MASTER THESIS

Hacking the Next Web: Penetration Testing over IPv6

Author:
Christiaan Ottow
s0070300
chris@6core.net

Graduation committee:
F. van Vliet, MSc
dr. P. de Boer
dr. A. Pras

programme	Master of Computer Science
track	Kerckhoffs
chair	Design and Analysis of Communication Systems
department	Faculty of Electrical Engineering, Mathematics and Computer Science

UNIVERSITY OF TWENTE.

June 2012

Contents

Chapter 1

Introduction

This is the report of a final project on the impact of IPv6 on penetration testing. The final project concludes the author's participation in the Computer Security specialization of the Master of Computer Science programme at the University of Twente. This specialization, called Kerckhoffs, is a joint effort between the universities of Twente, Eindhoven and Nijmegen, and focusses on the security of digital systems. Before enrolling in this programme, the author completed the Bachelor of Telematics programme, which deals with different aspects of networking. This final project has components of both, as it concerns the influence of a new network protocol on information security.

The project was executed in the period from February 2011 to April 2012 as an external project at Pine Digital Security in The Hague. The output of the research is a paper that was submitted to the EUNICE networking conference of 2012. The paper is included in this report in full, and has its own introduction that introduces the topic and the research itself.

1.1 Assignment

The final project proves that the student is able to perform independent scientific research, and concludes his master's education. The faculty of Electrical Engineering, Mathematics and Computer Science, where this master's programme is offered, has published criteria[3] which must be met by the final project. The student must be able to:

- Clearly formulate a problem statement
- Identify relevant literature
- Draw up a work plan
- Adjust problem statement and work plan in accordance with interim evaluations

- Analyse different possible solutions and motivate a choice between them

- Communicate the research and design activities both written and in presentations

- Show ability to reflect on the problem, on the research/design approach, on the solution and on one's own performance

- Demonstrate creativity and the ability to work independently

This project meets those criteria in the following ways. After formulation of the research problem, a literature study has been performed and reported in the Research Topics course, prior to the start of the final project. In the final project paper, research questions are derived from the problem statement. An approach is described by which these questions are answered. Different approaches have been analyzed, and the chosen approach is motivated in the paper. Throughout the course of the project, both the research questions and the approach have been adjusted and refined. The research is communicated through the paper, and if accepted to EUNICE, through a presentation. It is also communicated through a colloquium by the author at the university. Lastly, the project has been performed independently by the author, with regular feedback from the supervisors.

1.2   Thesis Organization

The next chapter consists of the paper that is the result of the research. In the paper, two algorithms are described for activities in the penetration testing process that change when IPv6 is used. The descriptions are very concise, as they are not part of the primary contribution of the paper. However, since they are useful to the external principal, they are described in more detail in Appendix A. In the paper, references are made to checklists that are used for penetration testing. These checklists are included in Appendix B, along with a list of changes proposed in the paper.

Chapter 2

Paper

This chapter contains the paper on IPv6 and penetration testing that has been submitted to the EUNICE 2012 Conference on Information and Communications.

The Impact of IPv6 on Penetration Testing

Christiaan Ottow[1], Frank van Vliet[2], Pieter-Tjerk de Boer[1], Aiko Pras[1]

[1] University of Twente, Enschede, The Netherlands,
`chris@6core.net`, {`p.t.deboer,a.pras`}`@utwente.nl`
[2] Pine Digital Security, The Hague, The Netherlands,
`frank.vanvliet@pine.nl`

Abstract. In this paper we discuss the impact the use of IPv6 has on remote penetration testing of servers and web applications. Several modifications to the penetration testing process are proposed to accommodate IPv6. Among these modifications are ways of performing fragmentation attacks, host discovery and brute-force protection. We also propose new checks for IPv6-specific vulnerabilities, such as bypassing firewalls using extension headers and reaching internal hosts through available transition mechanisms.
The changes to the penetration testing process proposed in this paper can be used by security companies to make their penetration testing process applicable to IPv6 targets.

Keywords: IPv6, security, penetration testing, host discovery, transition mechanisms

1 Introduction

IPv4 address space is nearing depletion in some regions [2, 28] and adoption of its successor, IPv6, is increasing [20]. At recent security conferences, security researchers have discussed flaws in both the IPv6 protocol and its implementations [3, 6, 14, 17]. Since IPv6 replaces IPv4 as network layer protocol, future network security activities can also be subject to change. Penetration testing is one such activity; companies that provide penetration testing services need to know how to change their process in order to perform penetration tests on IPv6 targets.

In this paper we will answer the following question:

"How will the use of IPv6 change remote penetration testing of web-applications and servers?"

Penetration testing can be performed in many ways, from physically breaking into structures to social engineering attacks and the hacking of web applications. Since there is no commonly accepted approach to performing penetration tests, we base our discussion on the case study of Pine Digital Security. Pine is a company in the Netherlands that performs remote penetration testing on web applications and servers. They use public checklists of vulnerabilities, combined

with clear defined methods to check for each vulnerability on those checklists. This results in a penetration testing process that can be used in this research to determine the impact IPv6 will have.

The following sub-questions will be used to answer the main question:

– What components of the current penetration testing process need to be changed for IPv6 targets?
– What new components can be added to the penetration testing process for IPv6 targets?

The first sub-question concerns how the current penetration testing process is affected when IPv6 is used: what components need to be modified in order to achieve their goal, or can be removed because they become obsolete. For each component of Pine's penetration testing process, the performed activities are examined to determine how they are affected by a change to IPv6. This is done by analyzing the dependencies of every action performed, to see if functions specific to IPv4 are relied upon. A suggestion for removal from the process or modification is made per activity that is found to be affected. By following this approach, we determine what components of the current penetration testing process are affected, but we do not discover new additions.

The second sub-question concerns what new vulnerabilities IPv6 introduces that are not currently tested for in Pine's penetration testing process. From existing literature and community sources such as protocol standards, conference presentations and weblogs, we collect documented IPv6 vulnerabilities. When these would be applicable within the scope of penetration testing as defined in this research (see Section 2), we propose a check for them as an addition to the process.

In the next section, we discuss our choice of Pine as a case study and Pine's penetration testing process. Components of this process that should be modified for IPv6 or can be removed are discussed in Section 3. Section 4 discusses proposed additions to the process. Finally, conclusions and suggestions for future work are presented in Section 5.

2 Backgrounds on Penetration Testing and the Case Study

In this section we discuss our choice of case study and describe their process of penetration testing for servers and web applications.

Pine Digital Security performs various security services to companies, among which penetration testing. Penetration testing at Pine is described as finding vulnerabilities in servers and web applications using checklists of vulnerability classes, from the perspective of certain types of attackers. Most often this is the perspective of an outsider at a remote location (somewhere on the Internet). Vulnerabilities qualify if they can be used to compromise the confidentiality or integrity of the protected assets. This implies that the penetration test does not focus on availability, although exceptions are made for availability vulnerabilities

that can be exploited without large force, and that can be tested for in a non-destructive manner.

The process of penetration testing at Pine follows the phases described by several sources [30, 31, 34]: planning, execution, post-execution. In the planning phase, apart from commercial activities, an assessment of the customer's systems and risks is made to determine the test targets and scope. Sometimes it is necessary to perform host and service discovery on the target to determine the size of the attack surface. In the execution phase, the penetration tester finds and exploits vulnerabilities in the targets, making sure that every vulnerability on the applicable checklists is checked for. Discovery and attack are executed in cycles, as described in [30]. In the post-execution phase, the results are formulated along with recommendations for mitigation, and are discussed with the client. Cleaning up and quality assurance also take place in this phase.

Pine is representative as a case study of remote penetration testing of web applications and servers. Since Pine uses public [7–10] checklists and the description of their penetration testing process is available, the impact that IPv6 will have can be determined in a structured way. As described, the phases of Pine's penetration testing process follow a recognized model. The checklists have been assembled by independent security experts with years of penetration testing experience. They contain the most prevalent security issues such as those from the OWASP Top Ten [29] and those exploited by popular automated testing tools, as well as lesser-known issues.

