College of Engineering Trivandrum

Department of Computer Science and Engineering

Selfrando: Securing the Tor Browser against De-anonymization Exploits

By

Krishnakumar.C.N

Roll No.:34, S7 R

August 24, 2016

Contents

List of Figures

1 Introduction

The Tor network is a group of volunteer-operated servers that allows people to improve their privacy and security on the Internet. The Tor Project provides a suite of free software and a worldwide network designed to facilitate anonymous information exchange and to prevent surveillance and fingerprinting of these interactions. The Tor network is open to anyone and widely used by civil rights activists, journalists, military etc. Many sensitive websites, including the late Silk Road black market, are only accessible over Tor.

The Tor Network is continually facing de-anonymization attacks by law enforcement, intelligence agencies, and foreign nation states. A de-anonymization attack aims to disclose information, such as the identity or the location, of an anonymous user. A practical way to de-anonymize Tor users is to exploit security vulnerabilities in the software used to access the Tor network. The most common way to access Tor is via the Tor Browser (TB). In 2013, the Federal Bureau of Investigation (FBI) exploited a known software vulnerability in Firefox to de-anonymize Tor users that had not updated to the most recent version of TB. In early 2016, it was confirmed that the FBI continues to monitor the Tor network, this time using a de-anonymization attack devised by Carnegie Mellon University researchers.

The usual defenses against the attacks were ASLR and ASan. ASan imposes high run time overhead.ASLR is very efficient. However, ASLR was used because it is widely supported by compilers and operating systems, not for its security properties. ASLR randomly choose base address of stack, heap and code segment. ASLR can be made significantly stronger by randomizing not just the base address of modules but also the code inside each module.

Selfrando- a load-time randomization technique that improves security over ASLR while preserving the features that enabled ASLR's widespread adoption.The use of load-time function layout permutation ensures that the attack surface changes from one run to another. Load-time randomization also ensures compatibility with code signing and distribution mechanisms that use caching to efficiently serve millions of users.Selfrando is the first approach that avoids risky binary rewriting or the need to use a custom compiler, and instead works with existing build tools.

Selfrando reduces the impact of information leakage vulnerabilities and increases entropy relative to ASLR, making selfrando more effective against guessing attacks. The use of load-time randomization mitigates threats from attackers observing binaries during download or after the executable files have been stored on disk.

2 The Onion Router

2.1 Overview

The Tor network is a group of volunteer-operated servers that allows people to improve their privacy and security on the Internet. Tor's users employ this network by connecting through a series of virtual tunnels rather than making a direct connection, thus allowing both organizations and individuals to share information over public networks without compromising their privacy. Along the same line, Tor is an effective censorship circumvention tool, allowing its users to reach otherwise blocked destinations or content. Tor can also be used as a building block for software developers to create new communication tools with built-in privacy features.

Individuals use Tor to keep websites from tracking them and their family members, or to connect to news sites, instant messaging services, or the like when these are blocked by their local Internet providers. Tor's hidden services let users publish web sites and other services without needing to reveal the location of the site. Individuals also use Tor for socially sensitive communication: chat rooms and web forums for rape and abuse survivors, or people with illnesses.

Journalists use Tor to communicate more safely with whistleblowers and dissidents. Nongovernmental organizations (NGOs) use Tor to allow their workers to connect to their home website while they're in a foreign country, without notifying everybody nearby that they're working with that organization.

Groups such as Indymedia recommend Tor for safeguarding their members' online privacy and security. Activist groups like the Electronic Frontier Foundation (EFF) recommend Tor as a mechanism for maintaining civil liberties online. Corporations use Tor as a safe way to conduct competitive analysis, and to protect sensitive procurement patterns from eavesdroppers. They also use it to replace traditional VPNs, which reveal the exact amount and timing of communication. Which locations have employees working late? Which locations have employees consulting job hunting websites? Which research divisions are communicating with the company's patent lawyers? A branch of the U.S. Navy uses Tor for open source intelligence gathering, and one of its teams used Tor while deployed in the Middle East recently. Law enforcement uses Tor for visiting or surveillance web sites without leaving government IP addresses in their web logs, and for security during sting operations. [1]

2.2 Evolution

The core principle of Tor, "onion routing", was developed in the mid-1990s by United States Naval Research Laboratory employees, mathematician Paul Syverson and computer scientists Michael G. Reed and David Goldschlag, with the purpose of protecting U.S. intelligence communications online. Onion routing was further developed by DARPA in 1997. [1]

The alpha version of Tor, developed by Syverson and computer scientists Roger Dingledine and Nick Mathewson and then called The Onion Routing project, or TOR project, launched on 20 September 2002. On 13 August 2004, Syverson, Dingledine and Mathewson presented "Tor: The Second-Generation Onion Router" at the 13th USENIX Security Symposium. In 2004, the Naval Research Laboratory released the code for Tor under a free license, and the Electronic Frontier Foundation (EFF) began funding Dingledine and Mathewson to continue its development.

