Automatic Discovery of Evasion Vulnerabilities Using Targeted Protocol Fuzzing

Antti Levomäki
Forcepoint
Helsinki, Finland
antti.levomaki@forcepoint.com

Olli-Pekka Niemi
Forcepoint
Helsinki, Finland
opi@forcepoint.com

Christian Jalio
Forcepoint
Helsinki, Finland
christian.jalio@forcepoint.com

***Abstract*— Network protocol normalization and reassembly is the basis of traffic inspection performed by NGFW and IPS devices. Even common network protocols are complex with multiple possible interpretations for the same traffic sequence. We present a novel method for automated discovery of errors in traffic normalization by targeted protocol stack fuzzing. These errors can be used by attackers to evade detection and bypass security devices.**

Keywords—network; intrusion prevention; evasion; fuzzing; IDS; IPS; NGFW

I. Introduction

Network security devices perform real-time analysis on network traffic to detect malicious or unwanted traffic. Intrusion prevention systems (IPS) and next generation firewalls (NGFW) are common examples of middlebox systems that can alert and possibly terminate offending traffic.

Successful traffic analysis requires that the middlebox interprets the traffic in the same way as the actual endpoint. Differences in traffic interpretation cause problems. Interpreting normal actions as malicious will cause the middlebox to cause false positive alerts on normal traffic. Interpreting malicious actions as normal allows attackers to evade detection and bypass the security device.

Network evasion techniques have been researched actively, and there exist a large amount of different evasion techniques. These techniques can be combined to cause a different error in the traffic analysis or sidestep attempts to detect an evasion technique. Our work focuses on rapidly finding evasion combinations that work against any given network middlebox.

II. Protocol fuzzing

In Wikipedia, fuzz testing is defined as “an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program” [1]. Fuzzing is a common method for finding implementation errors in software in an automated way, and is well suited for testing black box implementations [2].

Our goal is to find protocol reassembly problems in middlebox traffic inspection. Most middlebox security products are proprietary solutions with no visibility into the actual inspection process. The inspection behaviour also changes with signature updates, inspection policy changes and software upgrades. The inspection process is therefore a black box for testing purposes.

The search space for different ways to send a network exploit is large. The traffic for a relatively simple HTTP GET request is composed following at least the following specifications:

•RFC 2616 – Hypertext Transfer Protocol HTTP/1.1 [3].

•RFC 793 - Transmission Control Protocol [4].

•RFC 791 – Internet Protocol [5].

Many protocols also have commonly implemented extensions increasing the complexity. Each of the RFCs is written along the robustness principle stated by Jon Postel, “be conservative in what you do, be liberal in what you accept from others” [4], which makes implementation interoperability easier but increases the amount of abnormal behaviour that is accepted by receiving implementations. Implementations of the protocols also do not necessarily exactly follow the specifications.

Fuzzing the network protocol stack by randomly mutating individual packets has a low chance to produce traffic that is interpreted correctly by the intended service. Sending a short HTTP GET request involves 3 IP packets and around 600 bytes. Performing completely random mutations has a space of 4800 bits to flip, 2^4800 combinations, most of which lead to unintelligible traffic.

Instead of random mutations we use a set of atomic evasions that perform mostly valid transformations to the traffic. These atomic network evasion techniques have been actively researched by both the scientific and security communities [6][7][8][9][10][11][12][13]. Applying combinations of such evasions have a higher chance of producing valid traffic and interesting traffic patterns.

Gorton and Chambers [14] combined three different network layer evasions to bypass traffic inspection. The test

space of our evasion combinations is large enough that we cannot execute them all. Instead we perform random evasion combinations with random parameters to test the inspection performance.

Testing protocol normalization and inspection by controlling both endpoints is interesting when validating that the middlebox implements a given specification and performs as intended. For finding differences in traffic interpretation between a real operating system and a security device, we need the real operating system as the other endpoint.

Since protocol stacks are layered, finding a working evasion combination on a low level often means that everything above that layer is vulnerable. This is significant since a low level normalization flaw can render a middlebox blind to every attack on the upper levels. Sometimes evasion test results are represented as a percentage, obfuscating the fact that low level evasions are significantly more damaging than application specific evasions.

III. Test methodology

We use a combination of two testing tools, Mongbat and Evader. Evader performs a single exploit attempt using given evasions. Mongbat runs a large amount of parallel Evader instances and selects the addressing and evasions they use. The test setup consists of the following:

•Mongbat controlling the attack execution

•Evader running actual attacks

•Victim host vulnerable to the attack performed

•Device under test (DUT) inspecting the traffic and terminating all suspected attacks

The goal is to find evasion combinations that produce traffic that is understood by the victim host but not seen as malicious by the DUT. We use working exploit payloads to validate that the exploit has been received and executed correctly by the victim host, and the DUT to terminate everything it sees as malicious.

