Survey on malware evasion techniques: state of the art and challenges

Jonathan A.P. Marpaung*, Mangal Sain*, Hoon-Jae Lee*

*Department of Ubiquitous IT, Graduate School of General, Dongseo University,
San 69-1, Jurye 2-dong, Sasang-gu, Busan 617-716, Korea

jonathan@spentera.com, mangalsain1@gmail.com, hjlee@gdsu.dongseo.ac.kr

***Abstract*— Nowadays targeted malware attacks against organizations are increasingly becoming more sophisticated, damaging, and difficult to detect. Current intrusion detection technologies are incapable of addressing many of the newer malware evasion techniques such as return-oriented programming and remote library injection. This paper presents a survey on the various techniques employed in malware to evade detection by security systems such as intrusion detection and anti-virus software. The evasion techniques we cover include obfuscation, fragmentation and session splicing, application specific violations, protocol violations, inserting traffic at IDS, denial of service, and code reuse attacks. We also discuss mitigations such as sandboxing, session reassembly, data execution prevention, address space layout randomization, control flow integrity, and Windows 8 ROP mitigation. We also compare evasion techniques with an analysis on the sophistication of the attack, challenges or difficulty to detect, and degree of impact.**

***Keywords*— evasion techniques, intrusion detection systems, return oriented programming, data execution prevention, operating systems security**

I. Introduction

In recent years, organizations are seeing the importance of deploying intrusion detection systems (IDS) and security information & events management (SIEM) systems. The reasons being both to adhere to regulatory compliance as well as to minimize losses due to fraud, compromise of confidential information, and harm to reputation. An intrusion detection system (IDS) is a device or software application that monitors network and/or system activities for malicious activities or policy violations and produces reports to a Management Station [1]**.** IDSs can be either network or host based and commonly employ detection methodologies such as signature-based detection, anomaly-based detection, and stateful protocol analysis.

Software and hardware vendors are also introducing technologies to detect and prevent malware infection. For example data execution protection (DEP), address space layout randomization (ASLR), structured exception handler overwrites protection (SEHOP), and mandatory integrity control (MIC). These are some of the security features available in modern operating systems, however many of these techniques can be bypassed with today's evasion techniques, such as return-oriented programming.

We restrict the scope of this paper to discuss malware intrusion detection evasion techniques and the challenges in preventing or mitigating them. Therefore the development of the malicious payloads or exploits does not fall under this discussion. Based on our research we can conclude that vendors are lagging behind in developing adequate protections against sophisticated attacks.

The rest of the paper is organized as follows. Section II introduces the evasion techniques commonly used in malware development. Mitigation schemes and an analysis of the evasion techniques are discussed in section III and IV respectively. And finally in section V we share our conclusion.

II. Evasion Techniques

Malware development has gone a long way from the days of the first boot sector viruses. As vendors developed patches and defenses, malware writers have continued to find ways to outsmart them. In this section we discuss some of the classic and newer techniques of eluding malware detection mechanisms.

A. Obfuscation

The classic method of concealing the attack payload of malware is still commonly employed today, albeit with more advanced variants than in the early days of IDS evasion. These methods are designed to mostly overcome the simplest methods of string matching in signature-based systems. This can be achieved by concealing the payload in a manner that will not be detected by the IDS, but will still be decoded and executed by the target system.