In December 2006, Dingledine, Mathewson and five others founded The Tor Project, a Massachusetts-based 501(c)(3) research-education nonprofit organization responsible for maintaining Tor. The EFF acted as The Tor Project's fiscal sponsor in its early years, and early financial supporters of The Tor Project included the U.S. International Broadcasting Bureau, Internews, Human Rights Watch, the University of Cambridge, Google, and Netherlands-based Stichting NLnet. [1]

2.3 Why we need Tor

Using Tor protects you against a common form of Internet surveillance known as "traffic analysis." Traffic analysis can be used to infer who is talking to whom over a public network. Knowing the source and destination of your Internet traffic allows others to track your behavior and interests. This can impact your checkbook if, for example, an e-commerce site uses price discrimination based on your country or institution of origin. It can even threaten your job and physical safety by revealing who and where you are.

For example, if you're travelling abroad and you connect to your employer's computers to check or send mail, you can inadvertently reveal your national origin and professional affiliation to anyone observing the network, even if the connection is encrypted.

Internet data packets have two parts: a data payload and a header used for routing. The data payload is whatever is being sent, whether that's an email message, a web page, or an audio file. Even if you encrypt the data payload of your communications, traffic analysis still reveals a great deal about what you're doing and, possibly, what you're saying. That's because it focuses on the header, which discloses source, destination, size, timing, and so on.

A basic problem for the privacy minded is that the recipient of your communications can see that you sent it by looking at headers. So can authorized intermediaries like Internet service providers, and sometimes unauthorized intermediaries as well. A very simple form of traffic analysis might involve sitting somewhere between sender and recipient on the network, looking at headers.

But there are also more powerful kinds of traffic analysis. Some attackers spy on multiple parts of the Internet and use sophisticated statistical techniques to track the communications patterns of many different organizations and individuals. Encryption does not help against these attackers, since it only hides the content of Internet traffic, not the headers. [1]

2.4 Working of Tor

Tor helps to reduce the risks of both simple and sophisticated traffic analysis by distributing your transactions over several places on the Internet, so no single point can link you to your destination. The idea is similar to using a twisty, hard-to-follow route in order to throw off somebody who is tailing you — and then periodically erasing your footprints. Instead of taking a direct route from source to destination, data packets on the Tor network take a random pathway through several relays that cover your tracks so no observer at any single point can tell where the data came from or where it's going. [2]