A test case is successful if the victim host executes the exploit payload. This test method rejects any evasion combinations that break the traffic so that the target operating system or application no longer understand it. Also all traffic that is identified by the DUT as malicious will be terminated.

Intrusion detection systems (IDS) that only alert on malicious traffic but cannot terminate it cannot be tested with this method. These devices can not normalize traffic by dropping ambiguous network packets, which leaves them with the even more difficult task of determining which traffic interpretation is chosen by the intended target. We consider the inspection capabilities of an IDS to be an inferior subset of an IPS.

No special care is taken to avoid sending broken traffic as the result of combining unsuitable evasions. It is our view that it is better to test with large amounts of slightly broken traffic than to try to guess how the middlebox and victim protocol stacks interact. While blacklisting attacker IP addresses can impede the test, it is not a valid mitigation against botnets or prepared attackers who have done their proverbial homework in advance.

A. Evader

Evader is a command line tool that executes a single network exploit attempt and reports the result. If the attack fails the protocol state where the failure happened is reported.

The tool contains a userspace TCP/IP stack and application layer clients capable of performing different evasion techniques. The exploits implemented (Table 1) are old and should be detected by all intrusion prevention devices. TCP timeouts and other delays have been optimized to maximize test speed.

Exploit execution has been separated into stages, allowing different evasions to be used for different protocol stages. For example, TCP segmentation might only be applied to the first SMB handshake message. Many evasions also contain a probability of application that causes the evasion to be applied to different places in the attack.

B. Mongbat

Mongbat coordinates test execution and runs a set of Evader instances in parallel. The outline of a test run using a bind shell payload is the following:

1. Execute a clean payload run to see that the victim service is running and accessible.
2. Check that the bind shell port assigned to this worker thread is not open.
3. Execute the attack with a random set of evasions.
4. Check if the bind shell port opened. If it did, the attack was successful.

This ensures that we report that an attack was successful only when the target host executed the exploit code.

TABLE I. Evader test cases

CVE	Protocols used	Description
CVE-2004-1315	IPv4, TCP, <TLS>,HTTP	HTTP phpBB highlight
CVE-2008-4250	IPv4, TCP, SMB, MSRPC	MSRPC Server Service Vulnerability
CVE-2012-0002	IPv4, TCP, RDP	Windows RDP Denial of Service
CVE-2014-0160	IPV4, TCP, TLS	OpenSSL ‘Heartbleed’

IV. Results

We ran tests against six commercial NGFW / IPS products. The test set includes devices from Gartner “Intrusion Detection and Prevention” and “Enterprise Firewall” 2017 magic quadrant leaders. Each device was updated to the latest software version and used the latest signature set available. An attempt was made to tune the configuration on each device for the maximum amount of inspection while still allowing the Evader clean test to go through on all used attacks. Each device was able to stop the attack without evasions, except for one device lacking TLS inspection capabilities.

The attacks used and their labels in the result tables are:

•**HTTP**: PhpBB highlight vulnerability exploited over HTTP with a TCP bind shell payload.

•**HTTPS**: PhpBB highlight vulnerability exploited over HTTPS with a TCP bind shell payload.

•**Conficker**: MSRPC Server Service Vulnerability exploited with a TCP bind shell payload.

•**Heartbleed**: OpenSSL ‘Heartbleed’ vulnerability exploited. Success criteria is that the TLS server returns extra data in a heartbeat response to the attacker, i.e. the information disclosure flaw is triggered.

A 10 minute run of Mongbat for each of the four attacks was performed against all devices using 16 parallel worker threads.

Overall results in Table 2 show that multiple successful evasions can be found with our automated method during the relatively short test period. The most common working single evasions after removing the randomized evasion parameters are described in Table 2. Table 3 displays which device and attack the evasion worked against. The mapping is based on the results of the 10 minute test runs, more working evasions would likely be found after a longer testing period.