1) Data encoding and string manipulation

The simplest example of manipulating the string of a command can be demonstrated by an attempt to access the /etc/passwd file on a Unix system. The rigid Snort signature shown in Figure 1 can easily be bypassed by slightly altering the command to /etc/init.d/../../\passwd and many other examples. Working further on this idea we can combine string manipulation with character substitution by changing the data encoding with Hex, Unicode, or UTF-8.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-MISC
/etc/passwd";flags: A+; content:"/etc/passwd"; nocase;
classtype:attempted-recon; sid:1122; rev:1;)
```

Figure 1. A simple Snort signature for detecting attempts to access the /etc/passwd file.

An ordinary string matching database cannot be used to handle these variants as there are too many possibilities. However IDS devices have handled most of these evasion techniques by better understanding of the protocols and performing necessary protocol conversions before comparing payloads to the string. However poorly defined signatures, which might take the form of rushed patches for zero-day exploits, makes this evasion technique still applicable today.

2) Encrypted Sessions

Signature analysis is made difficult for man-in-the-middle (MiTM) network IDS implementations where a secure context, for example SSL, exists between the client and server. As it is commonly assumed that secure sessions are initiated by authorized users, the absence of mechanisms such as mutual authentication will allow attackers to evade network IDS detection by harnessing features of the security system itself. As the IDS cannot read encrypted traffic, the payloads pass through unnoticed.

3) Polymorphic Code

Polymorphic code is a method now commonly implemented in malware that uses a polymorphic generator to mutate the code while keeping the original algorithm intact. A typical implementation of a polymorphic code is to encrypt malware and include the encryptor/decryptor within the code. The malware, which often does not contain enough a unique enough pattern to be used as a signature, after bypassing the IDS is loaded by the target system OS into the program memory area where it will decode itself before the actual payload is executed.

The Mutation Engine, developed by Dark Avenger, is one example of a polymorphic generator introduced in 1991, which has been incorporated in various viruses [2]. The malware's dynamic nature makes it difficult for IDS detection without letting the malware decode itself first. However research for identifying polymorphic shell code by searching for a no-op characters, was discussed in a Next Generation Security Technologies' whitepaper Polymorphic Shellcodes vs. Application IDSs [3].

B. Fragmentation and Session Splicing

Fragmentation and session splicing are network level evasion techniques that are harder to defend against than string obfuscation. Packet fragmentation is a feature of IP protocol for handling of packets entering networks with different Maximum Transmission Unit (MTU) limits. This feature can be exploited by attackers to generate attack payloads that don't conform to standard TCP/IP fragmentation. Session splicing is the delivery of a malicious payload over multiple packets in a session. A spliced session can be recognized as a continuous stream of small packets.

Ptacek and Newsham exposed these methods of eluding intrusion detection systems in their paper “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection,” in 1998 [4]. These techniques defeats signature-based analysis by splitting attack payloads over several packets. The payload can be delivered using strategies such as modifying packet size, sequencing, and timing. Without session reconstruction, simple pattern matching detection can be trivially bypassed.

Experiments on network testbeds have shown that different IDS devices handle session splicing techniques with varying degrees of success [5]. Today's IDS devices do some reassembly of fragments before comparison, however there are two key constraints regarding proper reassembly that can be exploited in eluding detection. First is that the IDS needs to fully reassemble the packets in memory before comparing them to the string. This condition creates a resource constraint. Considering that most network IDSs are placed in choke points, waiting to fully reassemble packets creates a performance issue. Secondly, the IDS needs to understand how the packets will be reassembled by the target host. Kevin Timm discusses several evasion attacks based on these premises [6]:

1) Fragmentation Overlap

Fragmentation overlap is an evasion technique that works by having one fragment overwriting data from another . In the example of Figure 2 the packets are assembled so that packet #2 overwrites the last byte of packet #1 so that when they are re-assembled on the destination host the string is GET x.ida?(a buffer overflow attack).

```
Packet #1 GET x.idd   Packet #2 a.?(buffer overflow here
```

Figure 2. Fragmentation overlap example.

2) Fragmentation overwrite

Fragmentation overwrite is similar to overlap except that the destination system a complete fragment overwrites another fragment.