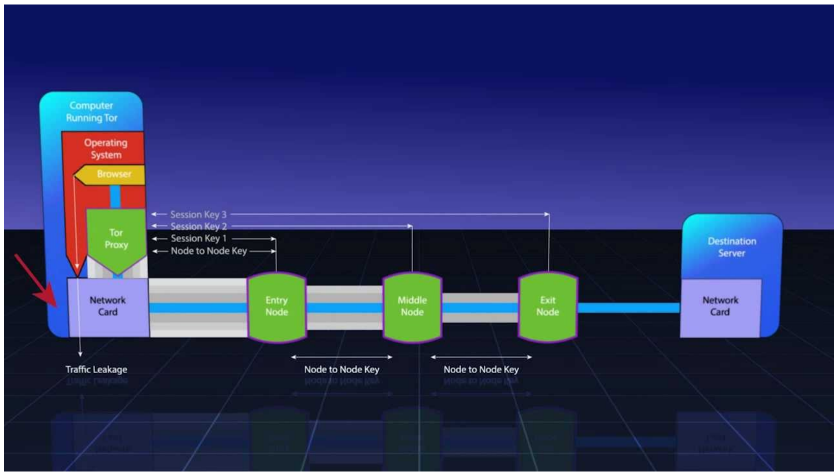

Figure 1: How Tor Works

To create a private network pathway with Tor, the user's software or client incrementally builds a circuit of encrypted connections through relays on the network. The circuit is extended one hop at a time, and each relay along the way knows only which relay gave it data and which relay it is giving data to. No individual relay ever knows the complete path that a data packet has taken. The client negotiates a separate set of encryption keys for each hop along the circuit to ensure that each hop can't trace these connections as they pass through. [2] Once a circuit has been established, many kinds of data can be exchanged and several different sorts of software applications can be deployed over the Tor network. Because each relay sees no more than one hop in the circuit, neither an eavesdropper nor a compromised relay can use traffic analysis to link the connection's source and destination. Tor only works for TCP streams and can be used by any application with SOCKS support.

For efficiency, the Tor software uses the same circuit for connections that happen within the same ten minutes or so. Later requests are given a new circuit, to keep people from linking your earlier actions to the new ones.

3 De-anonymization Attacks

A de-anonymization attack aims to disclose information, such as the identity or the location, of an anonymous user.

While many de-anonymization attacks rely on weaknesses in the network protocol, they often require that adversaries control a large number of Tor nodes or only work in a lab environment. An alternative and practical way to de-anonymize Tor users is to exploit security vulnerabilities in the software used to access the Tor network.The most common way to access Tor is via the Tor Browser (TB), which includes a pre-configured Tor client. Since TB is based on Mozilla's Firefox browser, they share a large part of their attack surfaces.

In 2013, the Federal Bureau of Investigation (FBI) exploited a known software vulnerability in Firefox to de-anonymize Tor users that had not updated to the most recent version of TB. Due to the success of this operation,exploit brokers and, presumably, governments and criminals are currently soliciting exploits for the TB. [3]

In early 2016, it was confirmed that the FBI continues to monitor the Tor network, this time using a de-anonymization attack devised by Carnegie Mellon University researchers [3]

4 Exploiting memory Corruption

Unlike modern programming languages, C and C++ rely on manual memory management, trading reliability for flexibility and performance. Hence, memory management errors often create vulnerabilities that can be exploited to hijack control flow and perform other malicious operations that were never intended by the program authors.

4.1 Buffer Overflow Attack

Traditionally, attackers used a buffer overflow to directly inject malicious code into a program and execute it.A buffer overflow occurs when data written to a buffer also corrupts data values in memory addresses adjacent to the destination buffer due to insufficient bounds checking. This can occur when copying data from one buffer to another without first checking that the data fits within the destination buffer.However, the introduction of the $W \oplus X$ policy that requires memory pages to either be writable or executable, but not both, made most code-injection attacks obsolete. [4]

4.2 Code Reuse Attack

As W $\oplus$ X became commonplace,attackers changed their tactics from code injection to code reuse. These attacks reuse existing, legitimate code for malicious purposes and have therefore proven far harder to stop than code injection.

4.2.1 Return-into-libc attack

A return-to-libc attack usually starting with a buffer overflow in which a subroutine return address on a call stack is replaced by an address of a subroutine that is already present in the process's executable memory and ridding the attacker of the need to inject their own code.Although the attacker could make the code return anywhere, libc is the most likely target, as it is almost always linked to the program, and it provides useful calls for an attacker (such as the system function used to execute shell commands.) [5]

4.2.2 Return-oriented programming

An attacker gains control of the call stack to hijack program control flow and then executes carefully chosen machine instruction sequences, called "gadgets" Each gadget typically ends in a return instruction and is located in a subroutine within the existing program and/or shared library code. Chained together, these gadgets allow an attacker to perform arbitrary operations on a machine employing defenses that prevent simpler attacks. [6]

5 Preventing Code Reuse Exploits

To successfully mount a code-reuse attack, several requirements must be met. First, the application must contain a memory corruption vulnerability that allows control flow to be hijacked. Techniques such as control- flow integrity and stack canaries make control-flow hi- jacking harder but do not prevent it outright

Another key requirement is knowledge of the absolute addresses of the gadgets to reuse. In principle, ASLR prevents adversaries from knowing the absolute locations of ROP gadgets

5.1 Address Space Layout Randomization

ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.Randomizing the address-space layout of a software program prevents attackers from using the same exploit code effectively against all instantiations of the program containing the same flaw. The attacker must either craft a specific exploit for each instance of a randomized program or perform brute force attacks to guess the address-space layout. Brute force attacks are supposedly prevented by constantly randomizing the address-space layout each time the program is restarted.