TABLE II. Success / attempt count

Vendor	HTTP	HTTPS	Conficker	Heartbleed
Vendor I	72 / 12364	crash[a]	21 / 858	0 / 557
Vendor II	133 / 8481	97 / 4119	16 / 2368	25 / 899
Vendor III	126 / 8788	277 / 4059	15 / 1204	40 / 1092
Vendor IV	746 / 1833	N/A[b]	2 / 1077	N/A[b]
Vendor V	3366 / 8975	2550 / 5970	8 / 3561	50 / 891
Vendor VI	0 / 7366	0 / 6337	0 / 7778	0 / 994

[a] Device repeatedly crashed during test run, invalidating results.

[b] Could not be configured for TLS inspection, did not terminate the attack.

TABLE III. Most common working single evasions

Evasion name	Protocol	Symbol	*Description*
tcp_paws	TCP	P	TCP PAWS elimination
tcp_chaff	TCP	C	TCP chaff packets
tls_seg	TLS	T	TLS record layer segmentation
http_request_method	HTTP	H	Nonstandard HTTP request method

The evasions are performed on different protocol layers. It can be seen that an evasion vulnerability on a lower protocol layer like TCP will prevent traffic inspection on higher protocol layers. For example, the TCP PAWS evasion can be used to evade inspection of both HTTP and MSRPC based attacks.

Some of the tested devices implement TLS inspection as a Man-In-The-Middle (MITM) where the device acts as a TLS server for the connection client and a TLS client for the server. We speculate that this is the reason why Vendor II is vulnerable to the TCP PAWS evasion on HTTP and Conficker attacks, but not on HTTPS and Heartbleed. In the latter attacks a MITM device is the TCP connection endpoint, so there can be no ambiguity between the inspected traffic and the endpoints interpretation of it.

V. Conclusions

We demonstrate that our automated method of finding evasion vulnerabilities can be used to rapidly find working evasions against modern up-to-date NGFW and IPS products.

The results also point out that low-level evasions are often payload independent. The fact that TCP level evasions are working in 2017 despite being known since 1998 [6] is a cause for concern. The authors speculate that the reason may be as simple as performance optimization based on dedicated networking hardware or memory consumption optimization.

Automated testing of traffic inspection should be used for tuning traffic inspection policies. An inspection policy could be tuned so that required normal traffic passes but automated attacks are stopped. This would provide some validation that network traffic is inspected as intended, with no required signatures or configuration options missing.

TABLE IV. Working single evasions

Vendor	HTTP	HTTPS	Conficker	Heartbleed
Vendor I	H			
Vendor II	P, C	T, H	P	T
Vendor III	P, H	P, C, T, H	P	P, C, T
Vendor IV	P, C, H	P, C, T, H	C	P, C, T
Vendor V	P, C, T, H	P, C, H		T
Vendor VI				

References

[1] Wikipedia contributors, "Fuzzing," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fuzzing&oldid=806227686 [Accessed November 20, 2017].

[2] R. McNally, K. Yiu, D. Grove and D. Garhardy, “Fuzzing: The state of the art”.

[3] IETF, “Hypertext Transfer Protocol – HTTP/1.1”, IETF, RFC2616, 1999.

[4] IETF, “Transmission Control Protocol,” IETF, RFC793,1981.

[5] IETF, “Internet Protocol,” RFC791, 1981.

[6] T. Ptacek and T. Newsham, “Insertion, Evasion and Denial of Service; Eluding Network Intrusion Detection,” Secure Networks Inc., 1998.

[7] Horizon, “Defeating Sniffers and Intrusion Detection Systems,” Phrack Magazine, vol. 8, no. 54, 1998.

[8] “NIDS Evasion Method named "SeolMa",” Phrack, vol. 0x0b, no. 0x39, 2001.

[9] R. F. Puppy, “A look at whisker's anti-IDS tactics,” 1999. [Online]. Available: http://www.ussrback.com/docs/papers/IDS/whiskerids.html. [Accessed: November 20, 2017].

[10] D. J. Roelker, “HTTP IDS Evasions Revisited,” 2004. [Online]. Available: https://www.defcon.org/images/defcon-11/dc-11-presentations/dc-11-Roelker/dc-11-roelker-paper.pdf [Accessed: November 20, 2017].

[11] B. Caswell and H. D. Moore, “Thermoptic Camouflage: Total IDS Evasion,” in Blackhat, 2006.

[12] O. Niemi and A. Levomäki, “Evading Deep Inspection for Fun and Shell,” in Blackhat, 2013.

[13] Steffen Ullrich, “HTTP Evader”, 2015. [Online]. Available: https://noxxi.de/research/http-evader.html. [Accessed: November 20, 2017].

[14] A. Gorton and T. Champion, “Combining Evasion Techniques to Avoid Network Intrusion Detection Systems,” Skaion, 2004.