```
Packet #1 GET x.id   Packet #2 somerandomcharacters   Packet
#3 a?(buffer overflow)
```

Figure 3. Fragmentation overwrite example.

3) Fragmentation Time-outs

Fragmentation time-outs are take advantage of the fact that network devices maintain a limit on how long fragments are stored in memory. Most systems will time-out an incomplete fragment stream in 60 seconds. If the IDS does not hold on to the fragment for the full 60 seconds, it is possible to send packets as follows:

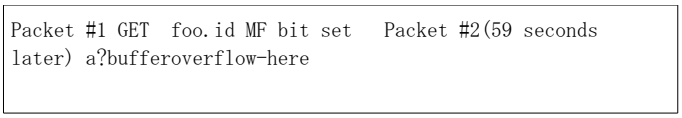

Figure 4. Fragmentation time-out example.

If the IDS does not store the initial fragments for a full 60 seconds it is possible to evade the IDS. Most network IDS in the market today can detect this technique with proper configuration.

Fragmentation can be combined with some other network techniques such as using expiring TTL values (inserting traffic at the IDS). To use this, the host being attacked needs to be enough hops behind the IDS that the IDS can see a packet that expires before reaching the destination host.

C. Application Specific Violations

Some applications implement protocols in different ways, not entirely enforcing RFC requirements. This may cause the same requests to be made using workarounds that are inherent in the implementation. An example in HTTP formatting, some web servers allow forward slash and backward slash. If the signature was designed to only check for forward slash patterns, then a simple backward slash would suffice to evade detection. Likewise, the GET and POST methods can be used interchangeably.

The Windows SMB implementation is another example that allows random data to be inserted between commands. As this principle can be applied to all types of applications, protocol decoders should be flexible enough to account for these violations [7].

D. Protocol Violations

This evasion technique violates RFC defined protocols in a way the target host will handle differently than the IDS. One commonly discussed example is the Remote Procedure Call (RPC) protocol and its inherent design flaws. RPCs are security risks as they allow other computers connected to a network to execute commands.

Joseph Taylor discussed evasion methods using Denial-of-Service (DoS) attacks against RPC servers that also can be escalated to DoS of the IDS [8]. This technique exploits the fact that the RPC service (portmap/rpcbind) will not begin processing a request until it knows it has the last data fragment. By setting “Last Fragment/Fragment Length” value to null portmap/rpcbind will deny connections to any other RPC-based application that needs its services. If the IDS is configured attempt to decode such conversations, it will also use up system resources.

Although only discussing one technique, Taylor noted that there are an infinite number of ways to evade IDS detection using the weaknesses inherent in the RPC protocol.

E. Inserting Traffic at IDS

Another technique also discussed by Ptacek and Newsham, inserting traffic at the IDS involves sending packets that will be seen and processed by the IDS, but not by the target host. This effectively causes a different state of connection to exist between the IDS and target system. One method of implementing this technique is by sending packets that have a time-to-live (TTL) that will allow them to reach the IDS, but expire before reaching the target system. In this manner, similar to the reassembly of fragmented packets, the IDS will fail to properly reassemble the packets and perform a string match.

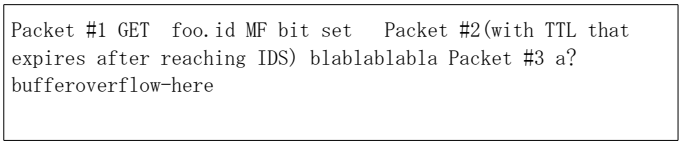

Figure 5. Insertion example.

F. Denial of Service

DoS attacks against are particularly tricky to deal with because IDSs are usually are the most resource intensive devices on the network. Unlike most other hosts on the network that only need to be concerned with traffic destined for them, IDS devices must monitor all network activity. CPU resources are used to read packets. Memory is required to store connection states, assemble queues, and maintain buffers for pattern matching. Logs must be stored on disk and network interfaces must be able to accept all of the network's raw traffic. This creates makes IDS devices more prone to starvation attacks. Since an IDS is generally fail-open, if it is overwhelmed by a DoS attack, further attacks on the network will not only be undetected, but also pass through unhindered.

Exploiting this vulnerability, attackers attempting DoS attacks can employ various methods to overwhelm the device's CPU, memory, and network capacity. This can be accomplished by triggering the device to utilize inefficient algorithms in their worst case conditions, cause the system to allocate memory for a prolonged amount of time, and flooding the network with more traffic than the network interface can handle.

G. Code Reuse Attacks

The strength of these attacks lies in the fact that they don't need to inject any code. Using these techniques, the traditional method of trying to detect or defend against injected malware signatures is circumvented because locally running processes seemingly make legitimate system requests.

1) Return-into-libc