However, since ASLR only randomizes the base address of a library, adversaries still know the relative positions of all functions inside a library. Using this knowledge together with a leaked code pointer, attackers can compute the absolute addresses of all functions in the same library. [7]

5.2 Address Space Layout Permutations

ASLP permutes all sections including code and static data in the program address space.Permuting the functions inside a library removes attackers' knowledge of the relative function layout inside each library, and additionally improves entropy by allowing an exponentially higher number of code layouts in comparison to ASLR. [8]

6 Selfrando

Selfrand is an enhanced and practical load-time randomization technique that defends against de-anonymization exploits. [9]

6.1 Design Objectives

The main objective is to substantially raise the costs for attackers to exploit memory-corruption vulnerabilities. The design support complex C/C++ programs(e.g., a browser) without modifying their source code, i.e., should avoid any modification to compilers, linkers, and other operating system components.To be applicable to privacy-preserving open-source tools, the design must not rely on any third-party proprietary software. Finally, the solution should not substantially increase the size of the program in memory or on disk.

6.2 Selfrando Design

Load-time Randomization over Compile-time Randomization

The easiest way to perform fine-grained code randomization is by customizing the compiler to take a seed value and generate a randomized binary.But, compiling and distributing a unique binary for each is impractical for introducing diversity among a population of programs. In the context of privacy-preserving software such as TB, compile-time randomization raises additional challenges. Randomized builds would complicate the deterministic build process, which is important to increase trust in the distributed binary.Moreover, compile-time randomization would

(a) Increase the feasibility of a de-anonymization attack due to individual, observable characteristics of a particular build, and

(b) Allow an attacker to build knowledge of the memory layout across application restarts, since the layout would be fixed.

With more implementation effort, randomization can delay until load-time which has several benefits. Most importantly, software vendors only need to compile and test a single binary. A single binary also means that users can continue to use hashes to verify the authenticity of the downloaded binary. Finally, modern content delivery networks rely extensively on caching binaries on servers; this optimization is no longer possible with unique binaries.

Framework

The framework makes the program binary randomize itself at load time. Function permutation (ASLP)is chosen as the randomization granularity, since it dramatically increases the entropy of the code layout while imposing the same low overheads as ASLR. Since discovering function boundaries at load-time by analyzing the program binary is unreliable and does not scale to large programs, pre-computing these boundaries statically and storing the necessary information in each binary is the technique used and is called Translation and Protection (TRaP) information. [9]

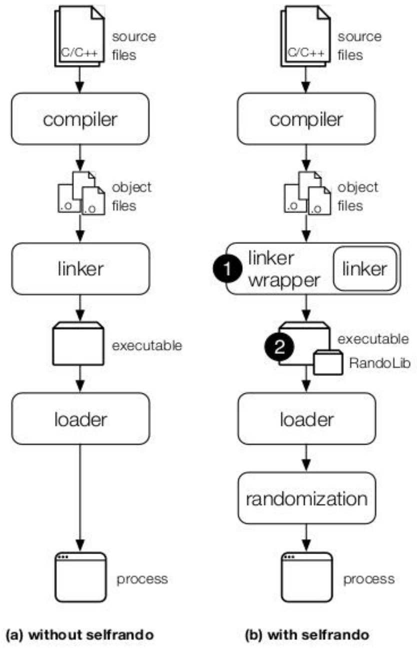

Figure 2: Building and running applications

Rather than modifying the compiler or linker, a small tool which wraps the system linker, extracts all function boundaries from the object files used to build the binary, then appends the necessary TRaP information to the binary itself is developed. The linker wrapper works with the standard compiler toolchains on Linux and Windows and only requires a few changes to the build scripts to use with the TB. Figure 2a [9] represents the usual workflow from the C/C++ source code to a running program. Figure 2b represents the modified work flow with selfrando. A linker wrapper intercepts calls to the linker and calls selfrando to gather information on the executable file.Then, it embeds TRaP information and a load-time randomization library, RandoLib, into the binary file. When the loader loads the application, it will invoke RandoLib instead of the entry point of the application. RandoLib will randomize the order of the functions in memory and then transfer control to the original program entry point.