3 Impact of IPv6 on the current penetration testing process

In this section we discuss the components of Pine's current penetration testing process that either need to be changed in order to be applicable to IPv6, or become obsolete and can be removed from the process. We go through the three phases described in the previous section, and discuss the activities that take place. For each of these activities, we analyze its dependencies to see if it relies on features specific to IPv4. We only discuss those items that would need change or can be removed.

3.1 Preparation phase

In the preparation phase, little activity involving the target systems of the penetration test takes place. Normally the customer will give an exact list of IP addresses that form the scope of the penetration test. When the customer however gives a network range instead of an exact list, a scan to detect the number of hosts in a network needs to be performed.

Host and service discovery When performing host discovery in a subnet, all TCP ports and some UDP ports on all hosts are probed. This is done since

many hosts do not respond to various kinds of alive polls (like ping requests), but do offer services on some ports. Alive detection can stop once a proof of aliveness is received from the server. A server without firewall will respond to a TCP connection request to a closed port by sending a TCP packet with the RST flag set, which indicates its aliveness. However, many servers have firewalls that are configured to ignore connection requests to all ports by default, and make exceptions for ports that are on a whitelist. In this case, scanning will have to continue until the first whitelisted port is probed, which will evoke a proof of aliveness from the server in the form of a TCP packet with either the SYN and ACK flags or the RST flag set. Scanning in this case can take very long since the scanner cannot distinguish a lost transmission from a dropped one and has to retry probes multiple times after waiting for a timeout.

This type of scanning is especially problematic in IPv6 networks [11, 33] because of the large IP space typically in use. The standardised netmask size for an IPv6 subnet is 64 bits, leaving 64 bits for numbering hosts [19]. To scan those 64 bits of address space as described would take an infeasible amount of time: 28 years under conditions that favor the scanner [33]. Therefore, exhaustive scanning of all addresses in the network is infeasible within the scope of a penetration test, unless an extremely small prefix has been given.

There are alternatives to exhaustive scanning. Address information can often be retrieved from external sources such as the Domain Name System (DNS) and online directories. Several websites track changes in WHOIS and DNS, and map virtual hosts to IP addresses. Other services index networks by found services. Address information can be taken from these websites. DNS can be used in multiple ways to find addresses. Some servers allow AXFR queries that enable an attacker to download the whole zone. Guessing hostnames using wordlists can also be used. When DNSSEC is used, both NSEC and its successor NSEC3 [23] offer possibilities to enumerate zones [4]. Additionally, the tree of PTR records can be queried to find addresses that are in use when the nameserver returns NOERROR for empty non-terminals. This behavior is RFC-compliant, and implemented by most nameservers [13]. PTR records need to be configured for this enumeration technique to work however, which is not the case in all networks.

Furthermore, assumptions can be made about address assignment policies. Different sources suggest that servers and routers are usually numbered through manual assignment or DHCPv6, not using SLAAC or privacy extensions [14, 17, 24]. In fact, DHCPv6 and manual assignment are even recommended [32]. Research indicates that manual assignment is often based on sequential numbering, numbering with the service port (e.g. assigning an address ending in ::80 to a webserver), wordiness (using hexadecimal characters to form words) and IPv4 addresses in IPv6 addresses. Since the focus of our host discovery is on servers and web applications, manual assignment and DHCPv6 are expected to be prevalent among the targets [11, 17]. To find alive hosts, one could assume that these schemes are used for assignment, and guess what other addresses are in use. Based on given or found addresses, address ranges can be found in which

sequential numbering, hex-word, service port and IPv4-address based addresses are in use. For each of the suggestions that this "smart guessing" algorithm generates, alive detection needs to be performed. For a small number of suggestions, traditional alive detection could be performed by scanning all TCP and UDP ports. As the number of suggestions increases, this approach becomes less feasible. A limited number of ports could be probed, alongside generic methods (ICMP ping for instance).

An algorithm using the above methods for "smart guessing" cannot guarantee to find all hosts in a network. If a host has a DNS record that is not easily guessable or no DNS record at all, and has an unpredictable IPv6 address, it may not be found. An early test with such an algorithm reports finding 90-95% of servers on public networks [18]. If a guarantee is required that 100% of the hosts have been included in the test, the only option as of yet is to require an exhaustive list of IP addresses from the customer.

3.2   Execution phase

In the execution phase, the penetration tester attempts to hack the targets and makes sure that each vulnerability on the checklists is checked for. Pine has defined methods for each check, and has documented tools that can be used. These tools are free, commercial or in-house developed. In order to use the tools on IPv6 targets, they need to have basic IPv6 support that includes specifying IPv6 hosts, resolving names to IPv6 addresses and transmitting data over IPv6 sockets. This is a change that affects every tool used, and will not be mentioned separately for every checklist item.

DNS information gathering In two items on the checklists, the Domain Name System (DNS) is queried for (possibly sensitive) information. The goal is not host discovery as was covered in the previous section, but to check if the DNS contains information that should not be stored there, and if the nameservers don't leak the entire zone to attackers. These items are checked by querying the nameservers for the domains in scope, and attempting to download the complete zone (AXFR) or failing that, find certain hosts by guessing hostnames. Reverse hostnames of known IP addresses are resolved as well.

A target that uses IPv6 might have nameservers that are accessible over IPv6, and might have AAAA records in associated zones. The checklist item can remain the same, but the procedure for testing should be changed to include checking both the IPv6 addresses and IPv4 addresses for the nameservers, as they might point to different systems. For all records that are resolved, both the AAAA and A records should be queried. Reverse lookups for both IPv4 and IPv6 addresses should be performed. Wordlists used for brute-force guessing of system names should be updated to contain names relevant to IPv6, such as words containing "ipv6" and the names of popular transition mechanisms such as Teredo and ISATAP.

Fragmentation to bypass firewalls Fragmentation in IPv6 works similar to fragmentation in IPv4. In IPv6 however, fragment information is transmitted in an extension header that is only present when an IP packet is a fragment, and only end nodes perform fragmentation and reassembly.

Fragmentation attacks in IPv4 have been known for years. RFC 1858 [36] describes two attacks that can be used to bypass firewalls using fragmentation: tiny fragments and overlapping fragments. The RFC also suggests mitigations against these attacks.

In the case of tiny fragments, the packet that contains the first step of the TCP handshake (SYN flag) is fragmented across two packets such that the SYN flag, which is at the end of the TCP header, falls within the second fragment. A firewall that tests if a packet constitutes a TCP connection request, will not be able to judge this having just the first fragment. It is possible that no rule will be matched, and the packet let through. The suggested mitigation against this attack is to discard fragments smaller than a minimum size to make sure the TCP flags always fall in the first fragment.

Overlapping fragments can be used to smuggle a connection request past a firewall as well. The first fragment can contain a TCP header with a certain source and destination port but no SYN flag. The firewall may allow this fragment to pass since it is not a connection request. The offset of the second fragment can be tailored to cause reassembly to overwrite the flags field of the TCP header with the data from the second fragment. In this second fragment, a flags field could be set that includes a SYN flag. The firewall might allow the fragment through since it is not the first fragment, while the host reassembling the packet (and parsing the overlap in a favorable way) will read a connection request. The suggested mitigation is to drop fragments with an offset value of 1, since higher offsets do not cause the flags field to be overwritten.

The original IPv6 specification allows overlapping fragments. A later RFC [22] forbids overlapping fragments, but as this is a relatively recent change, not all implementations follow it. As for tiny fragment attacks, many IPv6 implementations accept non-last fragments smaller than 1280 bytes [3]. Additionally, the IPv6 specification dictates that extension headers occurring any number of times in the same packet must be parsed. This requirement can be used to fill a first fragment with extension headers, and put the TCP header in the second fragment. Many implementations already follow a proposed standard [16] the specifies that all headers, including the upper layer header, must be in the first fragment. This mitigates the tiny fragment attack, but is not the standard yet. Other solutions to the tiny fragment problem are being discussed in [25].

When using fragmentation attacks in IPv6, extension headers can be used as described above in addition to the known IPv4 methods.

3.3   Evaluation phase

In the evaluation phase, the report is written and discussed with the customer. This report includes recommendations for mitigation of the found vulnerabilities.