Return-into-libc attacks were demonstrated by Solar Designer in 1997 as a method of getting around a non-executable stack [9]. Such attacks were created as a response to code injection defences by reusing existing executable code in a running process. An attacker would change the return address on the stack to point to another function. The standard C-language library, libc, is usually the target as it is loaded into every UNIX program and provides useful system calls for the attacker. However this technique is limited to straight-line function calls to the C-standard library, making it less flexible than injected malware.

2) Return-Oriented Programming:

Return-Oriented Programming (ROP) is a generalized version of return-into-libc introduced by Shacham in 2007 [10] to demonstrate where the widely depolyed W-xor-X model fails to provide adequate protection. W-xor-X or Data Execution Protection (DEP) is a memory protection policy for operating systems that marks memory as either writable or executable, but not both. ROP creates malicious computation by linking together small code snippets in an existing program's address space. Each snippet ends with a 'ret' instruction, which allows an attacker who controls the stack to chain them together. Return-oriented attacks are organized into *gadgets* (instructions), each performing a well defined task. ROP has been demonstrated as a technique for creating a Turing complete collection of gadgets that can be deployed on any architecture [11]. The sophistication of ROP exploits can be seen in the level of flexibility that can be accomplished without code injection or even function calls; indicators which are often used to detect foul play in systems. ROP exploitation is now considered one of the prime techniques used by today's exploit developers.

III. MITIGATIONS

Various malware mitigation strategies have been proposed and implemented. The approaches also vary from aiming to detect the intrusion of "foreign" code to methods of analyzing system and network for malicious behavior.

A. Sandboxing

Sandboxing is a security mechanism for separating running programs in a limited environment. Resources allocated for these programs are tightly controlled to contain the damage and spread of malware. This mechanism is widely used for executing untrusted programs, untrusted users and untrusted websites. With reference to evasion techniques, sandboxing at the end system is an appropriate defense against obfuscation techniques such as polymorphic code and encrypted sessions.

There has been an increasing trend for providing sandboxing features in security applications and even building in sandboxing mechanisms in ordinary applications. Well known implementations include Mandatory Integrity Control in the Microsoft Windows operating systems [12], the host based Comodo Firewall [13], Applets, Apache Chroot Jail [14], and more recently the Adobe Reader X version of the pdf reader [15].

An important issue to consider in sandboxing is the performance tradeoff if it used to verify code before being given access to more system resources. Since IDSs are often placed in chokepoints, it would also be virtually impossible to implement this at a network device level (MiTM) without significant loss of throughput.

B. Session Reassembly

Session reassembly is a vital component in addressing network level evasion. The fragmentation and session splicing techniques discussed earlier highlight two important requirements for proper session reassembly at the IDS device. Firstly the IDS must understand how network packets will be reassembled and read at the end node. Secondly the IDS must have enough resources to do this operation during the network's maximum traffic load.

On dynamic networks this could be more challenging, for example in detecting fragmentation time-outs. However stronger network architecture security can help in reducing the burden on IDS devices. High value networks should be designed statically with least privilege based access control thus enabling the IDS to know whether TTL values in packets are valid for the destination host.

C. Data Execution Prevention

DEP or W-xor-X is a security feature that marks memory as either writable or executable, but not both. It is included in most operating systems today. The DEP hardware mode enforces DEP for CPUs that enables the NX (No eXecute) bit. This feature is also known as XD (eXecute Disable) bit on Intel processors, Enhanced Virus Protection on AMD processors, and XN (eXecute Never) on ARM.

Software-enforced DEP is a different form of protection to hardware-enforced DEP in that it is designed to block malicious code that takes exception handling mechanisms on the Windows platform [16].

As pointed out by Shacham, DEP only prevents attackers from injecting and executing malware on a target process/system. Since return-into-libc and return-oriented programming techniques don't inject any code, this feature is prone to the mentioned attacks.

D. Address Space Layout Randomization

Address Space Layout Randomization (ASLR) is another widely deployed defense introduced by the PaX Team in 2003 [17]. ASLR creates randomness in the memory addresses of a target process. Security is created by making it difficult for attackers to find the areas they wish to attack, since the search space is made sufficiently large. Every time the program is restarted the address space layout is randomized thus preventing attackers from using the same exploit code against vulnerability.