7 Implementation

7.1 Extracting TRaP Information

When a module is loaded,selfrando permutes the order of all its functions. To do so, selfrando requires accurate information about function boundaries. If this information is not accurate, shuffling the function layout may inadvertently introduce errors that prevent correct execution of the application. After a function is moved, all references and pointers to this function, e.g., the target address of a call, become invalid because they still reference the old address. Hence, selfrando needs to update all references to the moved function, and therefore requires, for each function, a complete list of all locations that reference that function.

Such information is present in the intermediate object files. Since this meta data is usually not required during execution, the linker strips it from the final binary. Our linker wrapper therefore intercepts the linking process to extract function boundaries and references and embeds this information for use at load time. However, object files do not explicitly mark all function references. Specifically, we found that in some cases the compiler optimizes the code by inserting direct jumps between two functions. Such references are not marked with an explicit relocation because they are already resolved by the compiler.This can be disabled with a compiler option causing the compiler to place each function in a separate section.Since the compiler marks all references between sections, we can then see all function references. While enabling this option slightly increases build-time (0.07%), it also enables a linker optimization which increases locality.

After selfrando extracts the necessary metadata from each generated object, it adds an additional linker argument that instructs the linker to generate a map file, which is a text file that contains the memory layout of the final binary. Using the meta data and the map file, selfrando can compute the final location of each function in the executable file and all references to these functions. [9]

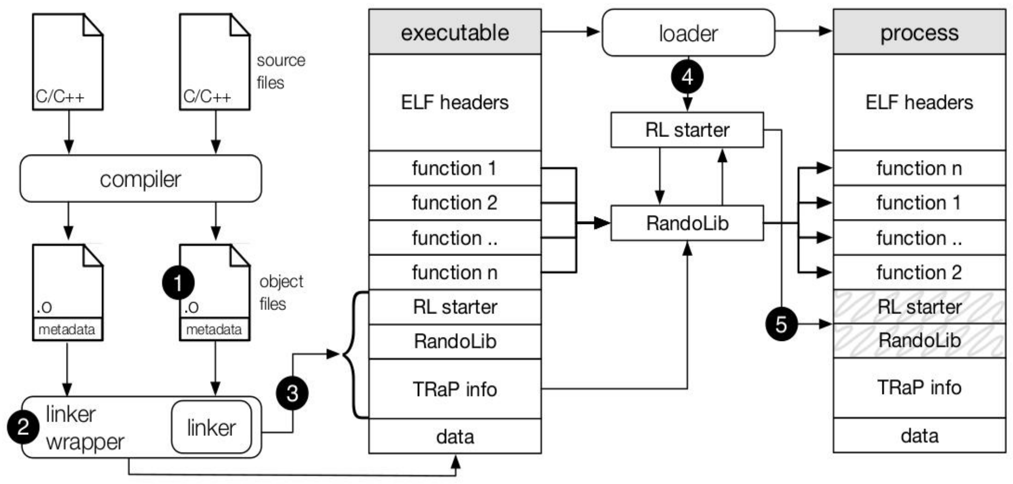

Figure 3: Workflow of Selfrando

7.2 Embedding TRaP information

The TRaP info is used by RandoLib, in the executable to make selfrando self-contained.However embedding the data in a space-efficient and binary format compatible way without modifying the linker is challenging.The main reasons are that

(1) Some of the meta data is only available after linking is complete, and

(2) We cannot pre-allocate space for the data since the exact amount of space needed is unknown until linking is done.

In particular, the start address of each function in the linked binary is determined by the order and final addresses of the object files in the binary, and therefore unknown until all objects are linked.

To add additional data to the final binary, we have to resort to a trick that involves changing the linker input so that it adds an empty segment header in the beginning of the binary.The linker creates a segment header which contains segment meta data, e.g., size and memory permissions, for each segment. The loader uses this meta data to load each segment of the binary into memory. Due to the structure of the binary format, adding an empty segment header in the beginning of the binary enables selfrando to append an arbitrary amount of data. When the linker is finished, we append the TRaP info and RandoLib to the end of the binary and set the values of the empty segment header accordingly. Finally, we change the start address of the binary to RandoLib. Hence, after the loader loads the binary into memory, it will transfer control to RandoLib, which then performs the function permutation [9]

7.3 Load-time Function Permutation

RandoLib performs function permutation using the embedded TRaP info, and consists of two parts: a small helper stub and the main randomization module. The purpose of the helper stub (RL Starter in Figure 3) is to make all selfrando data inaccessible after RandoLib finishes. The operating system loader calls this stub, invoking RandoLib as the first step of program execution.