Like the checklist items, these recommendations may have to change to achieve their goal on an IPv6 target.

Automated attacks The recommendation made by Pine against brute-force attacks is to implement an anti-automation measure in the form of rate-limiting or CAPTCHA solving, based on the client's IP address. Several items on the checklist have this recommendation, for instance those that check for brute-forcing user names and passwords or spamming via feedback forms.

This recommendation works in an IPv4 world where an IP address can be assumed to identify a user, since most users have at most one address per person at a point in time. With IPv6 however, a standard network has a 64-bit netmask (a "/64"), and a standard end-user assignment is a multiple of that [27]. An organization is likely to receive a /48, while a home user is more likely to receive a /56. Since March 2011, the exact allocations are left up to the ISP and are no longer defined in an RFC [27]. In any case, an attacker has a very large IPv6 address space compared to the IPv4 situation. Therefore, acting on a single IP address is not enough: the attacker can change IP addresses very often to circumvent anti-automation measures.

We suggest using a "smarter" algorithm for anti-automation measures, that takes into account the large address space an attacker has. This algorithm is based on "taint" actions: actions that are a trigger to the anti-automation system, such as a failed login attempt. After a defined number of taints in a /64, the whole /64 is blocked. Say the attacker has multiple /64 ranges, he can use multiple to begin with, or move to the next after the first has been blocked. Therefore, a threshold is also set for a /56, for a /48, and so on. After each taint, the algorithm checks for all prefix sizes if its taint threshold has been reached. If so, the prefix can be added to a blacklist.

Session hijacking Another recommendation that needs to change, is the recommendation made for web applications to allow the use of a session from only one IP address. Pine recommends that when a session is started (a session identifier is generated and sent to the user), the IP address of the user is stored on the server along with the session. Upon each subsequent request with that session identifier, the server checks if the IP address where the request came from is the same as the stored IP address. If not, access to the session is denied.

With IPv6, privacy extensions [26] are often used for end-user systems [21]. The addresses that are generated have a limited preferred lifetime, which means they are used for outgoing connections for a limited time. Ubuntu Linux for instance defaults to 24 hours, but router advertisements usually contains a lower lifetime value (4 hours in radvd by default). When the preferred lifetime is over, a new address is generated and the address is not used for outgoing connections anymore.

The described behavior poses a problem for session management according to Pine's recommendation. A user may use different addresses for his requests to the web application during his session, since sessions typically run for longer

than a few hours. Therefore, the IP address cannot be used to lock the user's session anymore.

The /64 prefix in which the random addresses are generated, remains constant in the address, but may be used by many more users. A trade-off has to be made between usability and security: if the current recommendation is applied to IPv6, users with privacy addresses will have trouble using the web application. If the /64 prefix is used, abuse of the session by remote attackers is prevented, but attackers on the same LAN may still hijack the session. With IPv4 this situation often already exists, since many networks use NAT, which results in a network segment having one external address.

We recommend tracking the session using the /64 prefix. The measure is for defense-in-depth: stealing of the session identifier should be made impossible by other means such as proper output escaping to prevent Cross-site Scripting (XSS), the use of SSL and setting proper flags on session cookies. Additionally, in IPv4, many users share the same public IP address with the whole LAN already, so the use of the IPv6 /64 does not offer less protection.

4 Additions to the penetration testing process

In this section we describe additions that should be made to the penetration testing process, based on existing literature, protocol standards and community sources. Proposed additions need to be within our scope of penetration testing. As such, vulnerabilities that are specific to the LAN (such as Neighbor Discovery and Multicast Listener Discovery vulnerabilities) are not considered, since they cannot be abused from a remote perspective. Vulnerabilities that can only be used for denial of service attacks and cannot be tested for in a non-destructive manner are out of scope as well.

4.1 Using extension headers to bypass router ACLs

Extension headers in IPv6 packets are placed between the fixed-size IPv6 header and the payload. The IPv6 standard dictates that any number of extension headers must be processed by receiving nodes.

Most routers can enforce packet filtering policies based on Access Control Lists (ACLs). Filters are usually composed of not only source and destination IP address, but also TCP/UDP ports and flags. To read this information, the router needs to read past the complete IP header in the packet, including extension headers. High-end routers achieve high throughput by implementing the forwarding plane (where ACL checking takes place) in hardware instead of in software. This hardware has a fixed-size view on the packet: often only the first 64, 128 or 256 bytes are evaluated to make a routing or policy decision [35].

This limited packet view can introduce a security problem. A packet can have so many extension headers that the upper layer header is moved outside of the view of the forwarding plane, meaning that the information required for a policy decision is not available. Some routers choose to pass such a packet on

to the control plane, which is software-based and can handle dynamic lengths. Others just let the packet pass [35], which provides an easy way to bypass policy enforcement.

An addition should be made to the checklists for checking whether adding a large number of extension headers leads to bypasses ACL restrictions as described above.

4.2 Unintended exposure due to transition mechanisms

Various mechanisms have been defined to ease the transition from IPv4 to IPv6. Some of these transition mechanisms provide automatic tunneling for hosts that want to use IPv6 in an IPv4-only network. These mechanisms are either based on existing IPv4 tunneling techniques such as IP-in-IP and GRE or encapsulating IPv6 packets inside layer 4 protocols such as UDP.

Hosts that are not reachable by a global IPv4 address, might become globally reachable over IPv6 by the use of these transition mechanisms. This poses a security risk [12, 15]. An addition should be made to the checklists to check for the reachability of hosts in the target network using IPv6 transition mechanisms. This could be achieved by performing alive detection on guessed addresses, based on assumptions about the transition mechanism used. Assumptions about the addresses can be made when ISATAP, Teredo or 6to4 are in use, since these transition mechanisms use predictable information in the IPv6 addresses that are generated.

ISATAP ISATAP is used to provide hosts in a network with IPv6 support, while not migrating the entire network to IPv6. The network itself must have an IPv6 prefix routed to it, but not all infrastructure inside the network needs to support IPv6. The IPv4 network is used as a link layer for IPv6. Hosts in the network use the IPv6 prefix combined with their own IPv4 address *v4addr* to generate an IPv6 address, that ends with the 48 bits :5efe:*v4addr*.

If the network prefix is known, for instance from other IPv6 enabled hosts or a WHOIS service, the generated IPv6 addresses can be guessed based on the IPv4 addresses that are in use. Private IPv4 addresses can even be accessed using this method: if a host has IP address 10.0.0.2 on the LAN and it uses ISATAP in prefix 2001:db8::/32, it will have IPv6 address 2001:db8::5efe:a00:2 (0a00:0002 is 10.0.0.2 in hexadecimal notation) which is globally reachable by default.

6to4 A host that has IPv4 address *v4addr* can have the prefix 2002:*v4addr*::/48 routed to it by setting up a 6to4 tunnel. By substituting known IPv4 addresses of the target into 2002:*v4addr*::/48, prefixes are acquired wherein host discovery can be performed.

Teredo Teredo addresses contain more variables than just the *v4addr*, such as the UDP port number. By making assumptions about these variables, addresses

could be guessed. However, since there are 2^{16} possible port numbers, one would need to test $\frac{2^{16}}{2}n$ guesses on average, where n is the number of IPv4 addresses in use. This will very quickly become infeasible.

An addition to the penetration testing process should be made to discover hosts reachable via the 6to4 and ISATAP transition mechanisms. Within ISATAP prefixes, known IPv4 addresses and commonly used private ranges should be tried. Within 6to4, the IPv4 addresses of border routers in the target network should be used to generate prefixes in which host discovery can be performed.

4.3 Evading policy enforcement using routing headers

One of the IPv6 extension headers is the Routing Header (RH). There are two types of routing headers, of which RH0 (type 0) can be used to specify a list of addresses that should be visited by the packet.

This feature offers the same functionality as the "Loose Source Route Record (LSRR)" option in IPv4, and poses the same security risk. With RH0 enabled, an attacker could craft a packet that follows a certain path into a network, bypassing a firewall. Or, the destination according to the destination address could be allowed by the firewall, while the real final destination specified inside the RH0 header is not [1,5]. The first destination host would then forward the packet over the internal network.