Both code-injection and code-reuse attacks can be thwarted or made significantly more difficult using this, as they both require knowledge of addresses in the target process. However brute force attacks have been demonstrated to be successful against the Linux PaX ASLR design due to the limitations of creating randomness in 16 bit architectures [18].

Another method of bypassing ASLR is by heap or Just-in-time (JIT) spraying of the process heap. "Spraying"generates the appropriate data on the heap to overwhelm ASLR, thus creating the conditions (order) in memory to enable attackers execute successful exploits [19]. A combination of techniques, e.g. ROP and heap spraying, can be considered as a sophisticated attack that would probably only be employed against high value targets.

E. Control-flow Integrity

In contrast to other mitigations that focus on preventing the intrusion of malicious code, control-flow integrity (CFI) aims to ensure the behaviors of programs are according to design. This essentially means that the control flow of running processes cannot be subverted to be hijacked by malware in the second step of exploitation. Research on CFI has found that CFI mechanisms embedded in software is practical and costs little performance overhead [20]. Shacham has emphasized that CFI is the better defensive approach to addressing ROP threats.

F. Windows 8 ROP Mitigation

Microsoft introduced a number of malware mitigation features in Windows 8 including increased ASLR randomness, enhancing kernel protections, adding integrity checks in the heap, and virtual function table guards in Internet Explorer [21]. One protection mechanism for virtual memory, attempts to mitigate ROP malware by checking if the stack pointer falls within the range defined by the Thread Environment Block (TEB). The VirtualProtect functions had previously been misused to mark executable memory segments as writable, and VirtualAlloc to create fresh memory segments.

The idea is that malware that keep the ROP payload on the heap cannot return into VirtualProtect or VirtualAlloc as this would fall outside the range defined in TEB. Security researcher Dan Rosenberg has demonstrated that this mechanism is fairly easy to bypass by simply implementing an additional pivot gadget, to return into the original stack and then back to VirtualProtect [22].

IV. Analysis

A. Future of Evasion Techniques

As seen from our discussion, evasion techniques are continuously refined to make malware less and less uniquely identifiable. Code reuse techniques are vivid examples of how traditional code injection defenses can be easily bypassed by today's malware developers. Furthermore layers of complexity are added when the techniques are used in combination for example in Packed, Printable, and Polymorphic Return-Oriented Programming [23].

We see that evasion techniques will continue to become much trickier to detect in that they will strive to camouflage malware in seemingly harmless system/program behavior. By hijacking existing processes and utilizing already available instructions, today's malware can be analogized with cancer; difficult to distinguish and separate from healthy tissue.

In Table 1 we compare the various evasion techniques and attempt to provide values based on the sophistication, difficulty to detect, and impact. In this analysis impact refers to the effort required by vendors to enhance their systems in addressing the threat. We concluded that code reuse attacks present the greatest threat, being very difficult to detect and to mitigate. However these attacks are still considered sophisticated and difficult to to implement by the average malware writer, thus reducing the probability of attack. This could change in the future with more documentation being shared on public hacker forums.

We categorized some of the more classic techniques, such as obfuscation and fragmentation, as being less sophisticated and easier to detect. These attempts of masking attack payloads still require a form of reassembly or decryption to be loaded into a target host's memory, allowing for signature detection of the resulting payload.

TABLE 1. Comparison of Evasion Techniques

Evasion Technique	Metric		
	Sophistication	**Detection Difficulty**	**Impact**
Obfuscation	Low	Low	Medium
Fragmentation & Session Splicing	Low	Low	Low
Application Violations	Medium	Low	Medium
Protocol Violations	Medium	Low	High
Insertion	Medium	Medium	Medium
DoS	Medium	High	Medium
Code Reuse	Very High	Very High	Very High

B. Future of Mitigations

Simple string matching signature-based detection may have been effective a decade ago but are no longer adequate today. Just by reviewing the mitigation schemes we discussed, it is noteworthy that most of them are not direct enhancements on IDS technology. From this fact it is apparent that IDS implementations cannot be viewed as adequate security mechanisms on their own. This reinforces the paradigm that security must be implemented holistically in an organization, encompassing application development, OS security features, network architecture, etc. We can also highlight due to these developments, the IDS cannot stand alone as a bastion against adversaries and must be used in concert with other technologies to properly prevent targeted attacks.