The function permutation algorithm proceeds in several steps. First, RandoLib generates a random order for the functions using the Fisher-Yates shuffling algorithm. Second, RandoLib uses the embedded meta data to fix all references that became invalid during the randomization. Finally, after RandoLib returns, the helper stub makes selfrando's data inaccessible, and jumps to the original entry point of the binary.

7.4 Stack Unwinding

During program execution, the program stack is divided into stack frames. Each stack frame corresponds to a function call and consists of local variables, the return address, and arguments which were passed to the callee. Stack unwinding is the process of iterating through all active stack frames, starting from the most recent. It is mainly used for stack traces and exception handling, as both require access to previous stack frames. Exception handling uses stack unwinding to find the exception handler for a given exception after the program has thrown an instance of that exception. [9]

Traditionally, stack unwinding is supported by chaining stack frames as a singly-linked list, where each stack frame includes a pointer to the previous stack frame. The head of the linked list is stored in a dedicated register called the base pointer (BP). When a new stack frame is added, the called function saves the BP register of the caller on the stack, then overwrites the BP register to point to the current stack frame.

Modern compilers omit the frame pointer for optimized code to reduce memory usage on the stack and free another register for general purpose computations. To still support stack unwinding, compilers generate additional meta data which can be used to identify individual stack frames. Function permutation invalidates function references inside the stack unwinding meta data, so RandoLib updates them.

7.5 AddressSanitizer

AddressSanitizer (ASan) is used to detect memory corruption bugs in the hardened releases of Tor Browser. To allow selfrando to be deployed on TB,selfrando needs to work correctly with ASan. [9]

Selfrando does not interfere with the normal operation of ASan. When ASan detects a memory corruption, it generates a stack trace, which is supported by selfrando. To help troubleshoot memory corruption bugs, ASan annotates the stack trace with symbolic information. Specifically, it uses a symbolizer to obtain the function name and the source code location of every address in the stack trace. [10] After selfrando randomizes the order of functions, the symbolizer can no longer correctly map the stack addresses to function names. We restore the symbolizer's ability to annotate stack traces by emitting a map file that stores the original and actual address of each randomized function. We modify the symbolizer of ASan to use the emitted mapping to map the addresses of the stack trace to the original address

8 Experimental Evaluation

8.1 Security Analysis

8.1.1 Randomization Entropy

For any randomization scheme the amount of entropy provided is critical, because a low randomization entropy enables an attacker to guess the randomization secret with high probability.

ASLR provides up to 9 bits of entropy on 32 bit systems and up to 29 bits of entropy on 64 bit systems. While the ASLR offset on 32 bit systems is guessable in a reasonable amount of time, such attacks become in-feasible on 64 bit systems because the address space is that much larger. However, an attacker can bypass ASLR by leaking the offset that the code is loaded at in memory through a pointer into application memory. Once this offset is known the attacker can infer any address within the application, because it is used to shift the address of the whole application.Selfrando, on the other hand, applies more fine-grained function permutation. This means the randomization entropy does not depend on the size of the address space, as it is the case for ASLR, but on the number of functions in the randomized binary

8.1.2 Real-time Exploits

The real world attack scenario :

Nearly all modern attacks exploit heap-based vulnerabilities, despite the existence of stack vulnerabilities. However, whether a vulnerability can be exploited to launch a code-reuse attack depends on different factors, like how reliably the vulnerability can be triggered and the present mitigation techniques. Today, most stack-based vulnerabilities are not exploitable because they are mitigated by modern stack defenses. [4]

Information disclosure attacks are often limited to leaking heap memory because they access memory relative to the address of the vulnerable memory object. A buffer overread, for example, can be exploited to disclose consecutive memory which might contain interesting pointers, whereas a use-after-free vulnerability can be exploited to disclose interesting pointers of the freed object. In both cases the attacker is not able to (repeatedly) disclose absolute, and therefore, arbitrary, addresses. For these reasons we assume that in a practical scenario the attacker cannot leak information that is not located on the heap.