The other type of routing header, type 2, is used in Mobile IPv6, and contains only one address that is validated to be the node's home address. It is therefore not usable by an attacker for firewall bypassing.

Because of the vulnerabilities identified in the RH0 functionality, the IPv6 standard was updated in 2007 [1] to deprecate RH0 extension headers. However, many IPv6 implementations already contained the RH0 functionality, and there is no guarantee that all of them have been updated to reflect the change. Therefore an addition to the penetration testing process should be made to check for firewall bypassing using RH0 headers.

5 Conclusions and Future Work

This paper shows that when IPv6 is used, changes to the penetration testing process are needed. Some activities need to be performed in a different way for IPv6 targets, and some new checks should be added to the process. None of the activities of the current penetration testing process need to be removed from the process.

The following elements of the process need to be changed. Host discovery by exhaustively scanning the entire IP space will not work for normal-sized IPv6 networks. We have described an algorithm that combines different sources of address information with assumptions about address assignment, to replace exhaustive scanning. Host discovery using DNS brute-forcing should be changed to search for IPv6 hosts (AAAA records) as well as IPv4 hosts (A records).

Fragmentation attacks that aim to bypass firewalls should be changed to include IPv6-specific fragmentation attacks. Recommendations for mitigation of vulnerabilities are also affected: brute-force attacks cannot be stopped by blocking single IPv6 addresses. We described an algorithm that takes into account the address space an attacker may have. Furthermore, users cannot be expected to keep the same IPv6 address throughout the lifetime of their session, so the user's network prefix must be taken into account when restricting access to the session.

Additions to the penetration testing process should be made to target several IPv6-specific vulnerabilities. The use of IPv6 extension headers in general allows for router ACL bypassing on some systems. If Routing Headers are allowed, firewalls might be bypassed as well. When certain transition mechanisms are in use, internal systems may be directly accessible to attackers. Additions to the checklists are proposed for these IPv6-specific vulnerabilities.

Using Pine's penetration testing process provided us with a structured way to examine the impact IPv6 may have, but may not be applicable to penetration testing as performed by others than Pine. As future work, the impact of IPv6 on other forms of penetration testing, such as Local Area Network penetration testing, could be investigated. Future research could also verify our findings by performing penetration tests on IPv6 targets with the checks and algorithms proposed here.

References

1. Abley, J., Savola, P., Neville-Neil, G.: Deprecation of type 0 routing headers in IPv6. `http://tools.ietf.org/html/rfc5095` (December 2007)
2. APNIC: APNIC IPv4 address pool reaches final /8. `http://www.apnic.net/publications/news/2011/final-8` (April 2011)
3. Atlasis, A.: Attacking ipv6 implementation using fragmentation. `http://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf` (March 2012)
4. Bernstein, D.: Breaking dnssec. `http://cr.yp.to/talks/2009.08.10/slides.pdf` (2009 August)
5. Biondi, P., Ebalard, A.: IPv6 routing header security. `http://cansecwest.com/csw07/csw07-ebalard-biondi.pdf` (April 2007)
6. Bowne, S.: Defcon 2010 - who cares about IPv6. `http://www.isoc.my/video/defcon-2010-who-cares-about-1`
7. Certified Secure: Certified Secure Advanced Server Scan Checklist. `https://www.certifiedsecure.com/checklists/cs-advanced-server-scan-v3.1.pdf`
8. Certified Secure: Certified Secure Advanced Web Application Scan Checklist. `https://www.certifiedsecure.com/checklists/cs-advanced-web-application-scan-v3.1.pdf`

9. Certified Secure: Certified Secure Basic Server Scan Checklist. `https://www.certifiedsecure.com/checklists/cs-basic-server-scan-v3.1.pdf`

10. Certified Secure: Certified Secure Basic Web Application Scan Checklist. `https://www.certifiedsecure.com/checklists/cs-basic-web-application-scan-v3.1.pdf`

11. Chown, T.: RFC 5157: IPv6 implications for network scanning. `http://www.rfc-editor.org/rfc/rfc5157.txt` (March 2008)

12. Davies, E., Krishnan, S., Savola, P.: IPv6 transition/coexistence security considerations. `http://tools.ietf.org/html/rfc4942` (September 2007)

13. van Dijk, P.: Finding v6 hosts by efficiently mapping ip6.arpa. `http://7bits.nl/blog/2012/03/26/finding-v6-hosts-by-efficiently-mapping-ip6-arpa` (March 2012)

14. Gont, F.: Results of a security assessment of the internet protocol version 6. `http://www.si6networks.com/presentations/hacklu2011/fgont-hacklu2011-ipv6-security.pdf` (September 2011)

15. Gont, F.: Security implications of ipv6 on ipv4 networks. `http://www.ietf.org/id/draft-gont-opsec-ipv6-implications-on-ipv4-nets-00.txt` (April 2012)

16. Gont, F., Manral, V.: Security and interoperability implications of oversized ipv6 header chains. `http://tools.ietf.org/html/gont-6man-oversized-header-chain-01` (April 2012)

17. Heuse, M.: Recent advances in IPv6 insecurities. `http://events.ccc.de/congress/2010/Fahrplan/events/3957.en.html` (December 2010)

18. Heuse, M.: Vulnerabilities, failures - and a future? `http://www.mh-sec.de/downloads/mh-ipv6_vulnerabilities.pdf` (November 2011)

19. Hinden, R., Deering, S.: RFC 4291: IP version 6 addressing architecture. `http://tools.ietf.org/html/rfc4291` (February 2006)

20. Huston, G.: Active BGP entries (FIB). `http://bgp.potaroo.net/v6/as2.0/index.html`

21. Kaps, R.: Ipv6: Privacy extensions einschalten. `http://www.heise.de/netze/artikel/IPv6-Privacy-Extensions-einschalten-1204783.html` (March 2011)

22. Krishnan, S.: RFC 5722 - Handling of overlapping IPv6 fragments. `http://tools.ietf.org/html/rfc5722` (December 2009)

23. Laurie, B., Sisson, G., Arends, R., Blacka, D.: RFC 5155: DNS security (DNSSEC) hashed authenticated denial of existence. `http://tools.ietf.org/html/rfc5155` (March 2008)

24. Malone, D.: Observations of IPv6 addresses. In: PAM'08 Proceedings of the 9th international conference on Passive and active network measurement. Springer-Verlag Berlin (2008)

25. Manral, V.: Tiny fragments in ipv6. `http://tools.ietf.org/html/draft-manral-6man-tiny-fragments-issues-00` (February 2012)

26. Narten, T., Draves, R., Krishnan, S.: RFC 4941: Privacy extensions for stateless address autoconfiguration in IPv6. `http://tools.ietf.org/html/rfc4941` (September 2007)

27. Narten, T., Huston, G., Roberts, L.: RFC 6177 - IPv6 address assignments to end sites. `http://tools.ietf.org/html/rfc6177` (March 2011)

28. NCC, R.: IPv4 exhaustion. `http://www.ripe.net/internet-coordination/ipv4-exhaustion` (2012)

29. OWASP: OWASP top ten. `https://www.owasp.org/index.php/Top_10_2010` (2010)
30. Saindane, M.S.: Penetration testing – a systematic approach. Tech. rep., infosecwriters.com (2006)
31. Scarfone, K., Souppaya, M., Cody, A., Orebaugh, A.: Technical guide to information security testing and assessment. Tech. rep., NIST (2008)
32. SURFnet: IPv6 numberplan. `http://www.surfnet.nl/nl/nieuws/Pages/HandleidingIPv6-nummerplanverschenen.aspx` (February 2011)
33. Vyncke, E.: IPv6 Security. Cisco Press (2009)
34. Wai, C.T.: Conducting a penetration test on an organization. `http://www.sans.org/reading_room/whitepapers/auditing/conducting-penetration-test-organization_67` (2002)
35. Ytti, S.: IPv6 ACL bypass. `http://blog.ip.fi/2011/08/ipv6-acl-bypass.html` (August 2011)
36. Ziemba, G., Reed, D., Traina, P.: RFC 1858 - security considerations for IP fragment filteringsecurity considerations for IP fragment filtering. `http://tools.ietf.org/html/rfc1858` (October 1995)

Acknowledgements

Bibliography

[1] ARIN. ARIN number resource policy manual. `https://www.arin.net/policy/nrpm.html`, 2012.

