Duan, L., Szekeres, L., McCamant, S., Song, D., and Zou, W. Practical control flow integrity and randomization for binary executables. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2013), SP '13, IEEE Computer Society, pp. 559–573.

[12] Zhivich, M., and Leek, T. Dynamic buffer overflow detection. In *Workshop on the Evaluation of Software Defect Detection Tools* (2005).