Most real-world attacks are based on ROP. To execute a ROP payload, the attacker needs to either inject his payload directly on the stack, or use a stack-pivot gadget to overwrite the stack pointer with an address that points to the ROP payload on the heap. As mentioned previously, the attacker usually has no access to the stack. Hence, the first gadget in the ROP chain is normally a stack-pivot. [6]

ROP is merely used to bypass $W \oplus X$ policies and enable code injection, i.e., a small ROP payload is used to (1) mark the data memory containing the shellcode as executable and (2) branch to the shellcode. The shellcode will then perform the actual task of de-anonymizing the user or installing surveillance software. To mark a data page as executable, only a single system call is needed. Hence, the attacker requires only gadgets that load the arguments for the system call into the registers, then issue a system call and return to the shellcode. [6]

Evaluation Result:

The analysis shows found ten stack-pivot and 76 system call gadgets of which only 4 and 29 respectively are available through virtual functions whose addresses are exposed on the heap through indirection tables called virtual tables. Memory objects on the heap contain a pointer to a virtual table which is located in the code or data section of the application and contains a number of pointers to virtual functions. Since the attackers can only disclose the virtual table pointer, but not the virtual table itself, as it is not on the heap, they cannot disclose gadget addresses. Note that, when only ASLR is applied, the address of the virtual table is randomized with the same offset as the ROP gadgets. Therefore, such an attack can bypass ASLR but not selfrando [9] [7] .

8.2 Performance Overhead

8.3 Load-time Overhead

The load time of TB is measured by inserting a return statement in the main function, after the dynamic libraries are loaded but before the program actually does anything.

The average load time for the normal version was 2.046s, while the selfrando version took 2.400s on average. The average overhead is 354ms. This is an acceptable overhead considering the improved protection against deanonymization attacks. [9]

8.4 Run-time Overhead

To test the run-time overhead of selfrando, the SPEC CPU2006 benchmark suite as well as a number of modern JavaScript benchmarks are used. The benchmarks are run with a version of selfrando that always chooses the original order for the randomization (identity transformation). This version runs all the load-time code but it does not actually modify the code segment. It allows us to distinguish between load-time overhead and run-time overhead. We ran each benchmark three times with the ref workload. The reported figures are the median values. The result shows that geometric mean of the overheads is 2.02%, while the worst overhead is 2.5%. [9]

9 Conclusion

Securing the Tor network from de-anonymization attack is necessary to preserve privacy. Selfrando, an enhanced and practical load-time randomization technique for the Tor Browser defends against de-anonymization exploits.It has negligible run-time overhead,and a perfectly acceptable load-time overhead. Selfrando significantly improves security over standard address space layout randomization (ASLR) techniques currently used.

References

[1] Tor overview. *https://www.TorProject.org.*

[2] How tor works – a compute cycle deep dive. *https://www.excivity.com/ComputeCycle/howtorworks/.*

[3] Did the fbi pay a university to attck tor users? *https://blog.torproject.org/blog/did-fbi-pay-university-attack-tor-users.*

[4] Data execution prevention (DEP). Microsoft. *http://support.microsoft.com/kb/875352/EN-US/.*

[5] M.Tran, M.Etheridge, T.Bletsch, X.Jiang, V.W.Freeh, and P.Ning. On the expressiveness of return-into-libc attacks. *14th International Symposium on Research in Attacks, Intrusions and Defenses, 2011.*

[6] Erik Buchanan, Ryan Roemer,Stefan Savage, Hovav Shacham. Return-oriented programming: Exploitation without code injection. *University of California, San Diego.*

[7] , Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, Dan Boneh Hovav Shacham, Matthew Page. On the effectiveness of address-space randomization.

[8] C.Kil, J.Jun, C.Bookholt, J.Xu, and P.Ning. Address space layout permutation (aslp): Towards fine-grained randomization of commodity software. *22nd Annual Computer Security Applications Conference, 2006.*

[9] Mauro Conti, Stephen Crane, Tommaso Frassetto, Andrei Homescu, Georg Koppen, Per Larsen, Christopher Liebchen, Mike Perry, and Ahmad-Reza Sadeghi. Selfrando: securing the tor browser against de-anonymization exploits. *The annual Privacy Enhancing Technologies Symposium in Darmstadt, Germany, 2016.*

[10] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, Dmitry Vyukov. Addresssanitizer: A fast address sanity checker.

[11] Selfrando: Q and a with georg koppen. *http://www.TorProject.org.*