[2] T Narten, G Huston, and L Roberts. `http://tools.ietf.org/html/rfc6177`, 2011.

[3] University of Twente. `http://onderwijs.cs.utwente.nl/pdf_documenten/Studiegids_master_CS_department_2009-2010.pdf`, 2009.

[4] RIPE NCC. Ipv6 address allocation and assignment policy. `http://www.ripe.net/ripe/docs/ripe-545`, 2012.

Appendix A

Algorithms

In the paper, two algorithms are proposed for activities that change when IPv6 is used. The first is host discovery, which is performed in the preparation phase of the penetration testing process. The second is the recommendation given to clients in the evaluation phase, to prevent automated attacks. The two algorithms are not part of the primary contribution of the paper, and are merely given as suggestions. However, since they are useful to the company where the case study was performed, they are included in this appendix.

A.1   Host discovery

In section 3.1 of the paper, an algorithm for host discovery in IPv6 networks is described in general terms. Due to the size of IPv6 subnets, host discovery by simply scanning all the addresses in a given range has become infeasible. Instead, various sources (see the paper for references) describe how making assumptions about assignment policies can lead to "smart guessing" most of the addresses that are in use in a given network. In the paper, an algorithm was described that makes use of external sources for address information and assumptions about assignment to perform host detection in IPv6 networks.

The algorithm is depicted in Figure A.1. The boxes indicate an action that is performed; the arrows indicate the information flow between the actions. Unless otherwise indicated, the information that travels between boxes consists of IPv6 addresses.

The following information is given as input to the algorithm:

1. Given DNS names, IPv4/IPv6 addresses and IPv4 portscan results. When a customer specifies a target for a penetration test, some information (a URL or IPv4 addresses for instance) is given. Any DNS names and/or IP addresses are given to the algorithm as starting point. An exhaustive portscan can be performed on the IPv4 ranges, the results of which can be used by the algorithm.

2. Prefix information. When the customer has specified a network that is the target of a penetration test, this prefix is fed to the algorithm. In some cases, prefix information can be retrieved from external sources such as WHOIS databases.

3. Wordlists. When generating candidate addresses, a list of possible hex-words (such as babe, cafe, 1337, b00b) is used.

The algorithm shown in the figure performs the following numbered steps:

1. **DNS resolving**
   Any DNS names that were given as input are resolved to IPv6 addresses.

2. **DNS info gathering**
   From DNS, more addresses can be gathered. By using brute force, or AXFR when available, more addresses from a target domain can be retrieved. When DNSSEC is in use with NSEC, the whole zone can be retrieved as with AXFR. In case NSEC3 is deployed, more effort must be made, but still large parts of the zone can be retrieved. The last technique is PTR traversal: if a reverse DNS zone has been configured for lookups of PTR records, under some conditions a feature of DNS can be abused to traverse the reverse zone, thus finding addresses. These techniques are described in more detail in the paper, with references to their original sources.

3. **Web info gathering**
   From online search engines and directories, extra information on possible DNS names and addresses is gathered. Search engines may have pages in cache that were served on the target network in the past. Online directories record visible DNS changes and overviews of virtual hosts hosted on different addresses.

4. **Address pool**
   The addresses that have been found so far are stored as well as used in the next steps.

5. **Subnet guessing**
   From the addresses that are known so far, subnets can be guessed. Addresses in the same network will have an overlap, starting at the MSB. An organization has a network range that it can use (a prefix), and within which it can create different subnets. By comparing the addresses found so far, assumptions about the prefix and subnets can be made. The part of the found addresses that is constant across all addresses and starts at the MSB, is probably the prefix. If information is known about the prefix, for instance from WHOIS databases, this can be used instead of guessing. If multiple addresses have an overlapping part after the prefix, one can assume this is the subnet. The subnets are needed when new candidates for alive detection are generated: guesses will be appended to known subnets.

22

6. **Vendor-ID extraction**
   If SLAAC addresses have been found, the Vendor-ID can be extracted from the last 64 bits, which are in modified EUI-64 format. This format includes the MAC address of the network interface the address belongs to, with the sequence 0xfffe inserted between the OUI and the NIC. The assumption can be made that more hardware from the same vendor is used, so only the NIC identifier of 24 bits needs to be guessed.

7. **Candidate address generation**
   This is the heart of the algorithm. From the found prefixes, vendor IDs, and addresses, new addresses can be generated that might be in use. The assumptions on numbering schemes that are mentioned in the paper are transformed into candidate addresses. The following rules can be used:

   - Generate new subnets by numbering around the current subnet numbers;
   - Generate low-byte numbered addresses in each subnet;
   - Generate hex-word numbered addresses in each subnet;
   - Generate IPv4-address based candidates in each subnet for all known IPv4 addresses of the site;
   - Generate service-based candidates for well-known services and services that have been found in the IPv4 scan if present;
   - For found addresses which have a at least 16 bits of consecutive zeroes in the last 64 bits, generate sequential addresses around them. Random generated addresses have an extremely small chance of containing 16 bits of zeroes, while sequential addresses often do. If the addresses were given out using DHCPv6 or sequentially by hand, this may result in finding surrounding hosts;
   - For found OUIs, generate candidates for the last 24 bits of the address in the prefixes where SLAAC addresses were found.

   Feedback from the next step (alive detection) is received on what candidates proved to be alive. Sequential address guessing can continue as long as alive hosts are found; once multiple consecutive candidates do not result in new alive hosts, generating candidates can stop for that sequence.

8. **Alive detection**
   The candidate addresses that are generated in the previous step are tested for aliveness. Multiple methods for alive detection are used, since no single method can guarantee detection of an alive host. Combining methods still does not give guarantee, but gives a higher chance of detecting alive hosts. Hosts are probed using the methods employed by the popular network scanner nmap, as described on `http://nmap.org/book/man-host-discovery.html`. Feedback on alive hosts is fed back to the previous step.

9. **Service discovery**

   On hosts that were found in the previous step and that were in the address pool, service discovery is performed. This takes place in the same way as for IPv4, by probing all TCP ports and certain UDP ports.
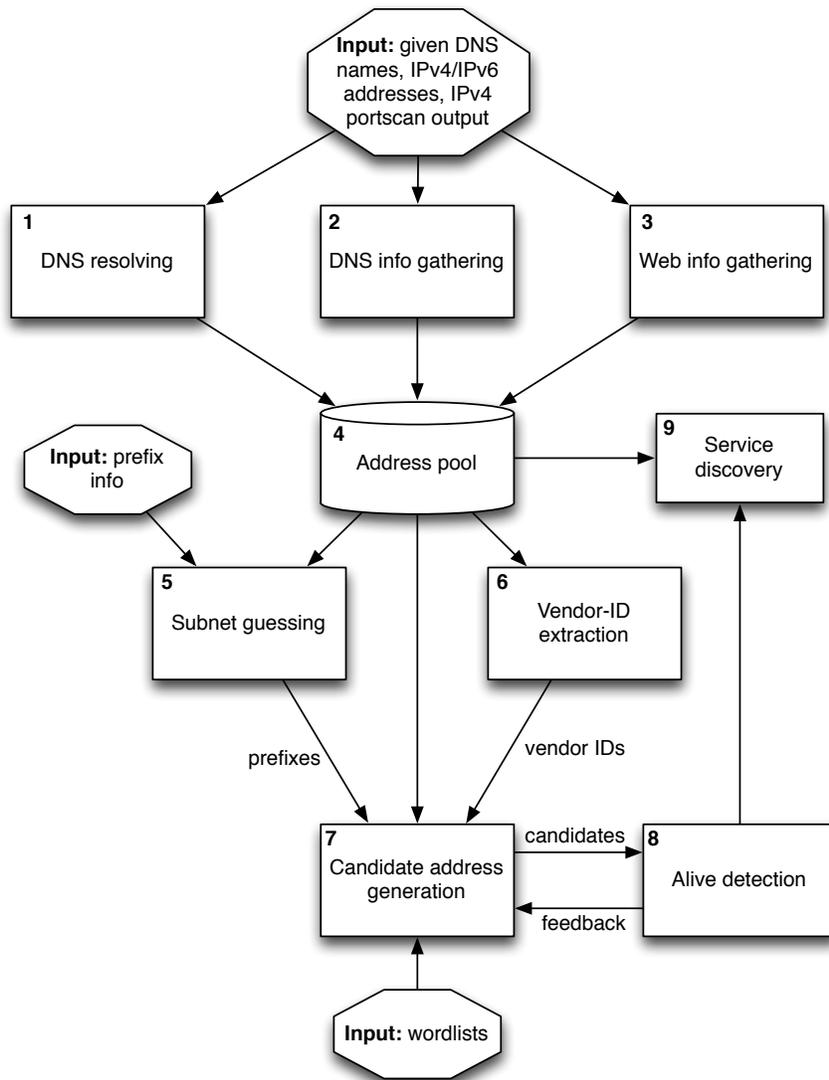
Figure A.1: IPv6 host discovery algorithm