Another interesting fact is that many security enhancements have been easily bypassed not long after being announced and even before product launching (e.g. Windows 8 ROP mitigation). Although it is common knowledge that developers and security vendors are always struggling to keep up with exploit development, there has been an absence of significant strides in developing defense mechanisms against sophisticated attacks. Echoing the views of other security researchers, we see that the traditional approach of preventing intrusion is no longer enough. More adoption of a behavioral approach, such as the integrity checks in CFI systems, would be breakthrough paradigm in defense mitigation development.

By making sure that programs are behaving as intended by the developer/user, it would effectively deny attackers the opportunity to subvert running programs.

V. Conclusions

Malware evasion techniques are developing at a pace that is leaving vendors lagging behind. In this paper we present a survey on the various techniques employed in malware to evade detection by security systems. Evasion techniques we cover in this paper were obfuscation, fragmentation and session splicing, application specific violations, protocol violations, inserting traffic at IDS, reuse attacks etc. We also discussed mitigations such as sandboxing, session reassembly, data execution prevention, address space layout randomization, control flow integrity etc. The sophistication of evasion techniques used by attackers nowadays should encourage vendors to adopt a more robust approach in providing stronger security mechanisms such as CFI. Organizations must also be aware that IDS implementations as standalone systems are far from adequate in providing the security needed to keep their assets safe.

Acknowledgments

We thank Tom Gregory, Hanny Haliwela, and Mada R. Perdhana of the Spentera Security team for sharing their insight on malware development.

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number: 2011-0004833 and 2011-0023076).

References

[1] Scarfone, Karen and Mell, Peter. *Guide to Intrusion Detection and Prevention Systems,* Computer Security Resource Center (National Institute of Standards and Technology) (800-94) February 2007.

[2] Yetiser, Tarkan. "Mutation Engine Report". Baltimore, MD, June 22, 1992.

[3] Serna, Fermin J., *Polymorphic shellcodes vs. application IDSs*, Next Generation Security Technologies. January 2002.

[4] Ptacek, Thomas and Newsham, Timothy, *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection* January 1998.

[5] Timm, Kevin, *Intrusion Detection FAQ: How does and attacker evade IDS with Session Splicing*, SANS. http://www.sans.org/security-resources/idfaq/sess_splicing.php.

[6] Timm, Kevin, *IDS Evasion Techniques and Tactics* http://www.symantec.com/connect/articles/ids-evasion-techniques-and-tactics May, 2002

[7] Abhishek Singh, Scott Lambert, Tanmay A Ganacharya, Jeff Williams, *Evasions in Intrusion Prevention/Detection Systems*, Microsoft USA, Virus Bulletin, April 2010.

[8] Joseph Taylor, Intrusion Detection FAQ: IDS Evasion and Denial of Service Using RPC Design Flaws, SANS. http://www.sans.org/security-resources/idfaq/rpc_evas.php

[9] SolarDesigner. *Getting around non-executable stack (and fix)*. Bugtraq mailing list, http://www.securityfocus.com/archive/1/7480, Aug. 1997.

[10] H. Shacham. *The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)*. In S. De Capitani di Vimercati and P. Syverson, editors, Proceedings of CCS 2007, pages 552–61. ACM Press, Oct. 2007.

[11] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. *Return-oriented programming: Systems, languages, and applications*. Manuscript, 2009. Online: https://cseweb.ucsd.edu/~hovav/papers/rbss09.html.

[12] Conover, Matthew. *Analysis of the Windows Vista Security Model*, Symantec Corporation. 2007

[13] Comodo Firewall. http://www.comodo.com/home/internet-security/firewall.php

[14] Sanchez, Carlos. *Running builds in a chroot jail.* https://svn.apache.org/repos/asf/continuum/trunk/continuum-docs/src/site/apt/administrator_guides/chroot.apt. June, 2008

[15] Adobe Reader X. http://www.adobe.com/products/reader.html

[16] *A detailed description of the Data Execution Prevention (DEP) feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003*.
http://support.microsoft.com/kb/875352/EN-US/

[17] PaX Team. *PaX Address Space Layout Randomization*. http://pax.grsecurity.net/docs/aslr.txt

[18] H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, D. Boneh, *On the Effectiveness of Address-Space Randomization*, in Proceedings of 11th ACM Conference on Computer and Communications Security, Oct. 2004, http://www.stanford.edu/~blp/papers/asrandom.pdf.

[19] Lupin. *Heap Spray Exploit Tutorial: Internet Explorer Use After Free Aurora Vulnerability*, January 2010. http://grey-corner.blogspot.com/2010/01/heap-spray-exploit-tutorial-internet.html

[20] Abadi, M., Budiu, M., Erlingsson, U., and Ligatti, J. *Control-flow Integrity: Principles, implementations, and applications.* In ACM CCS (Nov. 2005).

[21] Garms, Jason. *Building Windows 8: Protecting you from malware*. http://blogs.msdn.com/b/b8/archive/2011/09/15/protecting-you-from-malware.aspx

[22] Rosenberg, Dan. *Defeating Windows 8 ROP Mitigation*. September 2011. http://vulnfactory.org/blog/2011/09/21/defeating-windows-8-rop-mitigation/

[23] Kangjie Lu, Dabi Zou, Weiping Wen and Debin Gao *Packed, Printable, and Polymorphic Return-Oriented Programming*. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID 2011)*, Menlo Park, California, USA, September 2011