***Sniper3*** *–*Users usually choose a select few entrance guards, starting their circuits at one of these guards, which helps prevent direct observation. To prevent assaults, guards are often rotated. The Sniper Attack, on the other hand, concentrates on taking out a user's guards, forcing them to select new ones, maybe exposing them to hostile relays, and therefore needing continual guard changes. As a result of their on-demand circuit creation, hidden services are particularly concerned about this method since it makes an attack more practical.

In conclusion, the Sniper Attack takes use of Tor's architecture to launch a cheap yet effective denial-of-service attack. The attacker seeks to disrupt both the availability of Tor and the anonymity of its users by repeatedly focusing on a user's entry guards and forcing guard substitutions; hidden services are especially susceptible to this kind of assault. Users and maintainers of Tor must be constantly on guard against such attacks and make sure the continued security and effectiveness of the network.

**SNIPER ATTACK DEFENSE -** A simple but naïve defence against the Sniper Attack is to have the guard node monitor its queue length and destroy the circuit if it ever exceeds 1000 cells. This defence does not prevent the opponent from parallelizing the assault by employing numerous circuits (and then consuming 1000 cells on each), as we have demonstrated to be exceedingly successful.

As they advised to the Tor developers, the best defence is to construct a bespoke, adaptive out-of-memory circuit killer in application space (i.e., within Tor). When memory becomes scarce, the circuit killer is engaged, and it selects the circuit with the oldest front-most cell in its circuit queue. This will stop the Sniper Attack by destroying all attack circuits.

With this new defence in place, the adversary's next move is to try to cause Tor to kill an honest circuit. An adversary must guarantee that the front-most cell on its malicious circuit queue is at least marginally "younger" than the oldest cell on any honest queue in order to cause an honest circuit to be terminated. With this defence, we demonstrate that the Sniper assault is impractical: owing to Tor's fairness features, the adversary must spend an incredible amount of bandwidth keeping its cells fresh – bandwidth that would likely be better served by a more classic brute-force DoS attack.

***Sniper1*** –

Three basic elements make up the network: the Client, the Server, and the Network itself. Any organization or individual looking for information or services plays the part of the client. The system in charge of giving the Client the information or services they have requested is referred to as the Server in contrast. The "Network," which includes both the communication routes and the Client-Server pair, serves the collective purpose of enabling the data transfer between the Client and Server to occur seamlessly. The Sniper attack is a noteworthy illustration of an attack within this architecture, which differs from single-node-focused attacks by having a network-wide impact.

In this attack, the attacker takes use of Tor's flow control algorithm to attack a target Tor relay with a denial-of-service attack, flooding it with data until it shuts down. In order to reduce network performance and raise the possibility that a client may choose the attacker's node, the attacker may use this technique across numerous nodes, further exposing the network's vulnerability.

***Sniper2*** *–*

A diagram of data and data

Description automatically generatedThe Sniper attack, one of these attacks, tries to disable each Tor relay it encounters. As it involves deanonymizing hidden services by keeping track of cell counts at an adversarial guard node, this attack, also known as the Sniper attack, is included in the category of cell count attacks. The steps that make up the sniper attack happen in the following order: The target relay is used as the entry point in the initial step of a client establishing a circuit. The server then responds with the requested data after receiving a request for it from the client. The client then stops actively reading from the TCP connection to the target entry, but it still occasionally sends SENDME packets to the server without stopping to read any data from it. The target entry stores the buffered data up until the operating system kills the Tor process, ultimately making the attack successful and jeopardizing the hidden service's anonymity.

The following flowchart illustrates the strategic processes involved in the deanonymization of hidden services:

Building Rendezvous Circuits: To reveal the hidden service's guard node, the adversary forces the HS to build new rendezvous circuits to start the deanonymization process.

Sniping the HS Guard: The adversary employs a Snipe attack to coerce the HS into reselecting its guard node. The HS chooses an antagonistic guard node after several iterations of this process.

Reselection of the Guard Node: By repeatedly executing the Snipe attack, the attacker continually persuades the HS to choose one of the adversarial relays as its guard node, a critical step towards deanonymization.

Building Circuit to RP: After selecting an adversarial guard node, the HS creates a brand-new circuit to an RP.

A diagram of a computer program

Description automatically generatedPadding Cell Infiltration: In order to alter the cell counter, the adversary client sends 50 padding cells to the HS via the adversarial guard node (GA).

Detecting Entry Node: The attacker can determine the HS's entry node by keeping an eye on the cell counter at the adversary-controlled guard relay (GA), which increases to 52.