A.2 Anti-automation algorithm

In IPv4 networks, automated attacks against services can usually be stopped by blocking the attacker's IP address. In some cases this will block more than just the attacker, for instance when a proxy server is used by multiple users of the service. In other cases the attacker will have multiple IP addresses, and can continue with another address once the first is blocked. However, since IPv4 addresses are becoming increasingly scarce, even advanced attackers will have a limited number of addresses available.

With IPv6, this is different. An attacker who has a small domestic prefix of size /64 has $2^{128-64} = 18446744073709551616$ addresses. By switching between IPv6 addresses, an attacker could avoid being blocked by an algorithm that works fine for IPv4. Therefore, a new way of blocking attackers is needed.

In the paper we suggest an algorithm that takes into account the large address space an attacker may have. In addition to blocking a single IP address, we suggest blocking larger prefixes. Once multiple incidents have occurred within a given prefix, the whole prefix can be blocked. This check can be performed for prefixes from small to large, with a different blocking threshold for each prefix size. The algorithms keeps state in the data structures shown in Table A.1. When in the context of the algorithm an IP is "blocked", this means that the algorithm will return a negative value when it is queried for the IP. The application can of course use this result in different ways than to literally block the visitor; a CAPTCHA can be shown or throttling can be applied.

Name	Members	Description
Threshold list	prefix size, threshold, taint timeout, block timeout	Per prefix size defined, a threshold for the number of taints is kept, as well as the timeout values for taints and blocks.
Taint list	address, counter, last taint	A list of tainted IP addresses with the number of taints and time of last taint
Block list	prefix, block expire time	A list of blocked prefixes with expiration time.

Table A.1: Data structures used by the algorithm

When a visitor requests use of a sensitive function, his IP address is checked against the list of blocks, to see if his address is contained in any of the prefixes. If so, the algorithms returns negative, otherwise positive.

IP address are "tainted" when a security-sensitive failure occurs, such as a failed login attempt. Upon subsequent failures, the taint count for the IP address is increased. Once a certain threshold of taints is reached, the IP address is added to a blocklist. Once a certain threshold of taints is reached within a prefix, the prefix is added to the blocklist. Once added to the blocklist, no new taints can occur in the prefix until the block has expired.

The algorithm is best explained by an implementation. The source code of an implementation has therefore been included in Listing 1 at the end of this section.

Table A.2 shows the parameters used in the algorithm, with example values for a fictional site. For each prefix size, the threshold of taints that is allowed is specified as nQX. The value of n is fixed for each prefix size, since different prefix sizes have a different typical use on the Internet. A /64 is the standard size for a subnet, since it is required to be able to perform automatic address assignment. The IETF recommends assigning a larger space than /64 to end-users, such as /56 or /48 [2] to give end-users room for multiple subnets. A /48 is also the assignment size for an organisation, and an ISP receives at least a /32 [4, 1]. Due to the difference in use, the values of n for different prefix sizes have no linear relation. For the same reason, the increments in prefix size are irregular.

The Q and X parameters provide a way to tune the algorithm to a specific situation. X represents the taints per IP before blocking it, Q is used as a factor for prefixes larger than /128 and provides a way to dampen the impact that changing X has on prefixes. A large site will allow more taints per prefix size to happen, since it has more users in any given prefix, so the parameters need to be chosen for a specific use case. The higher the parameters are chosen, the more attempts an attacker is allowed when he tries to perform an automated attack. On the other hand, if the parameters are chosen too low, legitimate use will be hindered. If the X factor is increased without adjusting Q, the threshold of allowed taints will increase slightly for single IPs and small prefixes, but the threshold for large prefixes will increase dramatically.

Table A.2 shows the parameters for a fictional site, using $X = 3$ and $Q = 10$. The values of X, Q and n are chosen as example values. More research is needed to establish values that work well for different types of sites and prefix sizes.

The code of an implementation of the algorithm in the Python programming language is included below. To improve readability, the code has been stripped from graceful error handling and the parsing of commandline options: only the bare algorithm is shown.

The entry points to the algorithm (tainting an IP and checking if an IP exists in the blocklist) are implemented in the `Blocker.taint()` and `Blocker.is_blocked()` methods.

```python
import datetime
import sys
import textwrap
import time
from ipaddr import *


class TaintVO:
    '''implements a taint list item'''

```

Pfx.Size	Taints	Example	T_{taint}	T_{blcok}	Comments
/128	X	3	10m	30m	Single IP
/64	$\frac{1}{2} * Q * X$	15	10m	30	Default subnet size, may affect multiple users
/56	$1 * Q * X$	30	30m	60m	Recommended site allocation
/48	$6 * Q * X$	180	60m	60m	Minimum PI assignment, default site allocation
/32	$30 * Q * X$	900	180m	120m	Minimum LIR assignment
/24	$60 * Q * X$	1800	180m	120m	Maximum LIR assignment not defined, but this shouldn't be reached in practice

Table A.2: Example parameters for IP-based restrictions for $X = 3$ and $Q = 10$

```python
10      def __init__( self, address, touched, counter ):
11          # IPv6Address
12          self.address = address
13          # datetime.datetime
14          self.touched = touched
15          # int
16          self.counter = counter
17
18  class BlockVO:
19      '''implements a block list item'''
20
21      def __init__( self, prefix, expiry ):
22          # IPv6Network
23          self.prefix = prefix
24          # datetime.datetime
25          self.expiry = expiry
26
27      def __cmp__(self, other):
28          if str(self.prefix) < str(other.prefix):
29              return -1
30          elif str(self.prefix) > str(other.prefix):
31              return 1
32          else:
33              return 0
34
35  class Blocker:
36      '''implements a blocklist that blocks aggregated prefixes
37      '''
```

28

```
38
39        blocklist = []
40        taintlist = []
41
42        #taints per IP
43        par_X = 0
44
45        #dampening factor for networks
46        par_Q = 0
47
48        _initDone = False
49
50        threshold_table = {
51            # prefixlen, taints, tainttime(s), blocktime(s)
52            # taints is just factor here, is replaced by number
53            # in __init__()  timeouts may be lowered in
54            # __init__() for testing purposes as well
55
56            '128' : [1, 600, 1800],
57            '64' : [.5, 600, 1800],
58            '56' : [1, 1800, 3600],
59            '48' : [6, 3600, 3600],
60            '32': [30, 10800, 7200],
61            '24' : [60, 10800, 7200]
62        }
63
64        def __init__(self, x, q, timediv):
65            self.par_X = x
66            self.par_Q = q
67            self._initDone = True
68
69            # convert the factors in the table to threshold based on X and Q
70            for pfx in self.threshold_table.keys():
71                vals = self.threshold_table[pfx]
72                vals[0] = int(self.par_Q * self.par_X * vals[0])
73                if timediv > 0:
74                    vals[1] = vals[1] / timediv
75                    vals[2] = vals[2] / timediv
76
77            # the /128 has a threshold of X
78            self.threshold_table['128'][0] = self.par_X
79
80        def taint(self, ip):
81            '''add ip to taintlist
82            or increase counter if already present
83            '''
```

29

```python
84
85          if not self._initDone:
86              return False
87
88          # first, purge the lists for expired items
89          self.purge_lists()
90
91          # check if ip is not already blocked. if it is, we're not
92          # being invoked correctly.
93          if self.is_blocked(ip):
94              debug("Not tainting %s, already in blocklist" % ip)
95              return False
96
97          addr = IPv6Address(ip)
98          now = datetime.datetime.today()
99
100         taintobj = None
101         for item in self.taintlist:
102             if item.address == addr:
103                 item.counter = item.counter + 1
104                 item.touched = now
105                 taintobj = item
106                 break
107         if not taintobj:
108             taintobj = TaintVO(addr,now,1)
109             self.taintlist.append(taintobj)
110
111         # we now check if the added taint has impact on the
112         # blocklist: should new blocks be installed?
113         # since ip is not in blocklist already, blocks
114         # will not be added double this way
115
116         # get prefixes as ints
117         prefixes = map( lambda x: int(x), self.threshold_table.keys())
118         prefixes.sort()
119         prefixes.reverse()
120
121         # try every prefix for taintobj
122         for pref in prefixes:
123
124             debug("Checking prefix %s for taint %s" % (pref, taintobj) )
125
126             # find the threshold for taint count
127             threshold = self.threshold_table[str(pref)][0]
128
129             # each prefix has its own validity period for taints
```

```python
130             valid_start = now - datetime.timedelta(
131                 seconds = self.threshold_table[str(pref)][1]
132             )
133             matches = []
134             pref_obj = IPv6Network( taintobj.address ).supernet(128 - pref)
135
136             # find other taints in this prefix that are valid
137             for inner_t in self.taintlist:
138                 if inner_t.address in pref_obj and (
139                         inner_t.touched > valid_start):
140                     debug("%s matches" % inner_t)
141                     matches.append(inner_t)
142
143             # matches contains a list of addresses matching taintobj for
144             # the current prefix
145
146             sum = 0
147             for m in matches:
148                 sum = sum + m.counter
149
150             debug("Sum is %d, threshold is %d" % (sum, threshold ) )
151             if sum > threshold:
152                 exp = now + datetime.timedelta(
153                     seconds = self.threshold_table[str(pref)][2]
154                 )
155
156                 blk = BlockVO( pref_obj, exp )
157                 for b in self.blocklist:
158                     if b == blk:
159                         next
160
161                 self.blocklist.append(blk)
162                 debug("Adding %s" % blk)
163         return True
164
165     def is_blocked(self, ip):
166         '''check if a given IP is blocked'''
167
168         if not self._initDone:
169             return False
170
171         # first, purge the lists for expired items
172         self.purge_lists()
173
174         addr = IPv6Address(ip)
175
```

```
176            for item in self.blocklist:
177                if addr in item.prefix:
178                    return True
179
180            return False
181
182        def purge_lists(self):
183            '''remove expired items from blocklist and taintlist'''
184
185            now = datetime.datetime.today()
186            yesterday = now - datetime.timedelta(days=1)
187
188            # taintitems can be safely removed after 24h as currently, largest
189            # timespan in table is 180m
190            for item in self.taintlist:
191                if item.touched < yesterday:
192                    self.taintlist.remove(item)
193
194            # blocklist items are removed after they've expired
195            for item in self.blocklist:
196                if now > item.expiry:
197                    self.blocklist.remove(item)
```

Listing 1: Source code of a sample implementation of the algorithm

Appendix B

Checklists

Pine structures its penetration testing process around the checklists by Certified Secure. These checklists are included in full in this chapter. In the paper, changes to the checklists were proposed. These changes are listed here, categorized by checklist.

Preparation phase (no checklist item)

When the customer has specified a network range instead of a set of IP addresses, this item will need to be changed. A new method for host discovery has been proposed in the paper, and described in more detail in Appendix A.

Basic Server Scan Checklist

No items on this checklist are affected by IPv6

Advanced Server Scan Checklist

Items 1.1 and 1.2: DNS

We have proposed the following changes to these items:

- query IPv6 nameservers in addition to IPv4 ones;

- query for AAAA records and to do reverse lookups for found IPv6 addresses;

- the wordlists used for brute-forcing should include IPv6-specific words

Item 2.2: Fragmentation

We have proposed the following changes to this item:

- Run the tests that are run against IPv4 targets (overlapping fragments, tiny fragments) against IPv6 targets as well, using the fragmentation extension header;

- Use other extension headers to fill the first fragment, while keeping the upper-layer header in the second fragment.

New: Bypassing ACLs using extension headers

Due to limitations in the forwarding plane of some high-end routers, router-based ACLs may be bypassed by using extension headers. A check for this vulnerability can be added to this checklist.

New: Exposed hosts due to use of transition mechanisms

Different transition mechanisms bring the risk of exposing internal hosts to the outside world. Furthermore, existing security policies may not be applied to hosts that are exposed through transition mechanisms. We propose to add a check for exposed hosts to this checklist.

New: Bypassing firewalls using routing headers

By using the IPv6 routing extension header, firewalls can be circumvented on networks that support this header. We propose adding a check for vulnerable configurations in this checklist.

Basic Web Application Scan Checklist

No items on this checklist are affected by IPv6

Advanced Web Application Scan Checklist

Items 6.5, 6.6, 6.7 and 10.1: Anti-automation

The recommendation for the items 6.5, 6.6, 6.7 and 10.1 includes throttling requests based on IP address, or displaying a CAPTCHA when multiple requests are received from the same IP address. Since with IPv6 an attacker is likely to have many more IP addresses than with IPv4, the recommendation must be changed to employ a throttling/blocking algorithm that deals with this increase in IPs. Such an algorithm has been proposed in the paper, and described in more detail in Appendix A.

Item 11.6: Session tracking

The checklist recommends that the use of a session in a web application be restricted to one IP address only. With the use of privacy extensions in IPv6, this will render sessions unusable after a short time for many users. We recommend restricting sessions to the /64 prefix the IP address is in, instead of the whole address.

Checklists

On the following pages, the checklists for basic and advanced server and web application scans are included.

Certified Secure Basic Server Scan Checklist

About

This checklist is made freely available by Certified Secure. For Certified Specialists an annotated version is available in the Portal. Certified Secure also provides training and certification based on this checklist, visit www.certifiedsecure.com or contact info@certifiedsecure.com for more information.

Scope

This checklist should be used as a guideline when remotely assessing the basic security of a server. When this checklist is completed without incident, the Advanced Server Scan Checklist can be used to perform a more thorough security scan.

Usage

Every test on the checklist should be performed or explicitly marked as being not applicable. Once a test is completed the checklist should be updated with the appropriate result icon and an optional document cross reference. The filled-in checklist should not be delivered stand-alone but should be incorporated in a document specifying at least the results, scope and context of the performed tests.

License

Result Icon Legend

Icon	Explanation
✅	Test was performed and results are okay
❌	Test was performed and results require attention
🔲	Test was not applicable

#	Certified Secure Basic Server Scan Checklist	Result	Ref
1.0	**Network security**		
1.1	Check for extraneous open TCP/UDP ports		
2.0	**Version management**		
2.1	Check available services for missing security updates		
2.2	Check available services for unsupported software		
3.0	**User accounts and policies**		
3.1	Check for default and/or predictable accounts		
4.0	**Mail service configuration**		
4.1	Check for open relays		
5.0	**FTP service configuration**		
5.1	Check for anonymous uploading		
6.0	**Miscellaneous**		
6.1	Check for server specific problems		

Certified Secure Advanced Server Scan Checklist

About

This checklist is made freely available by Certified Secure. For Certified Specialists an annotated version is available in the Portal. Certified Secure also provides training and certification based on this checklist, visit www.certifiedsecure.com or contact info@certifiedsecure.com for more information.

Scope

This checklist should be used when the Basic Server Scan Checklist is completed without incident and a more thorough remote security scan is desired. This checklist must always be used and presented as an extension of the Basic Server Scan Checklist.

Usage

This checklist must only be used once the Basic Server Scan Checklist is completed without incident, the Basic Server Scan Checklist and related results should always be included when presenting the results of this checklist.