Sniper Attack on Entry Node: Using this knowledge, the adversary launches a Sniper attack against the HS's entry node, causing the HS to change the choice of its guard node.

Adversarial Guard Verification: The procedure is cycled through until the HS chooses an adversarial guard. The adversary client sends 50 padding cells to the HS to check whether a compromised guard node has been selected.

Identifying Hidden Service: The attacker locates the hidden service by tracking the cell counter at the adversarial guard.

A table of information

Description automatically generatedComparison table of attacking schemes –

**Securing TOR against Deanonymization exploits:**

**SLF1 –**

A diagram of a computer process

Description automatically generatedCompared to ASLR (Address Space Layout Randomization), Selfrando significantly increases entropy while still being compatible with standard software build and distribution procedures, unlike compile-time randomization techniques. Because it made code-reuse attacks more complex while causing little performance impact, ASLR was an essential first step in preventing memory-corruption issues. But ASLR has some significant drawbacks. On some platforms, it shows poor entropy, but Selfrando offers higher entropy. Second, if an attacker discovers a memory pointer (such as one that points to executable code), they can use straightforward arithmetic to determine the addresses of other regions of that memory area. Selfrando rearranges memory based on functions, operating at the function granularity. As a result, if a pointer is leaked by an enemy, it just divulges information about that particular function, without disclosing the location of the other functions.

**SLF2 -**

A diagram of a diagram

Description automatically generated Let's examine four levels of granularity for defences: (i) library, (ii) function, (iii) basicblock, and (iv) instruction. Different randomization granularities involve different trade-offs between security and performance. Better security is achieved with finer-granularity randomization, but at the possible cost of increased performance overhead.

A brief overview of each evaluated defense and how it is implemented:

The location of entire shared libraries can be changed with the coarsest granularity we can think of, ASLR. On the majority of computers, it is enabled by default and randomizes once during load.

To enable function-level randomization by the loader, Selfrando tweaks the LLVM compiler and the system loader. Since all binaries have distinct runtime memory configurations, they are all identical on disc. At loadtime, randomization only happens once.

In order to preserve symbolization information, CCR alters the LLVM compiler. Binaries are then randomly generated by a different shuffler, which fixes addresses in jump instructions following randomization using symbolization data. A randomized binary can be created on disc by running the shuffler either at install time or at load time.

Additionally an LLVM enhancement is multicompiler. In this instance, the compiler randomly inserts zero or more NOPs (no operations) between instructions. This is the most finely grained defence we take into consideration because it alters the address of certain instructions. Randomization happens during compilation rather than load. It does not randomize as frequently as our other defences, despite having finer granularity.

|  |  |
| --- | --- |
| Sniper3 | The Sniper attack is a potentially damaging denial-of-service attack meant to disable the Tor network, which protects users' online anonymity. In this technique, the attacker makes use of the reliable data transport provided by Tor to flood Tor relays with valid protocol messages, using a sizable amount of RAM. This puts users' anonymity at risk in addition to threatening Tor's availability. A hostile actor may be able to compromise a user's anonymity by manipulating entry and exit relays, which might possibly correlate with a user's entering and exiting communication timestamps. |
| Sniper1 | This comprehensive review examines the corpus of available research on attacks on the Tor network and emphasizes the most relevant dangers in this context. The paper presents a thorough taxonomy that enables the detection of cyberattacks targeting darknet settings by matching the nature of these attacks with their intended aims. This taxonomy is extremely helpful in identifying such assaults and improving our understanding of how they operate. |
| Sniper2 | In terms of the security of the Tor network, this paper provides a thorough analysis of the attack techniques currently in use. It examines these plans methodically, highlighting the fundamental ideas behind each and showing how they are related. The comparative approach, which emphasizes simplicity and effectiveness, offers a useful framework for comprehending the deanonymization of hidden services. |
| SLF1 | The paper offers Selfrando, a revolutionary method that increases program security by randomly generating small portions of executable code. Selfrando, in contrast to earlier techniques, manages code randomization every time an application is launched, producing different memory layouts for every run while utilizing the same application package. Through normal channels, this method allows for the dissemination of a uniform application package, but each execution shows a unique memory configuration. The Tor Browser, a web browser that prioritizes privacy, successfully integrated Selfrando. |
| SLF2 | This paper explores the application of moving-target defenses in the context of real-time systems. To investigate the ramifications of such designs in real-time applications, the worst-case performance of a number of address-space randomization defences is assessed. These findings imply that current moving-target defences, while effective in the average situation, can have considerable tail latencies. These tail latencies can be troublesome in real-time applications, especially if such overheads are not taken into account during the system's design and analysis. These findings guide future study on moving-target defences in real-time applications. |