Every test on the checklist should be performed or explicitly marked as being not applicable. Once a test is completed the checklist should be updated with the appropriate result icon and an optional document cross reference. The filled-in checklist should not be delivered stand-alone but should be incorporated in a document specifying at least the results, scope and context of the performed tests.

License

This work is licensed under a Creative Commons Attribution-No Derivative Works 3.0 Netherlands License. The complete license text can be found online at http://creativecommons.org/licenses/by-nd/3.0/nl/. Contact Certified Secure if you want to receive a printed copy.

Result Icon Legend

Icon	Explanation
✅	Test was performed and results are okay
❌	Test was performed and results require attention
🔲	Test was not applicable

#	Certified Secure Advanced Server Scan Checklist	Result	Ref
1.0	**Network Information**		
1.1	Check for AXFR transfers		
1.2	Check for sensitive information in the domain name system		
2.0	**Firewall**		
2.1	Check for firewall evasion by using special TCP flags		
2.2	Check for firewall evasion by using IP fragmentation		
2.3	Check for firewall evasion by using special source ports		
3.0	**User accounts and policies**		
3.1	Check available services for sensitive information		
4.0	**Mail service configuration**		
4.1	Check for mail service username enumeration		
5.0	**Web service configuration**		
5.1	Check for extraneous directory listings		
5.2	Check for too verbose error messages		
5.3	Check for installed "example" scripts		
5.4	Check for extraneous virtual hosts		
5.5	Check for TRACK and TRACE methods		
5.6	Check for internal IP addresses in header fields		
6.0	**Cryptography**		
6.1	Check for insecure SSL ciphers		
6.2	Check for insecure SSL certificates		
7.0	**Miscellaneous**		
7.1	Check for server specific problems		

info@certifiedsecure.com
Tel.: +31 (0)70 310 13.40
Fax: +31 (0)70 310 13.41
Loire 128-A
2491 AJ The Hague
The Netherlands

Certified Secure Basic Web Application Scan Checklist

About

This checklist is made freely available by Certified Secure. For Certified Specialists an annotated version is available in the Portal. Certified Secure also provides training and certification based on this checklist, visit www.certifiedsecure.com or contact info@certifiedsecure.com for more information.

Scope

This checklist should be used as a guideline when remotely assessing the basic security of a web application. When this checklist is completed without incident, the Advanced Web Application Scan Checklist can be used to perform a more thorough security scan.

Usage

Every test on the checklist should be performed or explicitly marked as being not applicable. Once a test is completed the checklist should be updated with the appropriate result icon and an optional document cross reference. The filled-in checklist should not be delivered stand-alone but should be incorporated in a document specifying at least the results, scope and context of the performed tests.

License

This work is licensed under a Creative Commons Attribution-No Derivative Works 3.0 Netherlands License. The complete license text can be found online at http://creativecommons.org/licenses/by-nd/3.0/nl/. Contact Certified Secure if you want to receive a printed copy.

Result Icon Legend

Icon	Explanation
✅	Test was performed and results are okay
❌	Test was performed and results require attention
🔲	Test was not applicable

#	Certified Secure Basic Web Application Scan Checklist	Result	Ref
1.0	**Authentication and Authorization**		
1.1	Check for client side authentication		
1.2	Check for default and predictable accounts		
1.3	Check for identifier based authorization		
2.0	**User Input**		
2.1	Check for filename injection / path traversal		
2.2	Check for SQL injection		
2.3	Check for cross site scripting		
2.4	Check for system command injection		
3.0	**File Upload**		
3.1	Check for uploading of (dynamic) scripts		
4.0	**Sessions**		
4.1	Check for Cross Site Request Forgery		
5.0	**Miscellaneous**		
5.1	Check for application or setup specific problems		

info@certifiedsecure.com
Tel.: +31 (0)70 310 13.40
Fax: +31 (0)70 310 13.41
Loire 128-A
2491 AJ The Hague
The Netherlands

Certified Secure Advanced Web Application Scan Checklist

About

This checklist is made freely available by Certified Secure. For Certified Specialists an annotated version is available in the Portal. Certified Secure also provides training and certification based on this checklist, visit www.certifiedsecure.com or contact info@certifiedsecure.com for more information.

Scope

This checklist should be used when the Basic Web Application Scan Checklist is completed without incident and a more thorough remote security scan is desired. This checklist must always be used and presented as an extension of the Basic Web Application Scan Checklist.

Usage

This checklist must only be used once the Basic Web Application Scan Checklist is completed without incident, the Basic Web Application Scan Checklist and related results should always be included when presenting the results of this checklist.

Every test on the checklist should be performed or explicitly marked as being not applicable. Once a test is completed the checklist should be updated with the appropriate result icon and an optional document cross reference. The filled-in checklist should not be delivered stand-alone but should be incorporated in a document specifying at least the results, scope and context of the performed tests.

License

This work is licensed under a Creative Commons Attribution-No Derivative Works 3.0 Netherlands License. The complete license text can be found online at http://creativecommons.org/licenses/by-nd/3.0/nl/. Contact Certified Secure if you want to receive a printed copy.

Result Icon Legend

Icon	Explanation
✅	Test was performed and results are okay
❌	Test was performed and results require attention
🔲	Test was not applicable

#	Certified Secure Advanced Web Application Scan Checklist	Result	Ref
1.0	**Multi-system Services**		
1.1	Check for HTTP request smuggling		
2.0	**Design**		
2.1	Check for extraneous files in document root		
3.0	**Information Disclosure**		
3.1	Check for too verbose error messages		
3.2	Check for debug enabling using a predictable parameter		
3.3	Check for valuable information in robots.txt		
3.4	Check for accessible CVS/SVN directories		
3.5	Check for accessible configuration directories		
3.6	Check for accessible backup files		
3.7	Check for accessible non-parsed dynamic scripts		
4.0	**Privacy and Confidentiality**		
4.1	Check for missing anti-caching headers		
4.2	Check for unencrypted transmissions of sensitive information		
4.3	Check for sensitive information stored in cookies		
4.4	Check for sensitive information in externally archived pages		
5.0	**Integrity**		
5.1	Check for client side state management		
6.0	**Authentication and Authorization**		
6.1	Check for missing authentication		
6.2	Check for authentication based on the knowledge of a secret URL		
6.3	Check for identifier based authentication		
6.4	Check for too verbose authentication-failure logging		
6.5	Check for brute-force username enumeration		
6.6	Check for brute-force password guessing		
6.7	Check for denial of service by locking out accounts		
6.8	Check for authentication or authorization based on obscurity		

#	Certified Secure Advanced Web Application Scan Checklist	Result	Ref
7.0	**User Input**		
7.1	Check for double decoding of headers / parameters		
7.2	Check for XML injection		
7.3	Check for XPath injection		
7.4	Check for LDAP injection		
7.5	Check for HTTP header injection		
7.6	Check for XSL(T) injection		
7.7	Check for SSI injection		
7.8	Check for resource identifier injection		
7.9	Check for dynamic scripting injection		
7.10	Check for regular expression injection		
8.0	**XML**		
8.1	Check for XML external entity parsing		
8.2	Check for XML external DTD parsing		
9.0	**File Upload**		
9.1	Check for uploading outside of intended directory		
9.2	Check for incorrect handling of very large files		
9.3	Check for local file disclosure via upload filename		
9.4	Check for uploading of configuration files		
10.0	**Email**		
10.1	Check for automated spamming via (feedback) scripts		
11.0	**Sessions**		
11.1	Check for session-cookies without the secure flag		
11.2	Check for session-cookies without the httponly flag		
11.3	Check for predictable session-ids		
11.4	Check for session collisions		
11.5	Check for session-fixation		
11.6	Check for external session-hijacking		
11.7	Check for insecure transmission of session-cookies		

#	Certified Secure Advanced Web Application Scan Checklist	Result	Ref
11.8	Check for missing session revocation if session-id transmitted unencrypted		
12.0	**Cryptography**		
12.1	Check for unproven cryptographic algorithms		
13.0	**Miscellaneous**		
13.1	Check for application or setup specific problems		